Babylscript: Multilingual JavaScript

Ming-Yee Iu

babylscript@babylscript.com

Abstract

Babylscript is a multilingual version of JavaScript. It has different language modes in which keywords, objects, and functions are translated to non-English languages. Babylscript uses separate tokenizers for each language and extends JavaScript's object model by allowing properties to have multiple names, so objects expose different interfaces for different languages.

Categories and Subject Descriptors D.3.3 [*Programming Languages*]: Language Constructs and Features–Classes and objects

General Terms Design, Human Factors, Languages

Keywords JavaScript, internationalization

1. Introduction

Most programming languages are biased towards a certain vocabulary, usually American English. Keywords are in English, objects are given English names, and methods are given English descriptions. The dominance of English in programming languages provides many benefits to the programmer community. With a common language, professional programmers form a single community that can share code and ideas. This dominance does lead to many peculiar effects though. In non-English countries, students cannot learn about programming until they learn English first. Office workers cannot write macros or scripts to automate their daily work without developing a proficiency in English first. Non-English programmers are often forced to write their code in English if they want their code to be used by others.

Although a programming language may have different language versions, such as French, Spanish, and English versions, these different versions are often mutually incompatible. Code that uses French keywords and functions cannot be used with an English compiler, for example. As such, these different language versions have small user communities, resulting in less documentation, fewer libraries, and slower evolution.

Babylscript is a multilingual version of JavaScript. It is an open source project available at `http://www.babylscript.com`. It is based on the premise that for a programming language to be multilingual it is not sufficient to simply provide translations of some keywords and function names. In a proper multilingual programming language, awareness of multiple languages must be integrated into the syntax and semantics of the language. Instead of separate incompatible language versions, Babylscript offers a single multilingual version into which different non-English programmer communities can pool their resources.

```
// English code
var win = new Window();
win.setText("Hello", Color.RED);

// French code
---fr---
win.écrireTexte(«Allô», Couleur.ROUGE);
```

Figure 1. When programming in English, Babylscript objects export an English API; when programming in French, the same objects export a French API

Although Babylscript extends JavaScript to support many different languages at the same time, one can write programs in a single language without being aware of Babylscript's support for other languages. Babylscript has different language modes. When using a French language mode, for example, all keywords, function names, and objects are in French. When Babylscript is used in the English language mode, all keywords and objects are in English, so Babylscript behaves like normal JavaScript. A programmer can write normal JavaScript code in the English language mode without any awareness of the multilingual features of Babylscript.

These language modes are not isolated silos. Programmers can switch language modes in the middle of their code, meaning that programs can be composed of a mix of different languages. Objects created in one language mode can be used by code in a different language mode. For example, a function can be defined in a Chinese language mode and then be called by code in an Arabic language mode. A Spanish programmer can take a code snippet from an English programming forum, and paste it into their Spanish code.

In JavaScript, the global scope, methods, and fields are represented as properties. A property is a data value with an associated name. Babylscript extends properties by allowing multiple names to be bound to a data value. In different language modes, different sets of names can be used to access properties. In this way, programmers can create their own libraries that have translated function and object names for different languages (Figure 1).

2. Design

The underlying basis for the design of Babylscript is the observation that punctuation and mathematical notation are fairly consistent among current natural languages. Since JavaScript's syntax makes heavy use of punctuation and mathematical notation, its syntax does not need to be significantly changed for different language modes. In Babylscript, the same grammar is used for all the different language modes, but the individual tokens for keywords and operations can differ. Similarly, the same functions, objects, and methods are used for all language modes, but the names given to these constructs can differ from language mode to language mode.

```
// Create a property with a default name
var obj = new Object();
obj['property'] = 5;

// Create a French translated name
obj['fr':'propriété'] = 'property';

---fr---
// Both default and translated names are valid
assert(obj.property == obj.propriété);
```

Figure 2. Properties can be referred to using their default name or translated name

Babylscript initially defaults to using an English language mode where keywords and library APIs are in English. To change language modes, programmers write three minus signs, the language code for the desired language mode, and then another three minus signs. For example, `---fr---` is used to switch into a French language mode. Once in a different language mode, all keywords and other symbols are translated into the given language. The command for switching language modes is always the same in all modes, so that programmers can always switch to their desired language mode without knowing the current language context.

To support having translated APIs for libraries in different language modes, Babylscript extends the object model of JavaScript. In JavaScript, objects can be viewed as associative arrays in which names are mapped to data. A data value and its associated name is called a property. In Babylscript, the name given to a property is called a *default name*. In addition to a default name, Babylscript allows properties to have multiple *translated names*. A property can have a different translated name for each of Babylscript's supported language modes. When in a certain language mode, programmers can access properties using the corresponding translated name, if one is available, or the default name (Figure 2). Translated names allow libraries to have different API translations for different language modes yet provide a reasonable fallback if no translations are provided. Since JavaScript stores global variables and functions as properties of a global "object," global variables automatically gain support for having translated names.

Babylscript also extends JavaScript's model for prototype inheritance to handle translated names. Translations can be specified in a parent object where they will be inherited by child objects. Child objects can also override the translations of their parent objects.

3. Implementation

Babylscript is implemented as a modification to the Mozilla Rhino JavaScript interpreter [1]. Since Babylscript uses the same grammar for all the different language modes, there is a single parser, bytecode generator, and bytecode interpreter for all languages. Each language mode does have a separate tokenizer for identifying keywords and other tokens though.

Since the access of the properties of an object is dependent on the language mode, the Babylscript compiler must track the language context of all operations. It does this by adding language tags to all structures passed between stages of the compiler. Figure 3 shows an example of how existing JavaScript tokens, parse trees, and bytecodes are extended with language tags.

During tokenization, the Babylscript tokenizers translate keywords into a canonical form (i.e. English) while leaving identifiers unchanged. The tokenizers also tag all tokens with the current language. These tags are propagated through the compiler chain. Since JavaScript names are resolved at runtime, the interpreter needs the tags to look-up the correct table of translated names of an object.

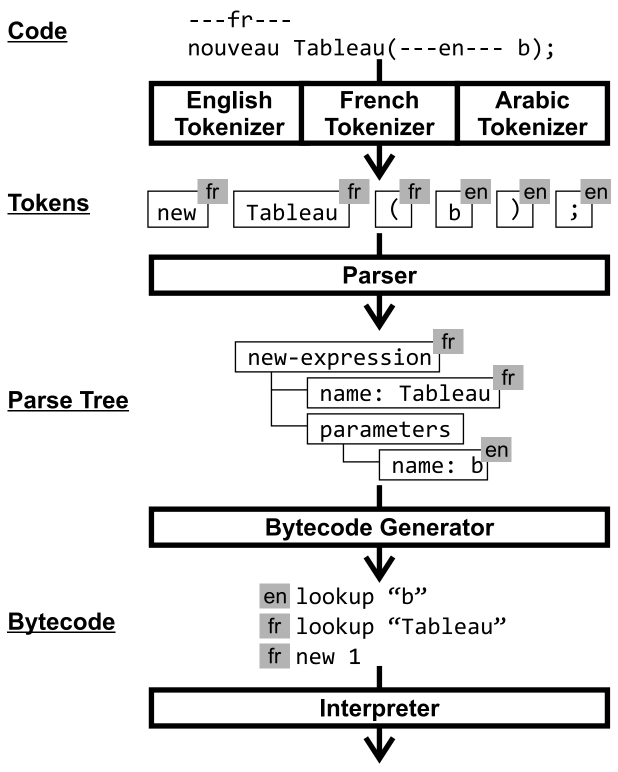

Figure 3. Babylscript modifies a standard JavaScript engine by supporting different tokenizers for different language modes and by extending different stages of the compiler with tags describing the language

The interpreter also has new instructions for setting and modifying translations of properties.

Babylscript provides translations of JavaScript's standard library for different languages. JavaScript's standard library is quite small compared to other languages, making this translation task fairly straight-forward. Unfortunately, JavaScript's standard library is also very US-centric, and Babylscript does not modify the libraries to properly support international number and date formats.

4. Conclusion

Babylscript is a multilingual version of JavaScript. It demonstrates that it is possible to design a language that supports the mixing of code written in different languages. Within each language mode, a programmer can write code using their own native languages without being aware of Babylscript's support for other languages. But programmers may unknowingly be using libraries written in a different language, and their code can, in turn, be used by code written in yet another different language.

In the future, some of the ideas behind Babylscript could be applied to statically typed languages like Java. The extra type information could potentially allow for more complex language models, improved performance from resolving translated names statically, and automated translation of program code between languages.

References

[1] Mozilla. Rhino: JavaScript for Java. `http://www.mozilla.org/rhino/` [accessed 2011-06-08].