Openwords Markup Language (OWML)

A open markup language for representing language learning problems.

Rationale

Language learning problems have their own specific elements or parameters, that are distinct from other language purposes such as machine translation. The Openwords Markup Language is intended to represent language learning problems to allow software programs like the Openwords webapp to read these problems and present them to learners. Additionally, a standardized problem format will allow computational linguists, technically trained language instructors, and programmers to automate the conversion of information from a variety of existing language databases into dynamic learning problems for use on mobile web applications (e.g. Openwords). These existing databases can include dictionaries (e.g. Wiktionary), parallel texts (e.g. Tatoeba.org), or existing paper based worksheet language learning problems.

Openwords is open source, so the code has certain goals, namely being human readable when possible, simple, text based, and sufficient to the task of representing a wide variety of language learning problems. This markup language is specifically ***not*** intended to represent UI elements, such as shape of button, the font of the text, color, or dimensions of the screen.

Purpose
The OWML code should represent the logic or essential content of a language learning problem. OWML is not intended to represent features of no essential interest to a language teacher.

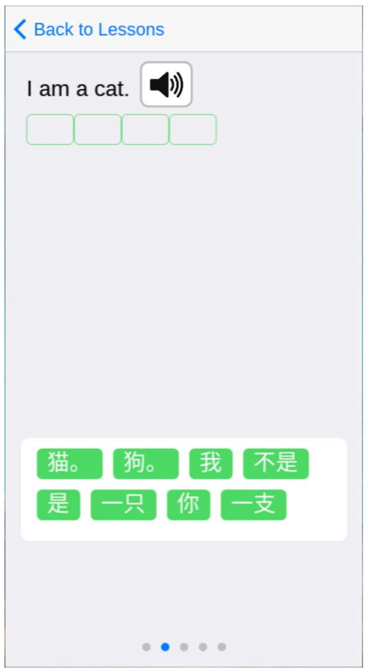
Inspiration
OWML was inspired by other markup languages like Markdown and open formats like Bibtex for citations.

Elements of the OWML

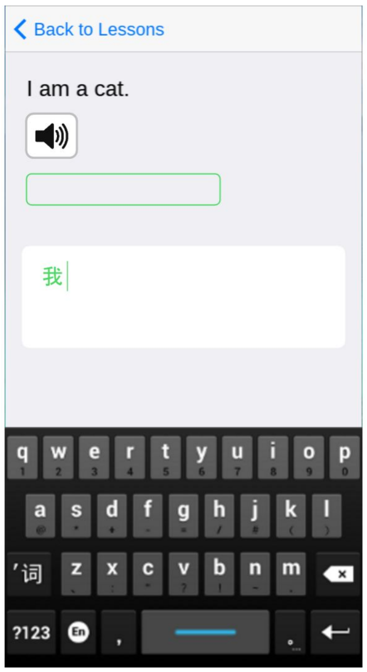
Let's start by providing three example problem expressed with OWML. Try to imagine what these problems might look like now. Below, we'll walk step by step through these problems going over every element of syntax and content. At the end of this document you will be able to read these three examples. With some creativity you may be able to imagine new types of problems consisting of different arrangements of fundamental elements.

Three Examples

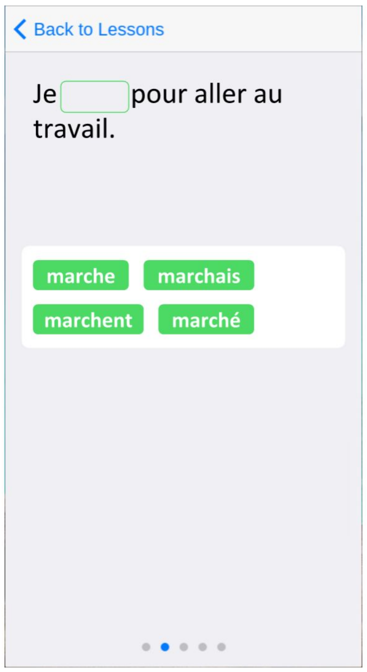
```
=ss
*[I am a cat.][](sound-out::URL)(image-out::URL)
*[@][@][@][@]
#[我]
#[是]
#[一只]
#[猫。]
%[你][不是][狗。][一支]

=ss
*[I am a cat.]
*[](sound-out::URL)(image-out::URL)
*[]t
#[我是一只猫。][我是一只猫]
%

=fb
*[Je ][][ pour aller au travail.]
#[marche]
%[marché][marchent][marchais]
```



I am a cat.



I am a cat.



Je pour aller au travail.

The OWML contains regular text along with markup syntax that instructs how the problem reader (e.g. Openwords webapp) should interpret the text. Every line of the OWML starts with a special character which indicates the type of information presented in that line. Let's start with the first line of a problem.

=
The first line starts with an '=' character and is followed by a two or three letter code. This line indicates a "***problem type***." Currently this line is ignored by the problem reader program and instead the "type" of the problem is entirely manifested by the arrangement and content of other components of the OWML for a particular problem. However this first line, the "***problem type***" line, is essentially a locus for future refinement of problem metadata.

*
The next line/s within an OWML problem are "***problem***" lines. These lines represent text, blanks, audio, or visuals presented to the learner. The vertical sequence of these lines does matter, with earlier problem lines appearing above later problem lines in the problem visualization.

The problem line/s are the first lines to contain several kinds of new elements/syntax. We'll go through those right now.

1) First, problem lines contain "***elements***" or element syntax. Elements are blocks of text, special characters (or an absence of characters) enclosed by square brackets. Look at the first problem line of the Example problem 2.
*[I am a cat.]

This element will present the flat text 'I am a cat' to the learner.

2. Next, problem lines can have "***blanks***." Blanks are a particular kind of element indicating empty spaces that are to be filled in by the learner either by selecting available answers (described below) or by typing in answers (also described below). A blank is represented in code with square brackets surrounding no characters or the @ symbol. As follows.
[] or [@]

Blanks can be interspersed with simple text elements as the following example illustrates (from the third problem above).
*[Je][][pour aller au travail.]

3. Next, problem lines can have "***external resources***" External resources are image, audio, or video files and are represented in OWML with parentheses with internal parameters separated by double colons as follows.
(image-out::URL)

The first parameter specifies the type of external resource (e.g. image-out, sound-out, video-out). The second parameter of an external resource represents the path, usually a URL, to the external resource file. In the example above we represent this simply with “URL” in a real example there would be a valid URL. Remaining parameters are available for additional external resource metadata.

If the external resource follows an element in the OWML, then the external resource is intended to be “attached” to this element. For example, in this example an audio file is attached to a text element. This would be displayed as the audiotext ‘I am a cat.’ If a user tapped the text they would hear the sound specified in the external resource element.
*[I am a cat.](sound-out::URL)

Next, through attachments, we can see another use of “blank” elements. Blank elements can also represent placeholders. Let’s take a look a look at one problem line from the first problem.
*[I am a cat.][](sound-out::URL)(image-out::URL)

In this problem line we actually have two external resources. These external resources are attached *to each other* and then they are attached to a blank element []. This blank element is not displayed as a blank but is instead a placeholder for the external resources. In this case the two external resources are not attached to the text element ‘I am a cat.’ but are instead attached to each other. The image can be a common sound display image such as seen in the the first example problem. This problem line displays as the flat text ‘I am a cat.’ followed by a sound icon image that plays the audio file when the learner taps this image.

Now we have introduced the problem lines and we have also introduced a variety of OWML elements.

#
The next type of OWML lines are “***answer***” lines. These lines begin with the # special character. These lines are particularly relevant when previous “problem” lines contain blank elements. Generally for every blank element (that isn’t a placeholder) contained within the problem lines of an OWML problem there should be a corresponding number of answer lines in an appropriate sequence. Let’s highlight the relevant part from the first of three examples above.

```
=ss
*[I am a cat.][](sound-out::URL)(image-out::URL)
*[@][@][@][@]
#[我]
#[是]
#[一只]
#[猫。]
%[你][不是][狗。][一支]
```

This problem above has four blanks in a row in one problem line. Thus, there are four answers in four lines, which would appear as '我是一只猫。' when written out. If multiple correct answers are possible for a particular blank. This can be represented by simply having alternative elements represented on the relevant answer line.
#[alternative1][alternative2]

We have not determined exactly how to represent several alternative correct answer element arrangements. This can be the subject of discussion between Openwords and CommonSpaces and other stakeholders.

Let's look at example problem 2 to explore other kinds of "answers." The relevant sections of the problem are highlighted. Here, the problem line contains a blank element with a postfix 't' following the blank element. This indicates that the corresponding answer is to be *typed* by the learner. Thus, in the answer line below, the entire string is presented as an answer, rather than as a series of individual elements for individual blanks. In example problem 2, there are actually *two correct answer strings*.

```
=ss
*[I am a cat.]
*[](sound-out::URL)(image-out::URL)
*[]t
#[我是一只猫。][我是一只猫]
%
```

%
The final type of line in the OWML are "***marplot***" lines or "***incorrect answer***" lines. These lines begin with the % special character. Marplot lines contain incorrect answers. Example problem 1 has incorrect answers, while example problem 2 does not.

Typing problems do not display incorrect answers because the learner is compelled to *recall* or generate a correct answer rather than to *recognize* correct answers or answer arrangements.

Summary

Now we have gone through all types of lines in a particular OWML problem. OWML currently has roughly ten semantic elements. As follows.

1. Problem type line specified by =
2. Problem line/s specified by *
3. Answer line/s specified by #
4. Marplot or incorrect answer line/s specified by %
5. Elements specified by square brackets [element]

6. Blank elements specified by no characters within square brackets [], or an ‘at’ character within square brackets [@]
7. Placeholder blanks, that allow external resources to be placed in problems without attaching to other elements.
8. External resources specified by parentheses with parameters separated by double semicolons (parameter1::parameter2::parameter3)
9. Postfixes like the ‘t’ character or external resources attached to elements.
10. And of course, text itself found within various elements.

Future Elements and Syntax
As we develop the OWML new elements of syntax will need to be designed and chosen by consensus methods in order to expand the language to accommodate a diverse variety of language learning problems.

Some possible future elements might include but would not be limited to the following.

1)
Reveal elements - Elements that are revealed after a sequence of learner actions, or timed appearance elements.

2)
Lesson metadata - Lessons consist of many OWML problems in sequence separated by blank lines. A lesson itself may require metadata header lines. For example such metadata could specify the Language1 and Language2 of the lesson, the number of problems the learner is allowed to get incorrect, and other lesson global parameters.

3)
Comment syntax, preliminarily we can simply choose the following.
//commentHere
And
/*commentHere*/