ARLANET DIGITAL ENGINEERS

Oude Blaauwweg 2
1521 RN Wormerveer
Tel +31(0)75 - 647 60 90

BTW nr. NL8103.58.347. B01
KvK 34 16 59 87
Rabobank 14.52.47.821

www.arlanet.nl
info@arlanet.nl

Copernica .NET Library

Written by: Y.E. Nunnink, Arlanet BV

Date: 14 January 2014

Version 1.00

.

Orderer	:	Arlanet BV
Contact	:	Paul de Metter
Projectnr.	:	2014/ARL/05
Date	:	15 January 2014
Description	:	Copernica .NET Library
Revisions	:	v1.00 17-12-2014 Y.E. Nunnink

Inhoudsopgave

1. General

The purpose of the .NET library is to simplify the communication with Copernica. The library can be used to synchronize data with Copernica.

With the library you can map objects by using attributes/base classes and synchronize to Copernica. So you don't have to worry about mapping the data.

The library contains the following main functionality:

- Data validation
- Copernica REST API handling
- Object validation
- JSON conversion

Most of these functionalities will take place in background processes. Although it is important for developing the connection with Copernica.

To make use of the libraries functionality a Copernica Handler needs to be implemented. An instance of this handler will give access to the methods. This is important for validating the fields between your objects and Copernica.

2. Authentication Token

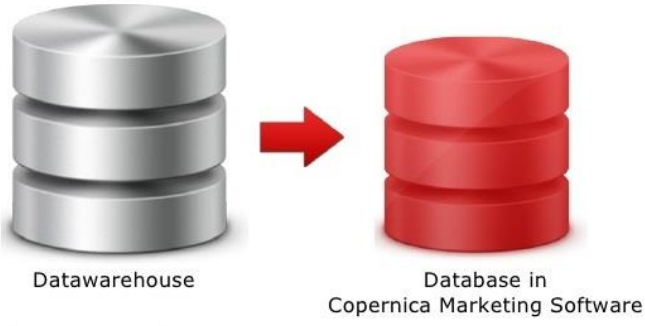
To get access to the database of your account you'll need an authentication token. To acquire this token you'll need to register your application on your account at Dashboard->Applications.

The authentication token needs to be initialized in the CopernicaHandler in order to have access to the databases. See 4.2 for more information.

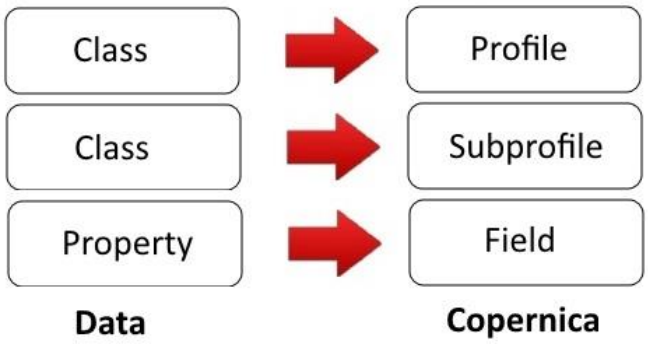
3. Data Mapping

3.1 General

Data mapping is a big component in the library. It is valuable because the data is often stored in two places.



A mapping needs to be made to make sure the data is transferred correctly. This mapping specifies the relationship between the objects/classes and the Copernica Database. Classes can be specified as a Profile or Subprofile in the database. Also properties get linked to what field they belong.



This mapping enables to syncrhonize data with Copernica.

3.2 Attributen en Klassen

As explained in the previous page, objects need to mapped in able to synchronize your data to Copernica. This can be done using attributes and base classes. Adding these to your classes will enable the library to validate and convert your data, which is needed in order to make sure the data gets inserted successfully.

The following attributes and base classes are available for specifying the mapping.

Attributen

- CopernicaField [Name, Type, Length]
  - Indicates which field the property belongs to in Copernica.
- CopernicaKeyField [Name, Type, Length]
  - Indicates which property should be used as identifier.
- CopernicaDatabase [ID]
  - Indicates which database a class belongs to.
- CopernicaCollection [ID]
  - Indicates which collection a class belongs to.

Base Classes

- CopernicaProfile
  - Indicates a class is a profile in Copernica.
- CopernicaSubProfile
  - Indicates a class is a subprofile in Copernica.

3.3 Implementation

In the image below you can see an object mapping implementation example.

```
[CopernicaDatabase(104)]
[CopernicaCollection(729)]
2 references
public class Product : CopernicaProfile
{
    [CopernicaKeyField("DatabaseId", Type = CopernicaFieldTypes.IntField)]
    0 references
    public string ID { get; set; }

    [CopernicaField("Naam", Type = CopernicaFieldTypes.TextField)]
    0 references
    public string Name { get; set; }

    [CopernicaField("Prijs", Type = CopernicaFieldTypes.TextField)]
    0 references
    public float Price { get; set; }
}
```

The ID is the identifier from the database that is being used by the application. ID is being specified as a KeyField in Copernica and is called DatabaseId. This simplifies the synchronization with Copernica as explained in the next chapter

3.4 Identifiers

The specified KeyField is being used as an identifier by the library to update data. This allows you the update data with your own identifier.

For example a customer signed up for a newsletter. You retrieve the customer from your database, edit the data, and synchronize it with your database and Copernica. Normally you would have to store the ID from the Copernica Database in your own database to be able to update the profile. Now this is not necessary because you specified your own identifier that is used by the library to update the profile.

4. Validation

4.1 General

A Copernica Handler implementation is needed in order to enable the library to validate your objects. In this implementation all the objects need to be registered. When registering an object the library will compare the fields with the fields in Copernica and validate the attributes.

Note! The core functionality of the library cannot be used without the Copernica Handler implementation.

4.2 Implementation

In the image below you can see an example of a Copernica Handler implementation. The class “Product” is beind registered as a data item. This validates the fields when CopernicaHandler.Instance is called for the first time.

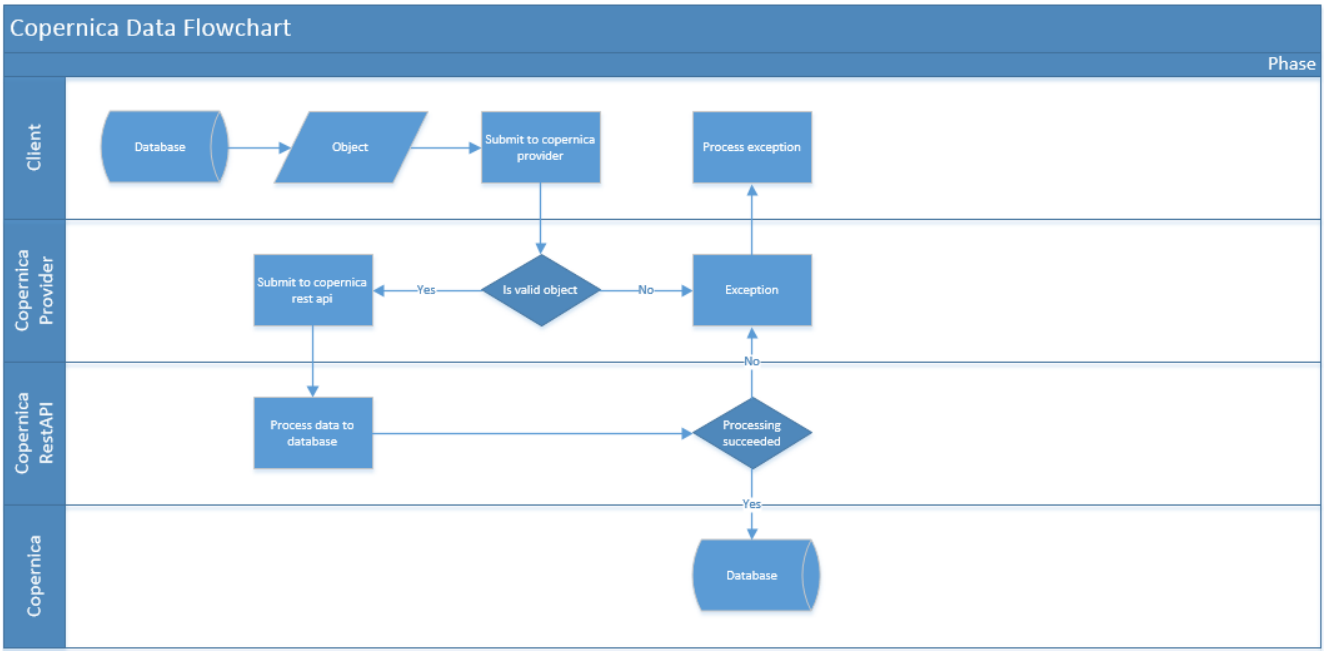
```
3 references
public class CopernicaHandler : Copernica<CopernicaHandler>, ICopernicaHandlerBase
{
    0 references
    public CopernicaHandler() : base("accesstoken here!")
    {

    }

    3 references
    public void RegisterDataItems()
    {
        RegisterDataItem(new Product());
    }
}
```

After the implementation you can use the core functionality in the following way.

```
var klant = new Client {ID = 16, Email = "support@arlanet.com", Name = "Arlanet"};
CopernicaHandler.Instance.Add(klant);
```

5. Data Flow

In the image below you can see the data flow between an application(client) and Copernica. Copernica provider is the library.



6. Questions

Do you have questions about the library or the implementation? We will be glad to hear them. Contacts us at support@arlanet.com or call us at 075-6476090. You can ask for Paul de Metter.