



Denodo MongoDB Custom Wrapper

Revision 20160107

NOTE

This document is confidential and proprietary of **Denodo Technologies**.
No part of this document may be reproduced in any form by any means without prior written authorization of **Denodo Technologies**.

Copyright © 2016
Denodo Technologies Proprietary and Confidential

CONTENTS

1 INTRODUCTION.....	3
2 WHAT IS MONGODB?.....	4
3 ARCHITECTURE AND FEATURES.....	5
4 USAGE.....	7
4.1 COMPATIBILITY.....	7
4.2 IMPORTING THE CUSTOM WRAPPER INTO VDP.....	7
4.3 CREATING A MONGODB DATA SOURCE.....	8
4.4 CREATING A BASE VIEW.....	9
4.5 CREATE BASE VIEW EXAMPLE.....	10
4.6 CREATE BASE VIEW USING INTROSPECTION QUERY.....	13
4.7 QUERYING MONGODB COLLECTIONS.....	14
4.8 INSERT.....	14
4.9 UPDATE.....	14
4.10 DELETE.....	15
6 LIMITATIONS.....	17

1 INTRODUCTION

mongodb-customwrapper is a Virtual DataPort custom wrapper for querying MongoDB collections.

It bridges (to some extent) the gap between NoSQL and relational databases by establishing a predefined schema for output, thus enabling SQL queries on MongoDB.

2 WHAT IS MONGODB?

[MongoDB](#) is one of the most prominent so-called NoSQL databases, although it would be more accurate to say that is a non-relational database, or as defined on their website: "A document-oriented, scalable and high performance database", developed by 10gen.

In MongoDB we can forget about the table: we have **collections** which are groups of **documents** that do not have to share any common schema. So documents are to MongoDB something very similar to what tuples represent to relational databases, only without the schema restrictions associated with the latter.

Another important feature is the way in which MongoDB stores documents: **BSON** (Binary JSON). The binary part is hidden from us when we work with MongoDB, and thus we see and treat these data in JSON format, so our documents are in fact JSON objects, and its **fields** are the equivalent to the columns of the relational world. But as we do not need the documents in a collection to share a common schema, each JSON document can have just the fields it needs, and all documents in a collection do not have to represent entities of the same nature.

And you can also forget about both the SQL language and the JDBC API. MongoDB does not use SQL but its own query API, and consequently offers a specialized driver for Java that acts as a client and interacts with the database.

A **database** consists of one or more collections, the documents in those collections, and an optional set of security credentials for controlling access.

3 ARCHITECTURE AND FEATURES

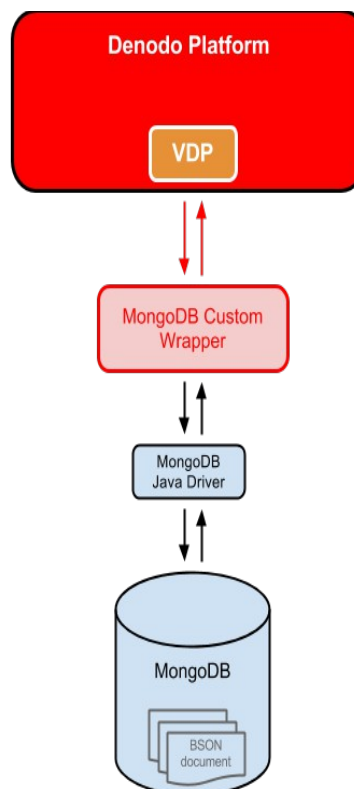
mongodb-customwrapper allows us to create base views on MongoDB collections and execute SQL queries on those collections.

The wrapper uses the official MongoDB API for Java Driver that is provided by 10gen.

The wrapper uses this driver to get a MongoClient object that represents a pool of connections to a MongoDB database. Optionally it can authenticate by user and password.

After making the connection, the custom wrapper can create base views to read collections and documents as if they were tables and rows. Each base view will be created on a specific collection.

Once the base views are created, VDP will be able to perform queries on MongoDB using the VQL Shell, create derived views, etc.



MongoDB Custom Wrapper Architecture

This is a brief summary of the wrapper's current features:

- User/password authentication is supported. User name and password can be specified as input values when building the base view.
- Base view result schema is defined as a comma-separated list of field names.
 - Extra fields in returned documents are ignored.
 - Lacking fields in returned documents are returned as null.
 - Types can be specified (from java.sql.Types).
 - Default is VARCHAR.
 - If documents in the same collection specify different types for fields with the same name, using VARCHAR (text in VDP) for that column will automatically perform the required conversions to show data from those documents.
- Alternatively the schema could be defined using an introspection query. **All documents** (remember, "schemaless") retrieved by this query are analyzed to reveal their fields and build the base view schema. For this reason the query should retrieve a significant sample of the collection we are interested in.
 - An introspection query should be used when treating with complex fields that holds arrays and subdocuments.
 - Since common fields may hold different types of data the resulting structure is the highest common denominator between all the fields with the same name. In case of incompatible fields --such as integer and subdocuments-- VDP interprets them as of type text. But notice that **MongoDB is strict about types** and you must query for data using the correct type, so this fields would not be searchable.
- Create/update operations using simple fields are supported.
- Delete operations are supported.
- Field projections are delegated to MongoDB.
- Some query conditions are delegated to MongoDB.
 - AND and OR conditions.
 - Operators: =, <>, <, >, <=, >=, like, IS NULL, IS NOT NULL.
 - IS NULL operator gets the documents where the specified field has the null value or does not exist.
- ORDER BY clause is delegated to MongoDB.

4 USAGE

4.1 COMPATIBILITY

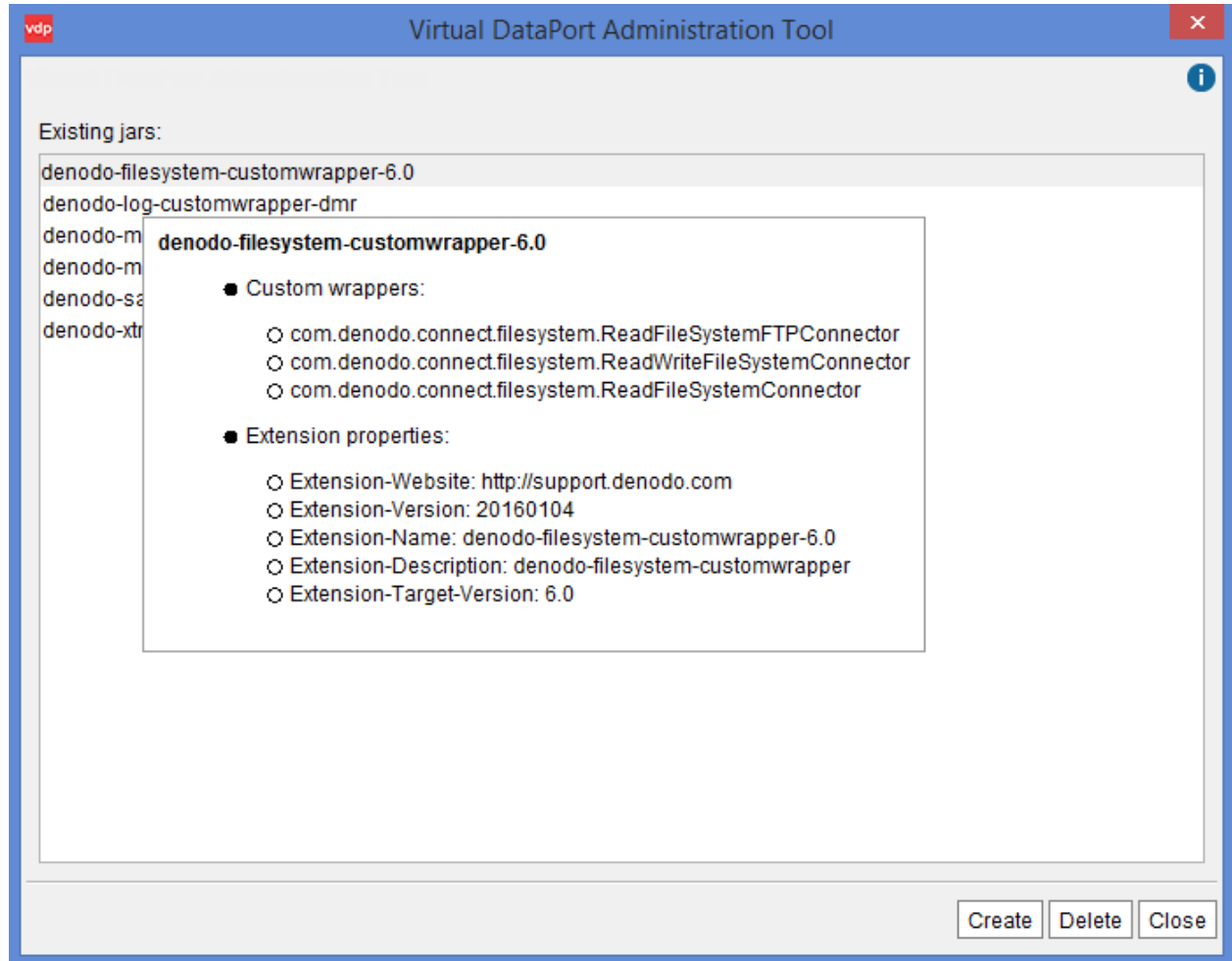
The driver does not support older versions of 2.4 MongoDB.

4.2 IMPORTING THE CUSTOM WRAPPER INTO VDP

In order to use the MongoDB Custom Wrapper in VDP, we must configure the Admin Tool to import the extension.

From the `denodo-mongodb-customwrapper` distribution, we will select the `denodo-mongodb-customwrapper-${version}-jar-with-dependencies.jar` file and upload it to VDP.

No other jars are required as this one will already contain all the required dependencies, including the MongoDB Java driver classes.

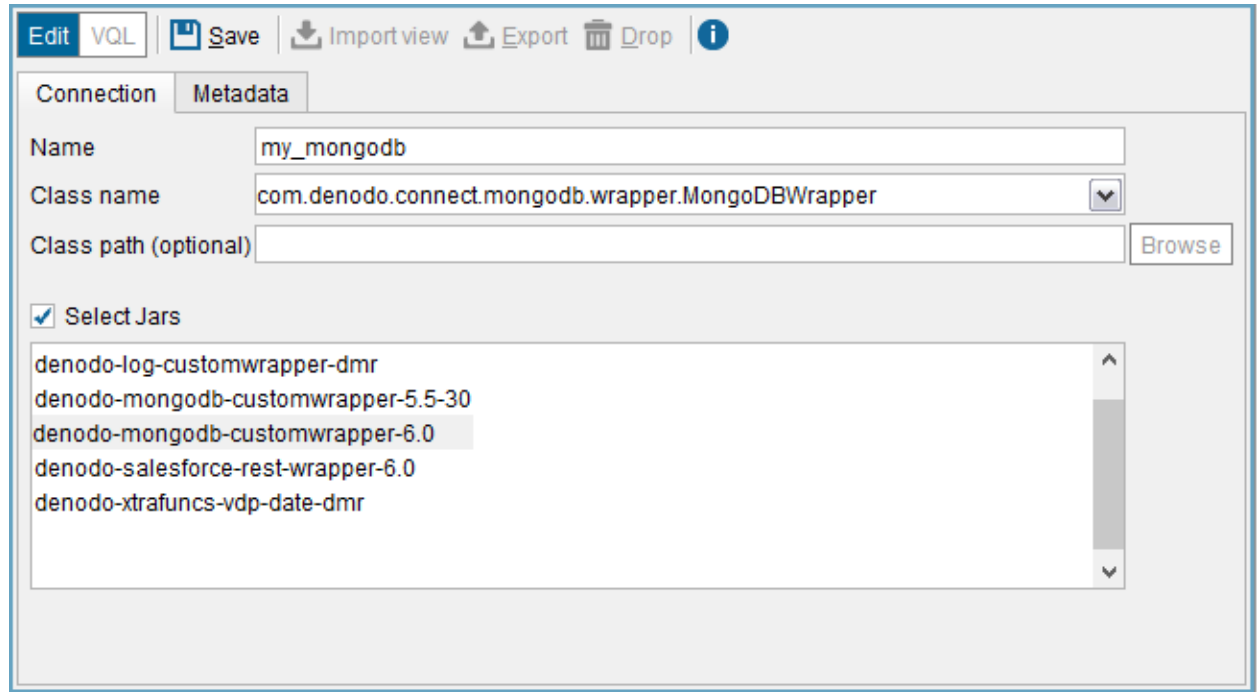


MongoDB Extension in VDP

4.3 CREATING A MONGODB DATA SOURCE

Once the custom wrapper jar file has been uploaded to VDP using the Admin Tool, we can create new data sources for this custom wrapper --and their corresponding base views-- as usual.

Go to New → Data Source → Custom and specify the wrapper's class name `com.denodo.connect.mongodb.wrapper.MongoDBWrapper`. Also check 'Select Jars' and select the jar file of the custom wrapper.



MongoDB Data Source

4.4 CREATING A BASE VIEW

Once the custom wrapper has been registered, we will be asked by VDP to create a base view for it.

Base views created from the MongoDBWrapper need to fill in the Database parameter or the Connection string parameter, both of them are mutually exclusive. Moreover Collection is mandatory

- Database: Database name.
- Connection String: this parameter allows more options in the connection between the driver and mongodb. This is the pattern: [mongodb://]host1[:port1][,host2[:port2],...[,hostN[:portN]]]/database[?options]. The prefix mongodb:// is optional, the database is mandatory and user and password are not written in this field, but in the User and Password parameters
You can see all the options in the documentation of mongodb, in the following link about [Connection String URI Format](#).
If you introduce this parameter, host, port and database should be empty.
- Collection: Collection name that we will import as a table in VDP.

Four **optional** parameters:

- Host: Name of the computer or IP address where MongoDB is running, default is 127.0.0.1. Only with Database parameter.
- Port: Port number to connect to MongoDB, default is 27017. Only with Database parameter.
- User/Password: Username and password to connect to MongoDB, if the authentication is enabled.

There are also two parameters that are **mutually exclusive**:

- Fields: The fields we would like to import as columns in VDP. We must keep the syntax `field1[:type1][,field2[:type2],...]`. Type, when specified, should be one of `java.sql.Types`.
- Introspection query: Documents retrieved by this query will be analyzed to reveal their fields and build the view schema. An empty query selects all documents in the collection. This query requires **MongoDB syntax**.

4.5 CREATE BASE VIEW EXAMPLE

In the following example we want to import a product catalog collection of an E-Commerce site database.

At the beginning of the document, the schema must contain general product information, to facilitate searches of the entire catalog. Then, a details subdocument that contains fields that vary between product types.

As we are only interested in albums and films products we use the following introspection query:

```
{ type: { $in: [ 'Audio Album', 'Film' ] } } )
```

Edit Wrapper Parameter values

Enter values for the following wrapper parameters:

Host	<input type="text" value="127.0.0.1"/>
Port	<input type="text" value="27017"/>
User	<input type="text"/>
Password	<input type="text"/>
Database	<input type="text" value="products"/>
Collection	<input type="text" value="product"/>
Connection String	<input type="text"/>
Fields	<input type="text"/>
Introspection query	<input type="text" value="{type:{\$in:['Audio Album','Film']}}"/>

Ok Cancel

MongoDB Base View edition using Host, Port and Database parameters

Edit Wrapper Parameter values

Enter values for the following wrapper parameters:

Host	<input type="text"/>
Port	<input type="text"/>
User	<input type="text"/>
Password	<input type="text"/>
Database	<input type="text"/>
Collection	<input type="text" value="product"/>
Connection String	<input type="text" value="mongodb://127.0.0.1:27017/products"/>
Fields	<input type="text"/>
Introspection query	<input type="text" value="{type:{\$in:['Audio Album','Film']}}"/>

Ok Cancel

MongoDB Base View edition using Connection String parameter

The resulting schema for this product catalog collection can be seen in the image below. It contains common product information like title, type, pricing and the details subdocument

View schema			Metadata
View name: my_mongodb			
<input type="checkbox"/> PK	Field Name	Field Type	
<input type="checkbox"/>	id_0	text	
<input type="checkbox"/>	sku	text	
<input type="checkbox"/>	type	text	
<input type="checkbox"/>	title	text	
<input type="checkbox"/>	description	text	
<input type="checkbox"/>	publisher	text	
<input type="checkbox"/>	pricing	my_mongodb_pricing	
	list	text	
	retail	text	
<input type="checkbox"/>	details	my_mongodb_details	

Set selected as PK

MongoDB Base View

Execute

Query Results

Results

Execution Trace

Stop

Refresh

Save


adb AS my_mongodb WHERE type <> 'Book' CONTEXT ('i18n'='us_pst', 'cache_wait_for_load'='true') TRACE

Total rows received: 2 (shown 2)

id_0	sku	type	title	description	publisher	pricing	details
568bb83dd292501...	1000001	Audio Album	A Love Supreme	by John Coltrane	Sony Music	[Register]...	[Register]...
568bb83ed292501...	1000002	Audio Album	Love Song	by Khali Fong	Sony Music	[Register]...	[Register]...

MongoDB Base View execution

Results Execution Trace			
Is FROM my_mongodb AS my_mongodb WHERE type <=> 'Book' CONTEXT ('i18n'='us_pst', 'cache_wait_for_load'=true) TRACE			
Total rows received: 2 (shown 2)			
RESU... -> details			
title	artist	genre	tracks
A Love Supreme [Original Recording Rei...	John Coltrane	Jazz	[Array]...



[RESU... -> details -> tracks](#)

tracks_item
A Love Supreme Part I: Acknowledgement
A Love Supreme Part II - Resolution
A Love Supreme, Part III: Pursuance
A Love Supreme, Part IV-Psalm

Array and record details of MongoDB Base View execution

4.6 CREATE BASE VIEW USING INTROSPECTION QUERY

If you want to treat with complex fields, you should use the Introspection query parameter. For this you should use a query over the collection, following the syntax for the mongodb method db.collection.find(). You should escape the braces.

A example of introspection query is :

```
\{ "properties": \{"LINEARID": "1101121687137", "observed": "",
"COUNTYFP": "051", "RTTYP": "M", "FULLNAME": "Airport Rd", "MTFCC":
"S1400", "STATEFP": "35"\}\}
```

Edit Wrapper Parameter values

Enter values for the following wrapper parameters:

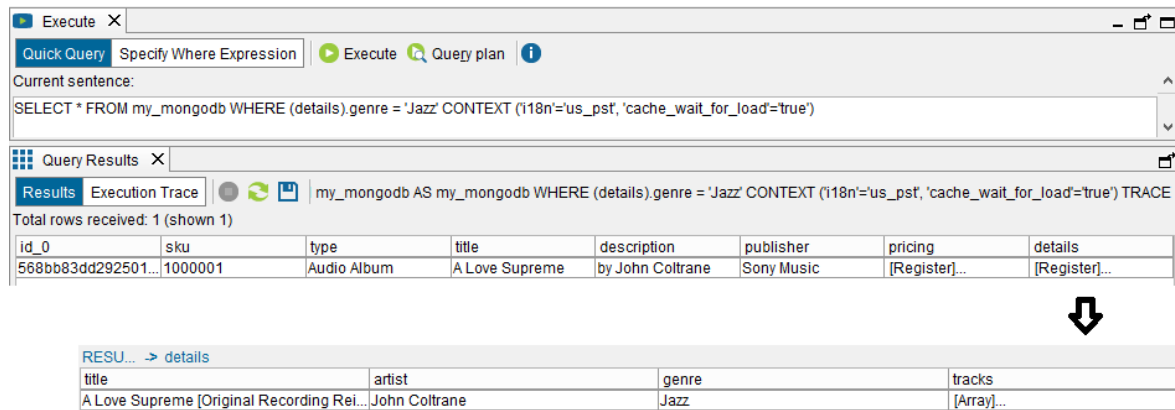
Host	<input type="text"/>
Port	<input type="text"/>
User	<input type="text"/>
Password	<input type="text"/>
Database	<input type="text"/>
Collection	<input type="text" value="roads"/>
Connection String	<input type="text" value="mongodb://127.0.0.1:27017/test"/>
Fields	<input type="text"/>
Introspection query	<input type="text" value='\{"properties": \{"LINEARID": 1101121687137\}\}'/>

Ok
Cancel

MongoDB Base View edition using Introspection query parameter

4.7 QUERYING MONGODB COLLECTIONS

From now on we can query using the VQL Shell tool of the Admin Tool, create derived views, etc.

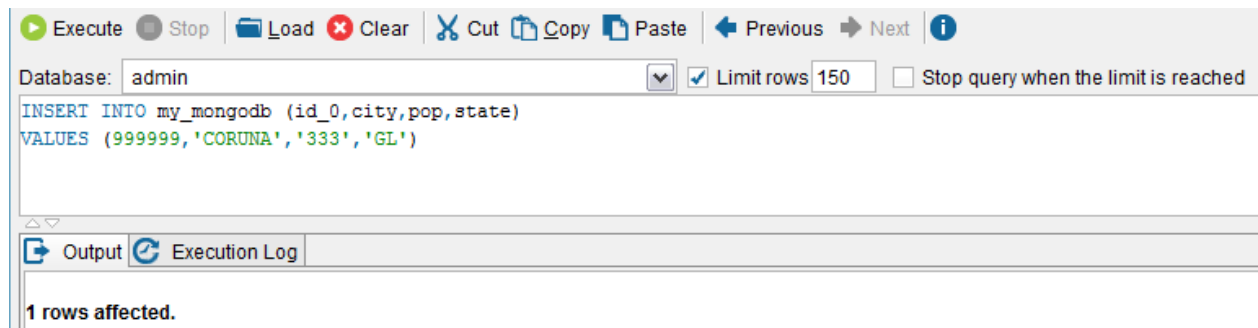


The screenshot shows the Denodo Admin Tool interface. The top section is the 'Execute' tab, where a VQL query is entered: `SELECT * FROM my_mongodb WHERE (details).genre = 'Jazz' CONTEXT ('i18n'='us_pst', 'cache_wait_for_load'='true')`. Below the query, the 'Query Results' tab is active, showing a table with 8 columns: `id_0`, `sku`, `type`, `title`, `description`, `publisher`, `pricing`, and `details`. The table contains one row of data. A downward arrow points to a detailed view of the 'details' field, which shows a table with 4 columns: `title`, `artist`, `genre`, and `tracks`. The data row shows 'A Love Supreme [Original Recording Rei...', 'John Coltrane', 'Jazz', and '[Array]...'.

Querying a MongoDB View

4.8 INSERT

This wrapper allows to make insertion with not complex fields



The screenshot shows the Denodo Admin Tool interface. The top section is the 'Execute' tab, where an INSERT query is entered: `INSERT INTO my_mongodb (id_0,city,pop,state) VALUES (999999,'CORUNA','333','GL')`. Below the query, the 'Output' tab is active, showing the message: '1 rows affected.'

4.9 UPDATE

This wrapper allows to make updates in a document with not complex fields

The screenshot shows the Denodo MongoDB Custom Wrapper interface. At the top, there is a toolbar with buttons: Execute (green play icon), Stop (grey circle icon), Load (blue folder icon), Clear (red X icon), Cut (blue scissors icon), Copy (blue document icon), Paste (blue document icon), Previous (blue left arrow icon), Next (blue right arrow icon), and an information icon (blue i icon). Below the toolbar, the Database is set to 'admin'. There are checkboxes for 'Limit rows' (checked, set to 150) and 'Stop query when the limit is reached' (unchecked). The SQL query entered is:

```
UPDATE my_mongodb
SET (city)=('Ferrol')
WHERE (id_0=999999);
```

Below the query editor, there are tabs for 'Output' and 'Execution Log'. The 'Output' tab is selected, showing the result: '1 rows affected.'

4.10 DELETE

It is allowed to delete documents of a collection

The screenshot shows the Denodo MongoDB Custom Wrapper interface. At the top, there is a toolbar with buttons: Execute (green play icon), Stop (grey circle icon), Load (blue folder icon), Clear (red X icon), Cut (blue scissors icon), Copy (blue document icon), Paste (blue document icon), Previous (blue left arrow icon), Next (blue right arrow icon), and an information icon (blue i icon). Below the toolbar, the Database is set to 'admin'. There are checkboxes for 'Limit rows' (checked, set to 150) and 'Stop query when the limit is reached' (unchecked). The SQL query entered is:

```
DELETE FROM my_mongodb
WHERE (id_0=999999);
```

Below the query editor, there are tabs for 'Output' and 'Execution Log'. The 'Output' tab is selected, showing the result: '1 rows affected.'

6 LIMITATIONS

Write operations

Create/update operations with complex fields, arrays and registers, on the MongoDB server are not supported through this custom wrapper at its current version.