# Scripting Enterprise Architect

A guided tour to Enterprise Architect's scripting capabilities

By Thomas Kilian

# Scripting Enterprise Architect

## A guided tour to Enterprise Architect's scripting capabilities

Thomas Kilian

This book is for sale at http://leanpub.com/ScriptingEA

This version was published on 2021-09-29

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

# Contents

# 1. Preface

Enterprise Architect[1] (EA) offers a wealth of API functions to support automated manipulation of UML models. For the beginner it is not easy to find a start. It is the intention of this book to help you getting a quick and smooth start

The contents of this book is the essence of a continuous work with EA since end 2003. Almost from the beginning on I have been playing with EA's automation. I have fallen into quite some pitfalls. With this book I hope to prevent newcomers from falling into the same pits again.

This book starts with a short introduction that takes you to reasons for scripting. The following section about EA's object model introduces the most important EA objects along with their attributes. The section about element creation and manipulation covers basic and a few advanced examples on how to populate a model with elements. The next section details the previously introduced objects. Attributes and operations are explained with references to GUI screenshots where applicable. Bits and pieces as an internal reference section details a couple of attributes. The final section shows screen-shots of many EA dialogues along with a back reference to where the single attributes are to be found in the API. So the circle is closed and you can see how GUI and API relate to each other.

I wish you will have fun when learning EA's API and as much success as I had so far. If you find any errors or you think that something needs to be explained differently just drop me a note[2].

[1] The EA version used to create this book was actually 9.3 (build 930). This book's version reflects changes from 10.0 (build 1006) and changes from 11.0 (build 1103) are currently being integrated. However, most of the references are also valid for earlier versions of EA.

[2] thomas.kilian@me.com

# 2. Copyright and Disclaimer

Also all of the information in this book has been tested by me in many circumstances I can not hold any liability for use of the here presented information[1]. However, I'd be glad to receive any kind of feedback to correct future updates of this book which you will receive for free in turn. Having said this, all information presented here is subject to change without notice.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Important note: Before going into production with your scripts follow these steps.

- Test with a dummy repository
- Test with a copy of the production repository
- Backup the production (initially and regularly later)
- Run and carefully observe on production
- If you are doing some conversion task also have a 'reverse script' that helps you undo certain steps.

[1] I really loathe writing such legal blurb since it should be obvious. By the way: German law applies! (Does that change anything?)

# 3. Introduction

Let me start with a

> Question: What's the best part of EA?
> Answer: You can script it!

Of course you find a lot of other pros[1], but scripting is definitely the most prominent advantage you will find in EA. Now you might ask the

> Question: Why should I use scripting?
> Answer: Because you often have repetitive work you want to automate.

That's a simple answer and probably too simplistic. The following list shows some major tasks you can tackle with scripting:

- **Regular repetitive task**: Repetitive would also mean you have to repeat a couple of step five to ten times. If that's it you would probably not start coding a script. But for tasks which you can foresee to use regularly and which involve say a minimum of 5 steps you must consider writing a script.
- **Importing Data**: If the build-in CSV import of EA is not sufficient you have the choice between manual import and a script. If it's only a couple of lines you should take the manual path. But if either the file is larger or you have to do the import on a regular basis you have no other choice than scripting.
- **Consistency checks**: Whenever you need to implement complex modeling rules it is necessary to cross check the model from time to time. A manual approach will certainly fail as the model complexity soon reaches the limits you can handle manually. So it is a good advice to script your own model checker.
- **Code generation**: EA has a build-in scripting for code generation. If you have only minor requirements for code generation you can live with that. But when it's going to be more complex this scripting language soon becomes too restrictive. Doing the same task with your preferred language will make things much easier.
- **Conversion of data**: Sometimes you need to change class names, add tagged values or move elements according to changed modeling rules. A manual conversion often is not feasible. That's why you will need scripting.
- **Fancy things**: Changing the behavior of the tool can support your modeling tasks a lot. Assume that you need to restrict the creation of elements depending on the model context. Or you need to set tagged values in several elements at once when you create, update or delete an element. In such cases you will find scripting a real benefit!

[1] I will not conceal various cons of the product. But with most of them you can live. Just take them as a challenge.

You see: there are quite a number of reasons why you should consider using scripting in EA. But that brings us to the next

> Question: Which language shall/can I use?
> Answer: The one you are familiar with!

Of course the real answer it not that simple, but quite. EA has a build-in scripting machine which lets you write `JScript` and `Visual Basic`. The nice thing here is that you can write scripts on-the-fly.

Before V10 debugging is only possible via `print` statements. And finding compilation errors can be a nightmare[2].

If you are not familiar with these languages and you prefer another one (e.g. `Python`, `Perl`, `Cxx` or whatever) you can do this as well. The only thing you need to find out is how to access the EA object where you can send API messages. You then can use the IDE of your choice to debug your application.

I can not go into details specific to any IDE or language[3]. For that reason I will not give language specific coding examples but abstract code sequences you need to adapt to the language of your choice. The advantage is that you need to think a lot more about what you are doing rather than taking ready-to-run samples. So in the end you learn more and faster!

## 3.1 Notation

As the different languages use different syntax I will use the most common dot-notation with C-like syntax elements with curly brackets. As a `Visual Basic` scripter you need to know that

```
for (<init>; <cond>; <post>) { <statements> }
```

translates to `VB`-like

```
<init>
while (<cond>) then
  <statements>
  <post>
wend
```

The dot-notation is similar for most other languages (`Perl` and `C++` use `->` instead of the dot).

```
Object.attribute
```

[2] I have not tested the new debugger but what I have seen from a Sparx demo looked promising.

[3] Actually I had chosen `Perl` as scripting language for myself. Together with the `Komodo` IDE debugging was real fun. Additionally I had build a framework to run `Perl` scripts as add-in from within EA. You can find more information here: http://community.sparxsystems.com/resources/scripts/extending-ea-perl

denotes the public `attribute` of `Object`.

```
Object.method(<parms>)
```

likewise represents the method call `method` with parameters `<parms>` where the latter is a comma separated list of object[4]/name-tuples.

More specifically I will use the following notation[5]:

Primitive = ‘`bool`’ | ‘`short`’ | ‘`long`’ | ‘`string`’;
CamelCase = “`type-writer string` starting with upper case and no spaces”;
EAobject = ‘`Ea`’ CamelCase; (* e.g. `EaElement` or `EaPackage` *)
Object = ‘`Object`’ | ‘`ActiveX`’ | ‘`DateTime`’ | EAobject;
TypeDefinition = Primitive | Object; (* declare attribute type/method result *)
VoidTypeDefinition = ‘`void`’ (* methods declared as `void` don’t return a value *)
| TypeDefinition;
Attribute = TypeDefinition ‘ ‘ CamelCase; (* e.g. `long ObjectID` *)
Method = VoidTypeDefinition ‘ ‘ CamelCase ( ParameterList );
ParameterList = [Attribute {‘,’ Attribute}]; (* optional comma-separated list *)

For better distinguishability all EA provided objects are prefixed with `Ea`. In your preferred language you will probably use these names without the prefix or in a dot-format (e.g. `EA.Element` instead of `EaElement`).

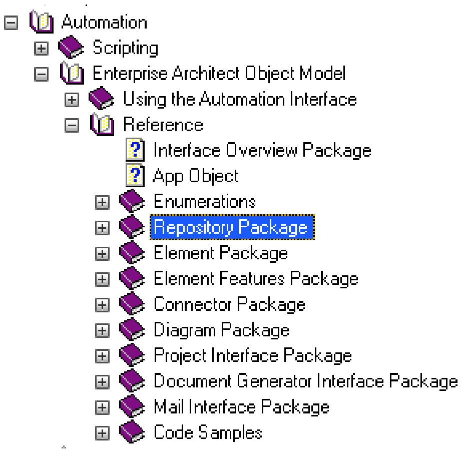
Further this document makes frequent use of hyper-links taking you to other locations in this document where the according term is defined in more detail.

[4] Actually in most cases a primitive like `short` or `long` is supplied.

[5] Basically I denote this using EBNF. If you don’t know what this is good for, just skip this information. You will be able to read this document without it.

# 4. EA’s Object Model

EA’s object model is quite complex. This book will not[1] cover all the gory details of EA’s API but it will just give you the point from where to start. You should consult the offline help inside EA via



- Automation
  - Scripting
  - Enterprise Architect Object Model
    - Using the Automation Interface
    - Reference
      - Interface Overview Package
      - App Object
      - Enumerations
      - Repository Package
      - Element Package
      - Element Features Package
      - Connector Package
      - Diagram Package
      - Project Interface Package
      - Document Generator Interface Package
      - Mail Interface Package
      - Code Samples

**Help/Help Contents/Contents/Automation/Enterprise Architect Object Model/Reference**

There is also an online help page with almost the same contents. But why use network capacity if you already have it on your machine? The structure is simple if you follow the Interface Overview directory or you skip the Enumerations directory. Inside the Repository directory you find the Repository class some way down. We will start with this one in the next section. The Package class is found in the same directory in the middle. Likewise you find the Element Class in the Element directory below the Repository directory. More about that later. All the class descriptions are in tabular format with the attributes listed first. Operations are found further down in a similar table. Most of this tabular documentation you can find in the Object Details chapter with additional comments. You will likely need to reference the EA help only for rarely used areas which are not detailed in this book.

This section will not go into details of the operations provided by the single classes unless needed. Instead we concentrate on the attributes to get a grip of EA’s repository structure.

In the following examples I simply assume you have the Example EA Model open. Probably you should have a copy of that as we might clobber it with our experiments.

But let’s start with the basics. The EA runtime object represents a single repository. Once you got hold of this you can create, read, update and delete everything inside the repository. Furthermore you can access a couple of complex operations (like exporting packages as XML) from the repository object.

Now, how can you access this magic object? If you are using the build-in scripting engine then EA offers the predefined object `Repository`. You can use that without any declaration in any

[1] Quite a number of secrets/hidden features are already explained in this book. As it is in the state of being written it is not complete and thus misses still some information. Over time this gap shall be closed so you can use this book not only for learning but also as a reference for the advanced use of EA.

context. That’s really simple and lets jump start you scripting immediately. Just skip the next chapter if that’s what you want to do.

## 4.1 Accessing EA from the Outside

So you need to develop your scripts outside EA. This means you need to get hold of the running EA application or you need to create one. Note that accessing a running EA instance will only get hold of the first one launched. I will not discuss the various possibilities including creating an instance here. You will likely be able to find that out later when it is getting important.

Here are some samples how to get the currently running EA instance from outside EA. The code snippets do not contain any validity checks. That means, when you run it without an active EA instance, it will loudly crash.

Eventually you need to link EA’s TLB to your IDE. This is to let it know about EA’s API classes, unless you deal with EA’s API objects as anonymous. The examples in this book do the latter as the declaration varies between the different languages. If you require object declaration it will generally be something like `EA.Repository` to declare the repository object or `EA.Package` for a package object and so on.

Since the EA help has improved in that respect you can also have a look at Sparx’ help page[2] (version 12).

### C++

```
        CLSID clsid;
        CLSIDFromProgID(L"EA.App", &clsid);
        IUnknown *pUnk = NULL;
        IDispatch *pDisp = NULL;
        HRESULT hr = GetActiveObject(clsid, NULL, (IUnknown**)&pUnk);
        if(SUCCEEDED(hr)) {
          hr = pUnk->QueryInterface(IID_IDispatch, (void **)&pDisp);
        }
```

These lines were borrowed from a Microsoft page. So I hope it will not be too misleading as I never have written a single line in `C++`.

[2] http://www.sparxsystems.com/enterprise_architect_user_guide/12/automation_and_scripting/setup.html

## Perl

```perl
        $App = Win32::OLE->GetActiveObject ("EA.App");
        $Repository = $App->Repository;
```

## Python

```python
         eaApp = win32com.client.Dispatch("EA.App")
         eaRep = eaApp.Repository
```

## Visual Basic

```vb
        Dim Repository As EA.repository
        Set EAapp = GetObject(, "EA.App")
        Set Repository = EAapp.repository
```

> I’m no VB expert, but in a differnt VB flavor you just use `Dim Repository` in the first line above.

## Java

Support for Java has recently[3] been added. You need to first copy two files from the Java API subdirectory of your EA program directory:

- `SSJavaCOM.dll` -> `%SystemRoot%\system32` directory.
- `eaapi.jar` -> a location in the Java `CLASSPATH` or where the Java class loader can find it at run time.

Now you should be able to access an EA repository.

```java
    public void OpenRepository() {
      org.sparx.Repository repository = new org.sparx.Repository();
      repository.OpenFile("<path to repository>.eap");
    }
```

[3] For me using EA for 20 years that means in the last couple of years - somewhere around V9.3 or so. I don’t remember exactly.

## 4.2 Repository

Now as you got hold of the repository object you can start exploring it a bit.

Let's print the name of the repository.

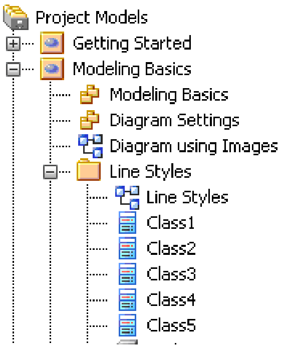
```
print Repository.ConnectionString;
```

This will output either the file path and name of the EAP

e.g. `C:\Documents and Settings\<user>\Desktop\EAExample.eap`

or the ODBC connection string depending on the type of repository you are actually using. Now let's take a first look into the repository.

On top level of a repository there is at least a single root node. It is possible to have multiple roots so this is a `Collection`. The visible part of the repository we are going to explore first looks like:



```
Project Models
  Getting Started
  Modeling Basics
    Modeling Basics
    Diagram Settings
    Diagram using Images
    Line Styles
      Line Styles
      Class1
      Class2
      Class3
      Class4
      Class5
```

**EAExample.EAP repository structure**

Lets simply retrieve and print the name of the (first and currently only) root package:

```
        root = Repository.Models.GetAt (0);
        print root.Name;
```

This will print the name of the root package `Project Models` which was created at first[4] in the repository. You can iterate through all root nodes and print their names. For the example model there's only a single root node. So create a second root on your own and try the iteration.

That's all you need to know for the beginning. From here the fun begins!

## 4.3 Package – and View – and Model

Okay, that sounds confusing. But it's only history. `Packages`, `views` and `models` are almost the same. They are all `packages`. Almost. And to make it a bit more confusing: `packages` are a tuple of package and element.

... omitted ...

[4] You should refer to the `Ordering` section once you are familiar with the details.

# 5. Creating New Things

Manipulating the repository via automation is the real reason for using the API. A main application is the import of mass data. But again, lets start from the top.

Everything that is added to the repository is created via

```
newThing = EaCollection.AddNew ("name", "type");
newThing.Update();
```

where the `name` is always the name of the new element and `type` has different meaning depending on the thing we want to add. Although some `AddNew` operations do not use the `type` parameter you must supply it (with an empty string). Common to all `AddNew` is that they return either the successfully created new element or a null pointer.

You already know about different `EaCollection` attributes (e.g. `Models`, `Packages`, `Elements`, etc.). These determine what is added and what the `type` parameter means. Further these objects contain a unique number `ObjectType` (according to above examples 6 for `Models`, 5 for `Packages`, 4 for `Elements`, etc.) so you can find out from what class they were created.

For the following code samples I assume that variables contain the value they were once initialized with. Unlike in the previous section I can not refer to any GUID as they will be arbitrary for the model you are using.

## 5.1 Adding a Root

Assuming you have the EAExample.eap as a copy we can simply go on adding a new root here. You are also free to create an empty `.EAP`.

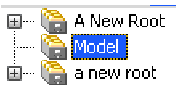
Just run the following code on your model:

```
        root2 = Repository.Models.AddNew ("A New Root", "");
        root2.Update ();
```

You will find a new root with the name `A New Root` in your repository. The second parameter for the `AddNew` is not used, so an empty string is passed.

Now run the script the with the name `a new root` (the first lower case character is important). As you will notice the first root is added at the very top while the second is below the previously existing root[1]. It should look like the following picture.

[1] Actually I haven't tested this with a case insensitive repository. It might turn out that those behave different.

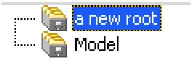


**Two New Roots**

The reason is simply that EA sorts these elements alphabetically unless a `TreePos` attribute is supplied. If you need to order the new root elements differently you have to supply the right `TreePos` attributes. Delete the two newly created roots and run the following code snippet:

```
        root1 = Repository.Models.GetAt (0);
        root1.TreePos = 1;
        root1.Update();
        root2 = Repository.Models.AddNew ("a new root", "");
        root2.TreePos = 0;
        root2.Update ();
        Repository.RefreshModelView (0);
```

The final `RefreshModelView` will update the browser view so that you should see the picture below. This is necessary as the sort order is changed which is not directly reflected by EA. If you don't invoke that you will not see the correct ordering of the root packages. Unless you care for the correct ordering to take immediate effect you usually can simply omit this call.



**Ordered New Roots**

A funny thing is when you try to add a root package with an empty `name` string. This is passed to the database where you get a message from the underlying database. Well…

… omitted …

# 6. Advanced Creations

If you have worked up to this point you are probably already an advanced EA scripter. Most likely you will be able to make your way yourself. But in the end it’s easier with a couple of recipes. I will add them in this chapter time by time.

… omitted …

# 7. Object Details

This section is referenced from above sections. You can use it as a quick reference but it contains also a number of information which is not detailed in EA's help.

Unless otherwise noted all properties are r/w. Collections are r/o in the way that they can not be set to anything different. However, each collection has operations to alter their contents.

There are two general operations `bool Update()` and `string GetLastError()` which are not listed below as most objects respond to them and their use is quite obvious. If `Update` returns false you should consider the return value of `GetLastError`. Note that it is not cleared by all operations but may reside from an arbitrary previous error. You should only check it after an `Update` or as noted for different operations.

Since a couple of objects do not have specific operations their appropriate section just lists the attributes. Else attributes and operations appear under separate sub-sections.

I have tested most, but not all of the API listed below. Where appropriate I have added some observation in the API use. Most of the API stems from pre 9.3 EA version which is where this book has been written. Every now and then Sparx adds new functionality and so I added a marker with the EA version in brackets to indicate attributes and operations that have been introduced ever since. Note that a lot of the functions introduced with 12.1 did not work they were described in EA's help[1] when the V13 beta was out for testing already.

## 7.1 EaRepository

### Attributes

**`EaCollection Authors`** —

An `EaCollection` of `EaAuthor` objects.

These are defined with `Settings/Project Types/People/Authors`.

**`bool BatchAppend`** —

Set this property to true when your automation client has to rapidly insert many elements, operations, attributes and/or operation parameters.

Set to false when work is complete.

This can result in 10- to 20-fold improvement in adding new elements in bulk.

**`EaCollection Clients`** —

An `EaCollection` of `EaClient` objects.

These are defined with `Settings/Project Types/People/Clients`.

[1] Consider reporting any of them as bug as I did not send a single one.

**`string ConnectionString`** —

The filename/connection string of the current Repository.

This is either a filename (`<drive>:\path\to\file.eap`) or an ODBC connection string.

If you find a filename you need to do some additional checks. If the size of the file is small (less than 512k Byte) this is a shortcut file, else it is a `.EAP`. If it is a shortcut file then you have to read its contents. The first line starts with `EAConnectString:` followed by the real connection string (this is only one recursion as it seems). You can forget about the optional following lines[2] which are commands to EA to do certain actions upon opening EA.

Detailed information about the construction of connection strings will be found via Google[3]. However, the connection string will contain `DBType=<n>;` where `<n>` is one of

| | |
|---|---|
| 0 | MYSQL |
| 1 | SQLSVR |
| 2 | ADOJET |
| 3 | ORACLE |
| 4 | POSTGRES |
| 5 | ASA |
| 7 | OPENEDGE |
| 8 | ACCESS2007 |

**`EaCollection Datatypes`** —

An `EaCollection` of `EaDatatype` objects.

These are defined with `Settings/Code Datatypes...`.

... omitted ...

[2] I might document these in a future version of this book.

[3] You may consult http://www.connectionstrings.com/ to see how such a string is constructed.

# 8. Bits and Pieces

## 8.1 Ordering of Collections

You will find that the ordering of collections does not match that you find in EA's project browser. Most objects have a `Pos` or a `TPos` property. Only if this is non-zero it is valid and supersedes the occurrence in the collection. Also you will find that the browser has some rules to internally group packages, diagrams and elements. Further there is a `Sort Features Alphabetically` option under `Tools/Options/General` which interferes. So if you need the same ordering as in the browser you have to code a bit.

## 8.2 GUID

A GUID (Global Unique ID) has the general format

```
{XXXXXXXX-XXXX-xxxx-XXXX-XXXXXXXXXXXX}
```

where XX is a hex code with upper case chars and xx one with lower case. A GUID in contrast to other object IDs in EA is (as the name suggests) unique[1] all over the world. There are a couple of Get<what>ByGUID operations available for the `Repository` class. The `<what>` can be `Package`, `Element`, `Attribute`, `Method` and `Connector`.

There's a bit of magic in some GUID especially with that of tagged values[2].

... omitted ...

[1] Theoretically this varies over $2^{16*8} = 2^{128} \approx 3 * 10^{38}$ which means that you can number amounts of $3 * 10^{-14}$ grams (much less than a dust grain) uniquely for the whole earth.

[2] I have detailed that in my book Inside EA.

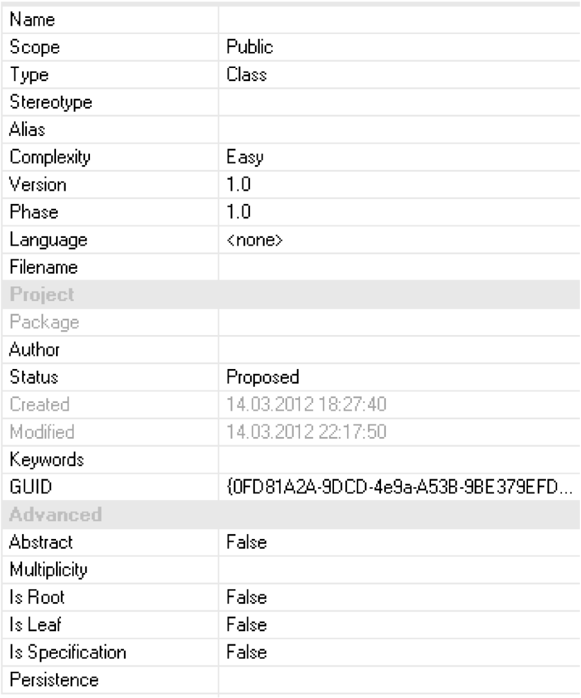
# 9. GUI Reference

In this chapter you will find most of the property dialogs. Where possible a green frame indicates the access via the API. In case of a yellow frame the according field is not addressable with the API directly and a bypass through EA's tables might be needed. In such cases you might need to have a look into my Inside book.

## 9.1 Element

Here you find property dialogs that deal with elements. Note that packages also have an element part and share information in the same dialog as for pure elements.

### Dockable Properties Window



| | |
|---|---|
| Name | |
| Scope | Public |
| Type | Class |
| Stereotype | |
| Alias | |
| Complexity | Easy |
| Version | 1.0 |
| Phase | 1.0 |
| Language | <none> |
| Filename | |
| Project | |
| Package | |
| Author | |
| Status | Proposed |
| Created | 14.03.2012 18:27:40 |
| Modified | 14.03.2012 22:17:50 |
| Keywords | |
| GUID | {0FD81A2A-9DCD-4e9a-A53B-9BE379EFD... |
| Advanced | |
| Abstract | False |
| Multiplicity | |
| Is Root | False |
| Is Leaf | False |
| Is Specification | False |
| Persistence | |

**Referenced by `EaElement` attributes**

`Name`, `Visibility` (Scope), `Type`, `Stereotype`, `Alias`, `Complexity`, `Version`, `Phase`, `GenType` (Language), `GenFile` (Filename), `PackageGUID` (GUID; shows `Name` of package in dialog) `Author`, `Status`, `Created`, `Modified`, `Keywords`, `ElementGUID` (GUID), `Abstract`, `Multiplicity`, `IsLeaf`, `IsSpec` (Is Specification), `Persistence`

For packages the `EaPackage` attributes for `Name`, `Version`, `Created` and `Modified` are the same as the shadow attributes in the according element.

For packages the `ElementGUID` of the related element is the same as the `PackageGUID`.

The `Is Root` flag is only available via `t_object.IsRoot`.

Though the help tells that `Created` and `Modified` are r/w they actually are not. The first will not be changed at all on an `Update()` and the latter will always be the current time stamp. You need to bypass the API in order to modify those values.

… omitted …

# 10. Further Reading

Finally I would like to add a few links where you will get further help.

## 10.1 Feedback

Any questions you have are important. So you should ask them. Feedback is important for you, me and of course all the other readers. So you are encouraged to send these to one of the below links:

**http://leanpub.com/ScriptingEA**
: The page where you bought the book has a small discussion forum enabled. Here you can also post.

**thomas.kilian@me.com**
: You can mail me directly if you have specific questions.

## 10.2 Inside Enterprise Architect

If you got to some in-depth knowledge of EA's automation you will likely be interested in the details about EA's database. My book **Inside Enterprise Architect** which is available at http://leanpub.com/InsideEA will give you these details. You'll be astonished how fast the API reacts when you write a single SQL statement instead of a clumsy API loop.

## 10.3 Sparx Forum

Probably not necessary to name this source, but anyway:

> https://www.sparxsystems.com/forums/smf/index.php

When you post your questions here you should select the right forum and only post once. Getting help here is most likely the fastest lane you can find. I also post regularly.

## 10.4 Sparx Community

A not so well known source as Sparx itself does not link this on its forum pages:

> http://community.sparxsystems.com

Here you will find a variety of articles and resources around EA.

## 10.5 Geert Bellekens

Geert has become a kind of institution at Sparx' forum. He is a regular poster. But beyond that he also has an excellent blog dealing with ULM in general and Enterprise Architect in particular. Check it out:

> https://bellekens.com

And last but really not the least I like to link Geerts EA navigator source:

> https://bellekens.com/ea-navigator/

This will take you into the wonderful world of EA Add-Ins. Have fun!