Enigma - Parans

Programmers Manual version 1.0

Information for programmers about the program Enigma Parans

Introduction

Enigma Parans (from now on *Parans* for short) supports the calculation of paranatellonta (parans). For a description of the functionality, please check the User Manual.
This document is for developers who want to use the code, create an own version or are just curious.
Parans is free software and open source. You are allowed to use it provided your own software is also free and open source. The same goes for the use of the Swiss Ephemeris, a library for astronomical calculations that is used by Parans. Check the file *copyright.txt* in the root of the source for a formal description about restrictions and possibilities regarding copyright.

Please note that my experience as a programmer is mainly with Java. Parans is written in Free Pascal. I have used several Pascal versions before but that was mostly a long time ago. So I am actually learning on the job. This will not result in the best code possible but I will try to improve it consistently.

Setting up your development environment

The source of Parans is written in *Free Pascal*, an open source implementation of Object Pascal, and to a high degree compatible with Delphi. However, I do use some constructions that are not supported by Delphi. If you want to use Delphi, you will need to make some changes in the code.
If you want to check the code and compile the program yourself, you should use Free Pascal and the standard IDE called *Lazarus*.

Free Pascal and Lazarus

You can install Lazarus which includes a current version of FreePascal. Download Lazarus from https://www.lazarus-ide.org/ and navigate to Downloads. Make sure you select Windows (32 and 64 Bits).
Lazarus also works on Linux but I did not test that.
The installation itself is straightforward. One point to keep in mind: do not install in the standard Program files folder. This will cause trouble if you want to update and updates happen frequently. The standard folder c:\lazarus is the best way to go.

External software and data

Enigma uses software by the Swiss Ephemeris (SE) to perform most astronomical calculations. You need to install the following:

- The file swedll32.dll
- The datafiles from the SE.

You can use the dll and datafiles that are part of the installation package of Enigma-Parans. The dll should be in the root of your Lazarus project. The datafiles should be in a folder *se* that is one level up, so at the same level as the folder for the project itself.

Source and files

The source is available at https://gitlab.com/jkampherbeek/enigma-parans
The application uses three projects that are combined within a project-group:

- **PgParans.lpg**: projectgroup
- **Parans**: project for application
- **UtParans**: project with unit tests
- **ItParans**: project with integration tests.

The code is in the folders:

- **parans**: general code, projects, projectgroup.
- **be**: code for the backend.
- **fe**: code for the frontend.
- **test**: unit-tests and integration-tests.

The project contains the following additional files and folders:

- **docu**: user manuals (English and Dutch) and developers manual (only English).

- **help**: html-files for help, including folder **css** with stylesheets.
- **language**: ini-files with translated texts for i18N.
- **log**: logfiles.
- copyright-files: **copyright.txt** and **gpl-3.0.txt**
- **readme.md**
- **swedll32.dll**: dll for the Swiss Ephemeris.
- **parans.ico**: icon
- **img**: image file.
- **lang.sel.properties**: properties-file for the selected language.

The folder data is not part of the distribution: you need to create it yourself. The file swedll32.dll must be downloaded from: https://www.astro.com/ftp/swisseph/
It is in the zip file *sweph.zip*.

Architecture

Separation of concerns

There is a separation between backend and frontend (user interface). The backend is accessible via an API. This is not enforced but should be done by convention. As this is a small application more effort in this respect is required when the application grows.

Naming and documentation

An attempt is made to use meaningful names for variables, methods and classes. Documentation in the source will be minimal.

i18N (Internationalization)

Free Pascal has a standard solution for i18N. There is however a drawback: texts from forms are automatically added to the language files, but not in all cases. And the language-files are verbose. No problem for a small application but it will become a problem if the application grows significantly. Therefore I developed a simple alternative for i18N. The logic is in the class *Rosetta* in the unit *UnitTranslation*. The translated texts are saved in ini files in the folder language. Only English and Dutch are supported.

Logging

Existing log-systems for Free Pascal are mostly aimed at debugging. Most importantly: they do not support rotating logfiles. To solve this I wrote a simple logging system that keeps only the logfiles for the last three sessions. In case of an error, these files can come in handy. The logging is implemented in the class *TLogger* in the unit *UnitLog*.

Singletons

Several classes are singletons: classes for logging, translation and the central controller. At https://wiki.freepascal.org/Singleton_Pattern you will find an excellent description of various implementations of this design pattern. I used the last example, Singleton3, as an approach for all Singletons.
In this approach, the Constructor is private, which leads to a warning Constructor should be public. To make it unnecessary to change the project options to disable this message, I added the following directive to the sources that contain singletons.

```
{$WARN 3018 off : Constructor should be public}
```

Testing

Unit-tests are implemented and also integration tests. Unfortunately, Free Pascal/Lazarus offers no possibility to calculate the coverage.
The focus of the unit tests is on parts of the backend. The integration tests mainly test the correctness of the calculations.

I do not intend to add tests for the UI. These types of tests are difficult to maintain and add little value, especially as most of the logic is within the backend. The UI is focussed only on presentation.