### **EvolveDB - Documentation for Developer**

Author: E-Mail:	Torben Eckwert, M.Sc. torben.eckwert@zdh.thm.de	
Subject Area:	Computer Science	
Supervisors:	Prof. Dr. rer. nat. Michael Guckert Prof. Dr. ing. Gabriele Taentzer	

December 8, 2022, Wetzlar



# FORSCHUNGSCAMPUS MITTELHESSEN

## Contents

1	Introduction	1
2	Prerequisites and Installation	1
3	EvolveDB3.1Create a new datasource3.2Create a new migration script generator	<b>3</b> 4 11
4	Metamodel4.1MDDE metamodel4.2Migration metamodel	<b>18</b> 18 19

### **1** Introduction

EvolveDB is an Eclipse-based framework for schema evolution in relational databases. The user specifies the evolution steps by freely editing a database model extracted by reverse engineering. EvolveDB analyzes the differences between the status quo and the evolved model structures and generates a data migration script. This user manual includes installation instructions and introductory tutorials on how to use EvolveDB as an end user.

### 2 Prerequisites and Installation

EvolveDB is a plug-in for recent versions of the Eclipse Modeling Tools (last tested version: 2022-06). EvolveDB is open source and licensed with the Apache license 2.0. The source code is available via our GitHub repository.

Before downloading the source code of EvolveDB, we need to install some additional Eclipse plugins. The following list contains the mandatory plugins.

- Eclipse OCL is available via the eclipse marketplace.
- The Henshin project provides a state-of-the-art model transformation language for the Eclipse Modeling Framework.
- ATL 3.5 (ATL Transformation Language) is a model transformation language and toolkit.
- ATL/EMFTVM 4.2.1 The EMF Transformation Virtual Machine (EMFTVM) is a runtime engine for the ATL Transformation Language (ATL).
- Xtend is a statically-typed programming languages for Java developers.

SiLift is a generic model comparison environment for EMF-based models. The SiLift update site is no longer available, but SiLift is included in the release version of EvolveDB. Help  $\Rightarrow$  Install New Software... (fig. 1).

2 Prerequisites and Installation



Figure 1: Eclipse: Install New Software...

The repository contains multiple categories (fig. 2). For installing SiLift, select *SiLift* and *SiLift Matcher*.

Install			
Available Software			
Check the items that you wish to install.			
Work with: EvolveDB local repo - file:/F:/EvolveDB/2.20.5/	~	Add	Manage
type filter text			Select All
Name	Version		Deselect All
→ w Henshin			
> V IIII Hensnin Patch			
> V III Nebula Grid			
> 🔽 🚥 SiLift			
> 🔽 💷 SiLift Matcher			
14 items selected			
Details			
			0
	Hide items that are already installed		
Group items by category	What is <u>already installed</u> ?		
Show only software applicable to target environment			
Contact all update sites during install to find required software			
?	< Back Next >	Finish	Cancel

Figure 2: Eclipse: EvolveDB local repository

If all requirements are fulfilled, the source code of EvolveDB can be cloned and imported.

### **3** EvolveDB

The schema evolution with EvolveDB is a model-driven reengineering process that includes three phases, reverse engineering, restructuring, and forward engineering. In the first step, we start by creating a representation of a database schema through reverse engineering. In the restructuring phase, we edit this model resulting in a second model version. Afterwards, EvolveDB analyzes the differences between the status quo and the evolved model structures. From the delta between the two model versions, SQL migration scripts are generated, which can be used to migrate the schema and the associated data. EvolveDB includes a data source and a migration script generator which supports MySQL databases. If other data sources or generators should be used, EvolveDB provides two extension points. By contributing to these extension points, custom data sources or generators can be integrated.

#### 3.1 Create a new datasource

As already mentioned, custom drivers can be integrated into EvolveDB. To do this, create a plug-in via **File**  $\Rightarrow$  **New**  $\Rightarrow$  **Other**  $\Rightarrow$  **Plug-in Development**  $\Rightarrow$  **Plug-in Project** and open the MANIFEST.MF file. Switch to the tab *Dependencies* and add the dependencies marked in Figure 4.

CNew Plug-in Proj	ect		—		×
Plug-in Project Create a new plug-i	n project				
Project name: Test	Datasource				
Use default loca	tion				
Location: C:\Users\	Torben\Documents\	Promotion_Feature_	Test\workspace\Tes	Browse	
Project Settings	oject				
Source folder:	src				
Output folder:	bin				
Target Platform This plug-in is targe © Eclipse O an OSGi frar	eted to run with:	~			
Working sets					
Add project to	working sets			New	
Working sets:			~	Select	
?	< Back	Next >	Finish	Cancel	

Figure 3: Eclipse: Create New Plug-in Project

equired Plug-ins	↓ªZ	Imported Packages	
pecify the list of plug-ins required for lug-in.	the operation of this	Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.	
b de.thm.mdde.datasource	Add	Add	
de.thm.mdde.connection org.eclipse.swt	Remove	Remo	ve
org.eclipse.emf.edit.ui	Up	Propert	es
	Down		
	Properties		
	Total: 4	Total: 0	

Figure 4: Plug-in Project Manifest.mf

Next we have to create a class that implements the *EDBDataSource* (fig. 5) interface. Figure 6 shows the class TestDatasource that implements the interface.

```
import org.eclipse.emf.ecore.EObject;[]
public interface EDBDataSource {
    /**
    * <u>Datasource</u> name
    * @return the name of the datasource
    */
    String getName();
    /**
    * Connection icon
     * @return Database Icon
    */
    Image getImage();
    /**
    * DBPDriver class
    * @return Driver class
    */
    DBPDriver getDriver();
    /**
    * DBPDriverDependencies
    * @return the corresponding driver dependencies
    */
    DBPDriverDependencies getDriverDependencies();
    /**
    * The UI for entering the connection details.
    * @return
    */
    void openConnectionUi();
    /**
    * Returns the root element of the newly created model;
    * @return
    */
    EObject getRootObject();
    /**
    \ast Returns true if the user left the wizard via the cancel button.
     * @return
     */
    boolean isCanceled();
```

}



```
🚯 TestDatasource 🛛 🛽 TestDatasource.java 🗡
    package de.thm.mdde.datasource;
  1
  2
  3@ import org.eclipse.emf.ecore.EObject;
  4 import org.eclipse.swt.graphics.Image;
  6 import de.thm.mdde.connection.model.DBPDriver;
  7 import de.thm.mdde.connection.model.DBPDriverDependencies;
  8
  9 public class TestDatasource implements EDBDataSource{
 10
 <u>11</u>Θ
         @Override
≏12
         public DBPDriver getDriver() {
a13
             // TODO Auto-generated method stub
 14
             return null;
 15
         }
 16
 17⊝
         @Override
≏18
         public DBPDriverDependencies getDriverDependencies() {
219
            // TODO Auto-generated method stub
 20
             return null;
 21
         }
 22
 23⊝
         @Override
 24
         public Image getImage() {
225
            // TODO Auto-generated method stub
 26
             return null;
 27
         }
 28
 29⊝
         @Override
△30
         public String getName() {
             // TODO Auto-generated method stub
231
 32
             return "TestDriver";
 33
         }
 34
 35⊝
         @Override
≏36
         public EObject getRootObject() {
237
             // TODO Auto-generated method stub
 38
             return null;
 39
         }
 40
 41⊝
         @Override
≏42
         public boolean isCanceled() {
43
             // TODO Auto-generated method stub
 44
             return false;
 45
         }
 46
 47⊝
         @Override
≏48
         public void openConnectionUi() {
249
             // TODO Auto-generated method stub
 50
 51
         }
 52
53 }
54
```

Figure 6: Class TestDatasource

Next we have to add the new class as an extension for EvolveDB. Switch back to the *MANIFEST.MF* and open the Extension tab. Click on add and select the extension point *de.thm.mdde.datasource*. (fig. 7)

New Extension			×
Extension Point Selection		=0	
Extension Points Extension Wizards			
Extension Point filter: de.th			
- de.thm.mdde.datasource			
Show only extension points from the required plug-ins Extension Point Description:			
(no description available)			
Available templates:			
? < Back Next > Finish		Cancel	

Figure 7: de.thm.mdde.datasource extension point

Select the *plugin.xml* and add the path to the class we created. Figure 8 shows the edited *plugin.xml*.

♣ TestDatasource × ☑ TestDatasource.java	
1 xml version="1.0" encoding="UTF-8"?	
2 eclipse version="3.4"?	
3 <plugin></plugin>	
4 <extension< th=""><th></th></extension<>	
<pre>5 point="de.thm.mdde.datasource"&gt;</pre>	
6 <client< th=""><th></th></client<>	
7 class="de.thm.mdde.datasource.TestDatasou	<pre>rce"&gt;</pre>
8	
9	
10	
11	

Figure 8: Plugin.xml

Finally, we have to deploy the new driver. Select the project in the package or file explorer and open the context menu with a right-click. In the context menu we choose export (9).

3 EvolveDB

陷 Project Explorer 🛛	💲 Plug-ins 👘 🖻 🕏 🍸 🕴 🗖	🖞 🗖 🚯 TestData
> 🗁 EvolveDemo		1 pack
<ul> <li>TestDatasource</li> <li>JRE Syste</li> <li>Plug-in [</li> </ul>	New Go Into	2
✓	Show In Show in Local Terminal	Alt+Shift+W > > >
<ul> <li>✓ Image: META-IN</li> <li>Image: META-IN</li></ul>	Copy Copy Qualified Name Paste Delete Build Path	Ctrl+C Ctrl+V Delete
	Refactor	Alt+Snift+1>
	Import	
	Export	
\$) 	Refresh Export Close Project Close Unrelated Projects	F5
0	Run As	>
<b>特</b>	Debug As Restore from Local History	>
	EvolveDB	>
	Team	>
	Compare With	>
	Plug-in Tools	>
	Configure	>
	SiLift	>
	Properties	Alt+Enter

Figure 9: Export a plugin

A dialog opens. In the dialog, we select **Plug-in Development**  $\Rightarrow$  **Deployable plug-ins** and fragements and click next (fig. 10).

3 EvolveDB

Export	×
Select Export the selected plug-ins and/or fragments in a form suitable for deploying in an Eclipse product.	Z
Select an export wizard:	
type filter text	
> 🗁 Install	^
<ul> <li>&gt; E Java</li> <li>&gt; Plug-in Development</li> <li>Peployable features</li> </ul>	
<ul> <li>Deployable plug-ins and fragments</li> <li>Eclipse product</li> <li>Target definition</li> </ul>	
> 🗁 Run/Debug	
> 🗁 Team	~
? < Back Next > Finish Ca	incel

Figure 10: Export Wizard: page 1

Next, select *install into host.Repository* and specify the path to the plug-in folder of your Eclipse installation (fig. 11). Click Finish and restart your Eclipse after successful installation. The new data source can now be used.

Export	—	×
Deployable plug-ins and fragments	-4	
Export the selected projects into a form suitable for deploying in an Eclipse produc	rt 🔪	
Available Plug-ins and Fragments:		
type filter text	Select Al	1
✓ IestDatasource (1.0.0.qualifier)	Deselect A	All
TestGenerator (1.0.0.qualifier)	Working Sc	-+
1 of 2 selected.		
l of 2 selected. Destination Options JAR Signing		
t of 2 selected. Destination Options JAR Signing O Directory:		
t of 2 selected. Destination Options JAR Signing Directory:	✓ Browse	
t of 2 selected. Destination Options JAR Signing Directory: Archive file:	<ul> <li>✓ Browse</li> </ul>	
t of 2 selected. Destination Options JAR Signing Directory: Archive file:	<ul> <li>✓ Browse</li> <li>✓ Browse</li> </ul>	
of 2 selected.  Destination Options JAR Signing  Directory:  Archive file:  Install into host. Repository:	<ul><li>✓ Browse</li><li>✓ Browse</li></ul>	
of 2 selected.      Destination Options JAR Signing      Directory:      Archive file:      Install into host. Repository:      Path_to_eclipse_folder\eclipse\plugins	<ul> <li>✓ Browse</li> <li>✓ Browse</li> <li>✓ Browse</li> <li>✓ Browse</li> </ul>	
of 2 selected.      Destination Options JAR Signing      Directory:      Archive file:      Install into host. Repository:      Path_to_eclipse_folder\eclipse\plugins	<ul> <li>Browse</li> <li>Browse</li> <li>Browse</li> </ul>	
t of 2 selected. Destination Options JAR Signing Directory: Archive file: Install into host. Repository: Path_to_eclipse_folder\eclipse\plugins	<ul> <li>Browse</li> <li>Browse</li> <li>Browse</li> </ul>	
of 2 selected.  Destination Options JAR Signing  Directory:  Archive file:  Install into host. Repository:  Path_to_eclipse_folder\eclipse\plugins	<ul> <li>✓ Browse</li> <li>✓ Browse</li> <li>✓ Browse</li> </ul>	

Figure 11: Export wizard: page 2

#### 3.2 Create a new migration script generator

The default installation EvolveDB includes a migration script generator for MySQL version 5.7 or higher. It is also possible to add custom generators via an extension point. To do this, create a plug-in via **File**  $\Rightarrow$  **New**  $\Rightarrow$  **Other**  $\Rightarrow$  **Plug-in Development**  $\Rightarrow$  **Plug-in Project** and open the MANIFEST.MF file. Switch to the tab *Dependencies* and add the dependencies marked in Figure 13.

CNew Plug-in Proje	ect				,
Plug-in Project Create a new plug-i	n project				-
Project name: Test	Datasource				
Use default location: C:\Users\	tion Torben\Documents\I	Promotion_Feature_	Test\workspace\Tes	Browse	i
Project Settings	oject				
Source folder: Output folder:	src bin				
Target Platform This plug-in is targe	ted to run with: nework: Equinox	~			
Working sets					1
Add project to v	working sets		~	New Select	
?	< Back	Next >	Finish	Cancel	

Figure 12: Eclipse: Create New Plug-in Project

equired Plug-ins	Jªz	Imported Packages
pecify the list of plug-ins required for the c lug-in.	operation of this	Specify packages on which this plug-in depends without explicitly identifying their originating plug-in.
de.thm.mdde.codegeneration (1.0.0)	Add	Add
org.eclipse.emf.ecore	Remove	Remove
- org.compse.core.runtime	Up	Properties
	Down	
	Properties	
	Total: 3	Total: 0

Figure 13: Plug-in Project Manifest.mf

Next, we have to create a class that implements the *ISQLGenerator* (fig. 14) interface. Figure 15 shows the class TestGenerator that implements the interface.

#### 3 EvolveDB

import org.eclipse.core.runtime.IProgressMonitor;

```
public interface ISQLGenerator {
   /**
    * Returns the display name of the extension.
    * @return
    */
   String getDisplayName();
   /**
    * Generate the migrations.
    * @param resEcoreFile --> The matching model.
    * @param project --> The currently selected project.
    * @param generator --> The generator chosen by the user.
    * @param monitor --> ProgressMonitor
    */
   void generate(Resource resEcoreFile, IProgressMonitor monitor);
    /**
    * Returns the content for the migration script.
    * @return
    */
   String getContent();
}
```

Figure 14: ISQLGenerator Interface



Figure 15: Class TestGenerator

Next, we have to add the new class as an extension for EvolveDB. Switch back to the *MANIFEST.MF* and open the Extension tab. Click on add and select the extension point *de.thm.mdde.extensionpoint.SQLGenerator*. (fig. 16)

New Extension	 C	) X
Extension Point Selection		₽
Extension Points Extension Wizards		
Extension Point filter: de.thm		
de.thm.mdde.extensionpoint.SQLGenerator		
Show only extension points from the required plug-ins Extension Point Description:		
(no description available)		
Available templates:		
? < Back Next > Finish	C	ancel

Figure 16: de.thm.mdde.extensionpoint.SQLGenerator extension point

II Extension Points		Extension Point Details
dit extension points defined by this plug-in in the followin	g section.	Set the properties of the selected extension point.
de.thm.mdde.extensionpoint.SQLGenerator	Add	ID: de.thm.mdde.extensionpoint.SQLGenerator
	Delete	Name: SQLGenerator
		Schema: schema/de.thm.mdde.extensionpoint.SQLGenerator.exsd Brows
		Show extension point description
		Open extension point schema
		₱ Find references

Figure 17: de.thm.mdde.extensionpoint.SQLGenerator extension point

Select the *plugin.xml* and add the path to the class we created. Figure 18 shows the edited *plugin.xml*.

#### 3 EvolveDB

♣ TestGenerator × D TestGenerator.java
1 xml version="1.0" encoding="UTF-8"?
<pre>2<?eclipse version="3.4"?></pre>
3 <plugin></plugin>
4 <extension< td=""></extension<>
<pre>5 point="de.thm.mdde.extensionpoint.SQLGenerator"&gt;</pre>
6 <client< td=""></client<>
7 class="de.thm.mdde.generator.TestGenerator">
8
9
10
11
12
Overview Dependencies Runtime Extensions Extension Points Build MANIFEST.MF plugin.xml build.properties

Figure 18: Plugin.xml

Finally, we have to deploy the new driver. Select the project in the package or file explorer and open the context menu with a right-click. In the context menu, we choose export (19).

3 EvolveDB

	- J	To	+Conor	1.0		
Ť	>		JRE Sy: Plug-ir		New Go Into	>
	>	e Co	src META-		Show In Show in Local Terminal	Alt+Shift+W > >
		<u>,</u>	build.p plugin		Copy Copy Qualified Name	Ctrl+C
				Ē	Paste	Ctrl+V
				×	Delete	Delete
					Build Path	>
					Refactor	Alt+Shift+T >
				è	Import	
				4	Export	
				8	Refresh Close Project Close Unrelated Projects	F5
				0	Run As	>
				*	Debug As	>
					Restore from Local History	
					EvolveDB	>
					Team	>
					Compare With	>
					Plug-in Tools	>
					Configure	>
					SiLift	>
					Properties	Alt+Enter

Figure 19: Export a plugin

A dialog opens. In the dialog, we select **Plug-in Development**  $\Rightarrow$  **Deployable plug-ins** and **fragements** and click next (fig. 20).

3 EvolveDB

Export	
Select Export the selected plug-ins and/or fragments in a form suitable for deploying in an Eclipse product.	Ż
Select an export wizard:	
type filter text	
> 😂 Install	^
> 😂 Java	
V 🗁 Plug-in Development	
Deployable features	
Deployable plug-ins and tragments     Eclipse product	
Target definition	
> > Run/Debug	
> 🗁 Team	~
? < Back Next > Finish	Cancel

Figure 20: Export Wizard: page 1

Next, select *install into host.Repository* and specify the path to the plug-in folder of your Eclipse installation (fig. 21). Click Finish and restart your Eclipse after successful installation. The new migration script generator can now be used.

- Export				×
Deployable plug-ins and fragments				-0-
Export the selected projects into a form suitable for deploying	in an Eclipse product			
Available Plug-ins and Fragments:				
type filter text			Select	All
TestDatasource (1.0.0.qualifier)			Deselec	t All
🗹 🚸 TestGenerator (1.0.0.qualifier)			Madina	C-4
			working	Set
1 of 2 selected.			]	
Destination Options JAR Signing				
O Directory:				
		~	Browse	2
Archive file:		~	Browse	2
Archive file:		~ ~	Browse	2
Archive file:  Install into host. Repository:		~	Browse	2
Archive file: Install into host. Repository: path_to_eclipse_installation\eclipse\plugins		× ×	Browse	2 2
Archive file: Install into host. Repository: path_to_eclipse_installation\eclipse\plugins		<ul> <li>✓</li> <li>✓</li> <li>✓</li> </ul>	Browse Browse Browse	2
Archive file: Install into host. Repository: path_to_eclipse_installation\eclipse\plugins		v 	Browse Browse	2 2

Figure 21: Export wizard: page 2

### 4 Metamodel

EvolveDB uses EMF for the database (MDDE model) and the migration metamodel. This section gives a short overview of both metamodels.

### 4.1 MDDE metamodel



Figure 22: MDDE metamodel

The abstract syntax of the MDDE metamodel (fig. 22) is similar to the relational model. When creating a new data source, the method *getRootObject()* from the *EDBDataSource* interface has to return the root object from an instance of this metamodel. Typically, the root *EObject* is an instance of class *Database\_Schema*.

#### 4.2 Migration metamodel



Figure 23: Migration metamodel

The symmetric difference model created by SiLift during the model matching phase is only used to represent the difference between the two model versions. The difference model is unsuitable for adding the additional information required for the migration. For this reason, we convert the *difference.symmetric* model into a migration model. Figure 23 shows the meta-model for the migration model. The migration model references the symmetric difference model and both mdde models.