

# Improvements in the Phalanger project

---

**Members:** Adam Abonyi, Daniel Balaš, Miloslav Beňo, Jakub Míšek

**Supervisor:** Filip Zavoral

**External Advisor:** Jan Šeda, Tomáš Petříček

## Background

Phalanger is an open-source compiler of the PHP language for .NET and Mono platforms, developed as a software project at MFF UK a few years ago. The goal of this software project is to implement several extensions to this project to make it more useable in a real world and re-implement part of the project core as described below.

Phalanger is currently implemented using .NET 2.0 with a massive use of runtime code generation. It doesn't use any compiler framework and generates .NET code using .NET Reflection Emit (managed classes for generating .NET bytecode).

The language is currently mostly compatible with the official PHP distribution however there are several incompatibilities that we'd like to address in this project (for example PHP language now includes its own support for namespaces). Phalanger also implements several additional language features to allow interoperability with .NET code (which enables calling .NET objects directly from PHP code running using Phalanger).

Phalanger also currently includes (managed) re-implementation of the core PHP libraries and is capable of loading native PHP4 libraries. Thanks to this it is able to run several open-source PHP applications (e.g. PhpBB forums), but there are still many missing features, which prevents it from running larger number of popular applications.

## Related Technologies

In this paragraph we shortly introduce several related technologies that are already being used in Phalanger and also technologies that we'd like to use during this project to improve Phalanger.

- "DLR" is a set of .NET libraries recently released by Microsoft, which makes it easier to implement dynamic languages for .NET. It makes it easier to implement various common problems with dynamic languages including compilation of a dynamic language or simplifies implementation of dynamic operation binding (e.g. whether "+" operation represents integer or float addition).
- Silverlight is a web browser plugin which contains simplified version of .NET runtime and can be used for developing rich client-side applications. Phalanger currently contains a "proof of concept" implementation of Silverlight support (using outdated alpha version).
- Visual Studio 2008 IDE is widely used development environments for developing .NET projects. Phalanger currently includes bindings for Visual Studio which makes it possible to create basic Phalanger project in Visual Studio with support of syntax-highlighting however no advanced feature (like bracket matching or IntelliSense is supported)

## Goals of the Project

### Improved IDE Support

Current visual studio extension supports the following features:

- Syntax highlighting of PHP keywords and comments, the rest of code (including HTML code in the PHP source code) isn't highlighted.
- Currently, there are several project templates for web (Standard PHP Website, PHP Silverlight Web Application), and several templates for creating standard .NET applications (PHP Console Application, PHP WinForms application). There is probably no need to include larger number of templates, but we may add some.
- The IDE editor currently supports debugging (which is a feature of the compiler, which emits debugging symbols and Visual Studio)
- Currently, IDE includes partial support for WinForms designer, which is a part of a related project based on Phalanger.

The goal of this project is to improve PHP bindings for Visual Studio using Phalanger by using its core for implementing advanced semantic analysis of the PHP source code. The editor for the PHP source code in Visual Studio should have roughly the following new features:

- Extended syntax highlighting including support for highlighting matching braces and coloring (currently unsupported) HTML code embedded in the PHP page.
- Intellisense (auto completion, word completion) – While writing the PHP code, best possible words will be offered (known PHP functions, user-defined PHP functions and variables). It will be used for completing not completed terms, auto completion of pair of braces and auto completion of quotes.
- Function arguments – While writing function call, whole function header with comments above the declaration, will be shown under the cursor.
- Intellisense will be predicting types of objects and accordingly, it will offer dropdown list with members of this object. In most of the cases isn't possible to predict a type of object, because PHP is a dynamic language and user can create types and assign objects dynamically at runtime, however several mechanism can be successfully used to “infer” the right type.
- New IDE support will underline warnings and errors (undeclared functions, syntax errors, etc.)

We will use Phalanger core to analyze the PHP source code, and the mechanism for “inferring” the types will analyze the Phalanger AST. Phalanger runtime will be also partially used to get the list of available PHP functions when displaying the word completion.

## Implement Phalanger core using “DLR”

Phalanger currently doesn't use DLR or any other compiler framework, it generates code by .net reflection emit. It's much more convenient to use DLR than using own code generation, so the code would be more readable for people, it'll be more optimized and any improvements of DLR will improve also Phalanger.

The primary outputs of this project should be Phalanger implementation using some parts of the “Dynamic Language Runtime”. The two components that will be the Phalanger core are:

- “DLR trees” (an abstract syntax tree implementation in DLR, which is used for compiling a code in dynamic languages)
- “DLR dynamic sites” (a DLR feature for implementing common dynamic language operators, like mathematical operators, dynamic type conversion, etc.)

Very likely, we'd also like to implement the following:

- “IDynamicObject” – an interface shared across multiple dynamic languages implemented using DLR (e.g. IronPython, IronRuby), which makes it possible to use objects created in one language from another and use dynamic invoke independently across multiple languages.

Reimplementing Phalanger core using DLR is an important goal, because DLR is becoming primary compiler framework for the .NET, which is especially designed for dynamic languages. This will largely simplify some of the complex parts of the Phalanger runtime as well as it could lead to more efficient execution of the dynamic aspects of PHP language.

It is still not clear, whether everything can be solely implemented using DLR, so we will work with the developers of DLR to make sure that it is powerful enough for projects of a similar complexity as Phalanger. Thus, a part of the output of this project may also be a set of suggestions to the DLR team describing what features that are important for Phalanger are missing in current implementation.

### **Mono & Moonlight support**

Part of the goal to re-implement Phalanger core on DLR is also to ensure that everything works smoothly on Mono. This is because Phalanger participates in Google summer of code as a project mentored by Mono and so Mono support is an important goal.

To ensure that Phalanger runs well on mono, we will use standard PHP language and library test adapted from PHP distribution. Other features like .NET interoperability and Silverlight/Moonlight support must be also tested, so new tests have to be created. Possibly, we may also develop ways for testing graphical features of Silverlight, however this isn't our primary goal.

## Other (possible) features

### **Stability and Extensions**

There are some minor stability issues with Phalanger now. It's hard to locate the problems, because it behaves normally in most of the cases. To improve stability and solve these problems there will be cooperation with commercial companies that use Phalanger. As a part of this cooperation it will be

also implementing missing extensions that these companies needs for using Phalanger in real world commercial environment.

The list of extensions that will be implemented as part of the project will change depending on the needs of our commercial partners. Phalanger currently includes support for functions from standard PHP library and implements MSSQL, MySQL and SimpleXML extensions. These extensions are available only on .NET implementation (which is reasonable for database functionality), though implementing SimpleXML support for Silverlight/Moonlight is important goal of our project.

### **Documentation, tutorials, etc.**

There is already a good documentation generated from Phalanger source code. But it's not sufficient for other purposes, so as a part of the project will be writing articles and tutorials, allowing people to learn how to use Phalanger and PHP community could finally accept it as alternative solution for running PHP applications, especially for Silverlight. Part of this goal is to describe Phalanger internals, which should enable the community to contribute to the Phalanger source code.

### **LINQ support**

LINQ (language integrated query) is a relatively new technology , which enables programmers to write SQL-like queries inside the program code. The advantage of this approach, as opposed to commonly used separation of code and data queries, is that the query is analyzed during the compilation thus producing a valid (in sense of syntax and types) query. Moreover with the support of expression trees (object representation of language expression stored as data), LINQ queries are then shared by several data sources (in-memory data, SQL server etc.) and even by user code. In order to support LINQ a language has to support several functional language features, which would make the LINQ queries comprehensible. For example in C# 3.0 implementation, lambda-expressions and tuples are commonly used.

Phalanger currently supports LINQ queries only for in-memory data. Producing an expression tree in dynamic language such as PHP is problematic, because when the expression is compiled, the types are unknown and thus additional constraints have to be applied on the code. This is an interesting problem, which we are looking forward to solve in our project. However we are not sure if this problem is solvable, so we consider this only as an optional goal.

## **References**

[1] Martin Maly: <http://blogs.msdn.com/mmaly>