Bible Online Learner:
Technical Documentation

Claus Tøndering
Ezer IT Consulting

2 May 2018

Copyright © 2018 by Claus Tøndering, claus@ezer.dk

The document is made available under a Creative Commons Attribution 4.0 International License
(see http://creativecommons.org/licenses/by/4.0/)

Contents

Contents 2

1 Introduction 5
- 1.1 License and Copyright . . . 5
- 1.2 How To Read This Document . . . 5
- 1.3 Terminology . . . 5

2 History 8
- 2.1 EQG: A Java Applet (2008) . . . 8
- 2.2 3ET: A Stand-alone Java Program (2009-2010) . . . 8
- 2.3 PLOTLearner: A EuroPLOT Product (2011-2013) . . . 8
- 2.4 Bible Online Learner: Web-based (since 2013) . . . 9

3 Installing Bible OL 10
- 3.1 Hosting Bible OL . . . 10
- 3.2 Bible OL Development System Setup . . . 14

4 Programming Languages and Frameworks 17
- 4.1 PHP . . . 17
- 4.2 CodeIgniter . . . 17
- 4.3 SQL . . . 17
- 4.4 MQL . . . 17
- 4.5 HTML . . . 18
- 4.6 CSS . . . 18
- 4.7 Less . . . 18
- 4.8 JavaScript . . . 18
- 4.9 TypeScript . . . 18
- 4.10 JSON . . . 18
- 4.11 jQuery and jQuery UI . . . 18
- 4.12 Bootstrap . . . 19
- 4.13 RGraph . . . 19
- 4.14 What You Must Know . . . 19

5 High-level System Architecture 20

6 Understanding Emdros 22

7 Source Code Tree Overview 24

8 Emdros Databases in Bible OL 27
- 8.1 The *visual* Feature . . . 27
- 8.2 ETCBC4 . . . 27
- 8.3 Nestle1904 . . . 30

8.4 A Note on Greek Accents in Unicode . . . 31

9 Database Description Files **32**
9.1 Database Specification File: PRIM.db.json . . . 33
9.2 Database Description Files for Old Databases . . . 46
9.3 Database Type Information File: PRIM.typeinfo.json . . . 46
9.4 Database Book Order File: PRIM.bookorder . . . 47

10 Quiz Templates **49**
10.1 `<sentenceselection>` . . . 50
10.2 `<quizobjectselection>` . . . 53
10.3 `<quizfeatures>` . . . 53
10.4 Templates Using MQL for Selection . . . 54

11 Multiple-Choice Questions **56**

12 Hints **58**

13 Data Exchange **59**
13.1 Displaying Text . . . 59
13.2 Running an Exercise . . . 60

14 The *dictionaries* Variable **62**
14.1 The TypeScript *DictionaryIf* and PHP *Dictionary* Classes . . . 62
14.2 The *MonadObject* Class and Its Subclasses . . . 66
14.3 The *MatchedObject* Class . . . 67

15 The *quizdata* Variable **69**
15.1 The TypeScript *QuizData* and PHP *Quiz_data* Classes . . . 69

16 The CodeIgniter Framework **72**
16.1 URL Decoding . . . 72
16.2 Model-view-controller Structure . . . 72
16.3 Library Functions . . . 73
16.4 Adding Code . . . 73

17 Server Code **74**
17.1 Models, Views, and Controllers . . . 74
17.2 MQL Requests in the Server . . . 76
17.3 Google and Facebook Login . . . 77
17.4 Account Expiry . . . 80

18 User Database **81**
18.1 The *user* Table . . . 82
18.2 The *class* Table . . . 82
18.3 The *userclass* Table . . . 83
18.4 The *userconfig* Table . . . 83
18.5 The *alphabet* Table . . . 83
18.6 The *font* Table . . . 84
18.7 The *personal_font* Table . . . 84
18.8 The *exercisedir* and *classexercise* Tables . . . 84
18.9 The *exerciseowner* Table . . . 85
18.10 The *bible_refs* Table . . . 85

18.11 The *bible_urls* Table . . . 85
18.12 The Statistics Tables . . . 86
18.13 The *heb_urls* Table . . . 89
18.14 The *migrations* Table . . . 89
18.15 The Localization Tables . . . 89

19 Less Style Sheets **93**

20 Client Code **95**
20.1 TypeScript . . . 95
20.2 The *ol* Client Code . . . 95
20.3 The *editquiz* Client Code . . . 99

21 Internationalization and Localization **100**
21.1 CodeIgniter's Internationalization Mechanism . . . 100
21.2 Bible OL's Modifications to CodeIgniter's Mechanism . . . 100
21.3 Internationalization of the Client Code . . . 101
21.4 Grammar Localization Structure . . . 101
21.5 Grammar Localization Comments . . . 107
21.6 Lexicon Localization . . . 108
21.7 Importing and Exporting Translations . . . 108

22 Plug-ins **110**
22.1 The Learning Journey Plug-in . . . 111

23 Complementary Websites **112**
23.1 The Resource Website . . . 112
23.2 The SHEBANQ Website . . . 113

A ETCBC4 Details **114**
A.1 The *word* Object . . . 114
A.2 The Other Object Types . . . 124

B ETCBC4 Words Database **125**
B.1 The Origin of the ETCBC4 Words Database . . . 125

C ETCBC4 Hints Database **126**
C.1 The Origin of the ETCBC4 Hints Database . . . 126

D Nestle1904 Details **127**
D.1 The *word* Object . . . 127
D.2 The *sentence* Object . . . 130
D.3 The *clause1* and *clause2* Objects . . . 130
D.4 The Other Object Types . . . 131
D.5 The Origin of the Data . . . 131

Index **132**

Chapter 1

Introduction

You must read this chapter.

This document gives a detailed technical description of the internal workings of Bibel Online Learner (Bible OL). The document is intended for people who need to install Bible OL on their own server and developers who need to modify or expand the way Bible OL works.

1.1 License and Copyright

Except where otherwise noted, the Bible OL source code is distributed under an MIT License[1]. The code is Copyright © 2018 by Claus Tøndering, claus@ezer.dk.

The present document is made available under a Creative Commons Attribution 4.0 International License[2].

1.2 How To Read This Document

If you are reading this document because you want to host a Bible OL server, you should read chapter 3. The rest of the document is optional reading for you.

If you are reading this document because you want to modify or enhance Bible OL, you should note the first paragraph of each chapter. It will help you decide if you need to read a particular part of this document.

On page 132 you will find an index, which may help you find your way through this document.

1.3 Terminology

Unfortunately, various parts of the system do not use a completely uniform terminology. This section gives an overview of what you may come across here:

Term	Meaning
quiz exercise	These terms are synonymous. They refer to an exercise executed by a user.

(Continued...)

[1] http://opensource.org/licenses/MIT
[2] http://creativecommons.org/licenses/by/4.0/

Term	Meaning
quiz template	A file located in the `quizzes` directory. In XML format it describes how Bible OL should generate questions for an exercise.
question	An exercise consists of several questions. Figure 1.1 shows *one* question.
question item	Each question in an exercise contains several question items. The question in Figure 1.1 has *two* question items.
sentence unit quiz object question object	These terms are synonymous. The terms refer to the Emdros objects that are the subject of a question item. They are marked in red in the question text. (See Figure 1.1.)
display feature	An Emdros feature presented to the user as part of a question item (see Figure 1.1).
request feature	An Emdros feature which the user must provide as part of a question item (see Figure 1.1).
passages universe	The collection of Bible passages from which an exercise draws its sentences. The term "universe" is found in older parts of the code.
grammar selection box	A box found on the left part of the display where the user can select what in-line grammatical features to display. See Figure 1.2.
grammar information box	A box found on the right part of the display containing grammatical information about the word under the mouse pointer. See Figure 1.2.
grammar hierarchy	The organization of components of text. At the lowest level of the grammar hierarchy we have the *words*. Above that we may find *phrases*, then *clauses*, and finally *sentences*. The exact words for the levels of the grammar hierarchy varies between databases.

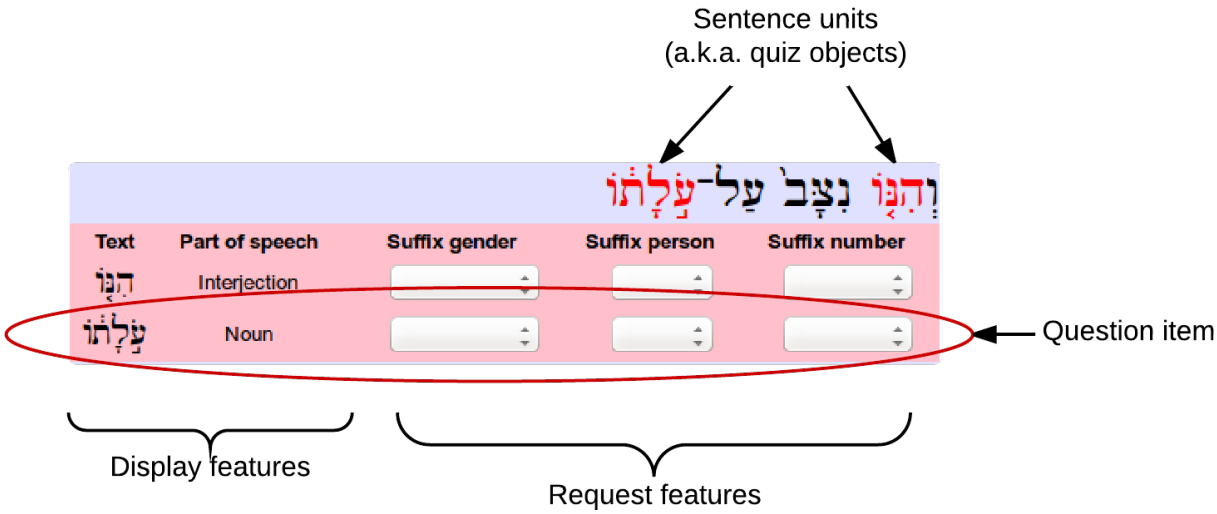


Figure 1.1: A question containing two question items.

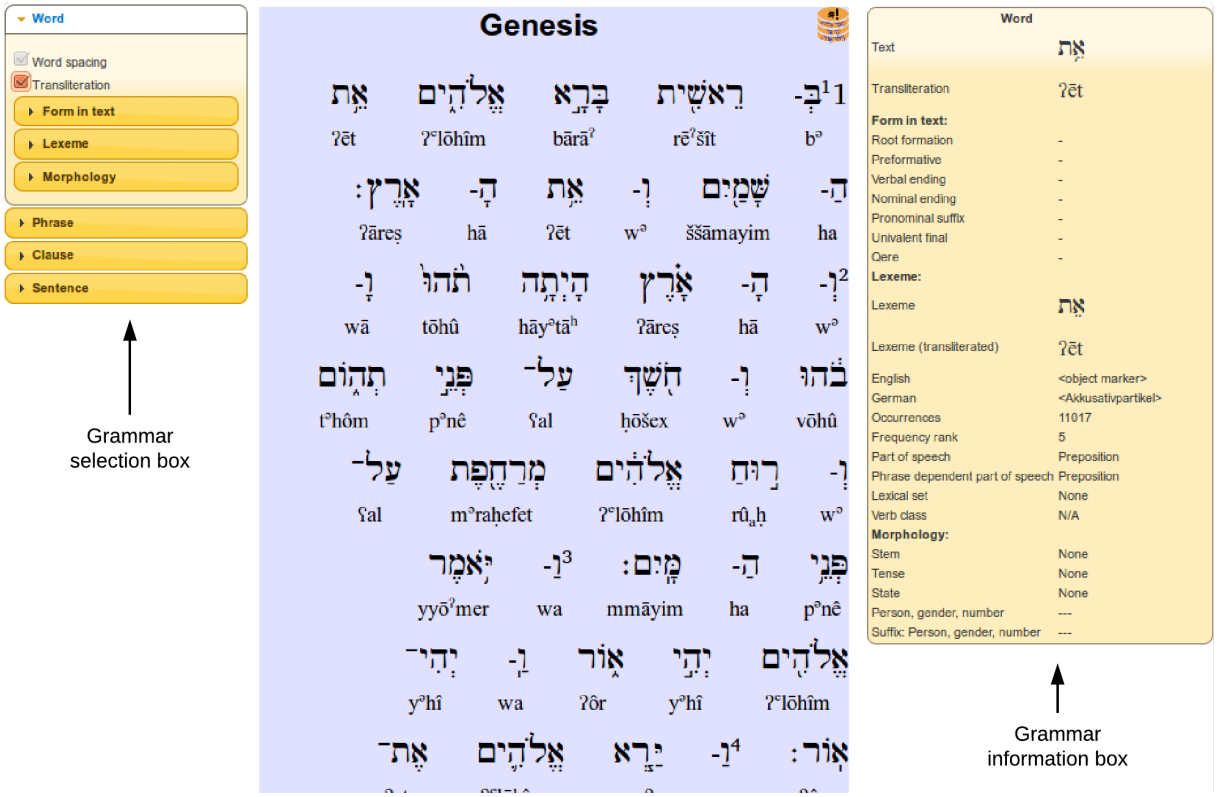


Figure 1.2: The grammar selection box and the grammar information box.

Chapter 2

History

Read this chapter if you want to.

2.1 EQG: A Java Applet (2008)

In 2008 professor Nicolai Winther-Nielsen told me about a text database system, Emdros, developed by Ulrik Sandborg-Petersen. Nicolai was teaching Biblical Hebrew at what was then the Copenhagen School of Theology.[1]

I further learned that Emdros databases exist with the entire text of the Bible in the original languages, complete with grammatical information about every single word.

We discussed how these tools could be used by Nicolai in his teaching, and in the autumn of 2008 the first version of *EQG*, the *Emdros-based Quiz Generator*, was demonstrated, running as a Java applet in a web browser.

2.2 3ET: A Stand-alone Java Program (2009-2010)

The Java applet solution was not practical, especially since there was no easy way to access an Emdros database over a network connection. In 2009 the applet was therefore abandoned in favour of a stand-alone PC application, still written in Java but running directly under Microsoft Windows.

The name was changed to *3ET, Ezer's Emdros-based Exercise Tool*, and an attempt was made to market it through 3BM, a company owned by Nicolai and his colleague Jens Bruun Kofoed.

The name 3ET is still reflected in the file extension `.3et` used in quiz template files.

2.3 PLOTLearner: A EuroPLOT Product (2011-2013)

In 2011 3ET became part of an EU project about *Persuasive Learning Objects and Technologies*, PLOT. The project became known as EuroPLOT,[2] and 3ET changed its name to *PLOTLearner*.

Since PLOTLearner was part of an EU project the source code was made open under an MIT License.[3] PLOTLearner was the last Java-based version of the product, and it can be downloaded from `http://eplot.3bmoodle.dk/index.php/downloads`.

[1] A.k.a. Dansk Bibel-Institut. This later became the Fjellhaug International University College Denmark.
[2] `http://www.eplot.eu`.
[3] `http://opensource.org/licenses/MIT`.

2.4 Bible Online Learner: Web-based (since 2013)

Since 2013 the program has been moved from a Java-based stand-alone PC application to a web-based solution. The name of the product was changed once more and became *Bible Online Learner*, or *Bible OL* for short. This is the product that is described in this document.

Chapter 3

Installing Bible OL

Read as much of this chapter as you find necessary.

This chapter describes how to install Bible OL on a computer. What you need to do, depends on what you mean by "installing" Bible OL. If you want to set up a server that runs Bible OL for the benefit of researchers and students, follow the instructions in Section 3.1. If you want to get your own copy of Bible OL in order to enhance or modify it, follow the instructions in Section 3.2.

3.1 Hosting Bible OL

Bible OL runs on a Linux server. I'm sure that it would be quite easy to host it on a Mac or a Windows computer, but I haven't tried this myself, and the following instructions are aimed at people with a computer running the Linux operating system.

3.1.1 Step 1. Server Software

The following software must be installed on your server (please pay attention to the PHP version number):

- Apache (web server). I use version 2.4.18, but version 2.2.22 also works.
- MySQL (database system). I use version 5.7.21.
- SQLite3 (database system used by Emdros). I use version 3.11.0.
- PHP. I use version 7.1.
- git. I use version 2.7.4.

All of this can be installed on an Ubuntu Linux system using the usual *apt-get* installation program. I have not done a thorough investigation into the minimum required versions of these software packages; you should not see the versions mentioned above as minimum requirements.

3.1.2 Step 2. Install Emdros

You must install the Emdros database system. This is probably the most complex part of the installation process since Emdros is not a standard component in any operating system.

I use Emdros version 3.5.23. Emdros can be downloaded from `http://sourceforge.net/projects/emdros/files/emdros`.

Follow the instructions that come with Emdros to compile and install it on your computer. You will need a C++ compiler and the Gnu make system (and probably a few other things as well – check the Emdros documentation).

When you compile Emdros, be sure to include support for PHP 7. I use this *configure* command to set up the Emdros compilation:

```
./configure --with-sqlite3=local --with-mysql=yes --with-default-backend=sqlite3 \
    --with-swig-language-php=yes --with-swig-language-java=no --with-swig-language-python=no \
    --with-swig-language-perl=no --with-wx=no --with-swig-language-ruby=no \
    --with-swig-language-php7=yes --with-swig-language-nodejs=no
```

If you want to include support for other programming languages, you will need to change this command.

When you have compiled and installed Emdros (using the "make" and "sudo make install" commands, respectively) you need to enable access to Emdros in PHP. The correct way to do this depends on your Apache and PHP installation. In my case, I had to copy the file `EmdrosPHP7.ini` to the directory `/etc/php/7.1/mods-available` and then run the command

```
sudo phpenmod -v 7.1 EmdrosPHP7
sudo service apache2 restart
```

If you cannot or do not want to enhance PHP with Emdros support, you can configure Bible OL to use the *mql* command line tool instead, but this is not recommended. More information about that is given in Section 17.2.3.

If you already have Emdros installed on a development system, and if your runtime system uses the same operating system as your development system, you can simply copy the Emdros files to your runtime systems. The files to copy will be typically found in `/usr/local/bin` and `/usr/local/lib/emdros`. (I have never actually done this, and I suspect that you may need to run *ldconfig* after copying the files to your runtime system.)

3.1.3 Step 3. Download Bible OL

You can download Bible OL from GitHub using these commands:

```
cd installation directory
git clone --recursive https://github.com/EzerIT/BibleOL
```

This will fetch all the Bible OL software, including five submodules needed by Bible OL. The *git* command will create a directory called `BibleOL` under the current directory.

Note: If you have forked Bible OL on GitHub you should replace the URL in the "git clone" command above with a URL that points to your repository on GitHub. (See section 3.2.1 for more information.)

When the download completes, execute the following commands:

```
cd BibleOL
git-hooks/setup.sh
```

This will install a Git hook that automatically downloads the necessary databases from Dropbox when needed.

3.1.4 Step 4. Configure MySQL

Create an empty database in MySQL. Then copy the file `myapp/config/database.php-dist` to `myapp/config/database.php` and modify the following lines in the copy:

```
'username' => 'USERNAME',
'password' => 'PASSWORD',
'database' => 'DATABASE',
```

Change the text USERNAME, PASSWORD, and DATABASE to be the database username, password, and database name.

3.1.5 Step 5. Additional Configuration

Copy the file `myapp/config/ol.php-dist` to `myapp/config/ol.php` and modify the following lines in the copy:

```
$config['pw_salt'] = 'xxxxxxx';
$config['mql_driver'] = 'native';
$config['mail_sender_address'] = 'xxxxx@xxxxx.xx';
$config['mail_sender_name'] = 'Bible Online Learner';

$config['google_login_enabled'] = false;
$config['google_client_id'] = 'xxxxxxxxxxxx.apps.googleusercontent.com';
$config['google_client_secret'] = 'xxxxxxxxxxxxxxxxxxxxxxxx';

$config['facebook_login_enabled'] = false;
$config['facebook_client_id'] = 'xxxxxxxxxxxxxxx';
$config['facebook_client_secret'] = 'xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx';
```

You should modify these values thus:

Variable	Contents
`$config['pw_salt']`	A random text string of, say, 6-8 characters. This is used to randomize the user passwords stored in the user database.
`$config['mql_driver']`	Set to `'native'` to select a built-in MQL driver. Set to or `'extern'` to run MQL commands external to the PHP interpreter (see Section 17.2.3).
`$config['mail_sender_address']`	The email address to be used as sender when Bible OL sends email to users.
`$config['mail_sender_name']`	The name to be used as sender when Bible OL sends email to users.
`$config['google_login_enabled']`	Set to *true* if you have a Google apps account that allows you to service Google logins; set to *false* to disable Google login.
`$config['google_client_id']`	Your Google apps client ID. Used only if you enable Google login.
`$config['google_client_secret']`	Your Google apps client secret, if any. Used only if you enable Google login.
`$config['facebook_login_enabled']`	Set to *true* if you have a Facebook developers account that allows you to service Facebook logins; set to *false* to disable Facebook login.
`$config['facebook_client_id']`	Your Facebook app ID. Used only if you enable Facebook login.
`$config['facebook_client_secret']`	Your Facebook app secret. Used only if you enable Facebook login.

Copy the file `myapp/config/config.php-dist` to `myapp/config/config.php` and modify the following line in the copy:

```
$config['base_url'] = 'http://example.com';
$config['lj_enabled'] = false;
```

Replace 'http://example.com' with the top URL of the web site for your Bible OL installation.
If you plan to use *Learning Journey* (see Section 22.1), set `$config['lj_enabled']` to `true`.

3.1.6 Step 6. Initialize MySQL

The following commands should be executed with the current directory set to `BibleOl`.

Substep 1

Initialize the contents of the database from the content of the file `bolsetup.sql`.

If you plan to use *Learning Journey* (see Section 22.1), you may also want to import the content of the file `ljsetup.sql`.

Substep 2

Populate the database with the translations of the interface by issuing these commands:

```
for i in comment da en es pt zh-simp zh-trad
do
    echo Importing language files $i
    php index.php translate if_php2db $i myapp/language/langsrc/$i
done
```

(On my computer, this takes less than 20 seconds.)

Substep 3

Update the database with the translations of the grammar terms by issuing this command:

```
php index.php translate gram_prop2db
```

The will output lines such as this:

```
  Handling file nestle1904.comment.prop.pretty.json - unchanged
```

Some files may be marked "unchanged", others may be marked "updated +++++". Both are normal.

Substep 4

Populate the database with Hebrew and Greek lexicons by issuing these commands:

```
for s in aram heb
do
    for d in en de da
    do
        echo Source language: $s. Destination language: $d.
        php index.php translate import_lex $s $d lexicons/${s}_${d}.csv
    done
done

echo Source language: greek. Destination language: en.
php index.php translate import_lex greek en lexicons/greek_en.csv
```

(On my computer, this takes about a minute.)

3.1.7 Step 7. Add Administrator

You must manually add an administrator to the MySQL database. You do this by adding an entry to the table *bol_user* (see Section 18.1). The entry should have the *isadmin* field set to 1 (*true*). The *password* should be set to the value of this PHP function:

```
md5($config['pw_salt'] . 'xxxxx');
```

where *$config['pw_salt']* is configured in `myapp/config/ol.php`, and *xxxxx* is the user's desired password.

You can also find the correct value of the password by executing this shell command:

```
echo -n sssssxxxxx | md5sum
```

where *sssss* is the value of *$config['pw_salt']* and *xxxxx* is the user's desired password.

3.1.8 Step 8. Apache Configuration

Set up the Apache web server to access Bible OL. Make sure that Apache is configured to allow `.htaccess` files. This is controlled by the Apache *AllowOverride* directive.

Copy the file `.htaccess-dist` to `.htaccess` and modify the following line in the copy:

```
RewriteBase /
```

The correct value here depends on your Apache configuration. If your web server is set up to serve Bible OL at the url `http://example.com/alpha/beta`, then the above line should be changed to:

```
RewriteBase /alpha/beta
```

If you have a dedicated hostname for Bible OL (for example, `http://example.com`), you should leave *RewriteBase* as it is: a single slash.

3.1.9 Step 9. Set Up Quiz Template Directory

Create a directory called `quizzes` in the installation directory. Quiz template files will be stored here. If you want to, you can copy the contents of the `quiz_templates` directory, which contains sample quiz templates, to `quizzes`.

Make sure that the permissions on the `quizzes` directory and its contents is set to allow the web server to modify the files.

3.2 Bible OL Development System Setup

This section describes how to set up a complete development system for working with all aspects of Bible OL development. Depending on the type of development you are going to do, you may not need all of this.

You may want to read Chapter 4 before you proceed with the following.

Section 3.1.3 describes how to download Bible OL. If you plan to make any modifications to the software, I recommend that you fork Bible OL on GitHub before downloading it. See section 3.2.1 for more information.

If you want to test Bible OL on your own computer, you should also set it up as a Bible OL server. Please see Section 3.1 for information about how to do this.

Much of the description here is quite vague. In many cases I am simply describing what *I* have done. Your system may be different, and you may need to do things I have not described.

For Bible OL, I use a computer with the Linux operating system (the Ubuntu distribution). I am sure the system could also be set up on a Windows or Mac computer, but I have not tried it and you will not find any instructions how to do it here.

The following should be installed:

- SQLite3 (database system used by Emdros). I use version 3.11.0.
- PHP. I use version 7.1.
- nodejs (JavaScript runtime, used for compiling TypeScript and Less). I use version 4.2.6.
- npm (package manager for nodejs). I use version 3.5.2. This package may not strictly be required, but it is useful for installing nodejs modules.

- git. I use version 2.7.4.

All of this can be installed on an Ubuntu system using the usual *apt-get* installation program. I have not done a thorough investigation into the minimum required versions of these software packages; you should not see the versions mentioned above as minimum requirements.

You will also need to install the Emdros database system, a Less compiler, and a TypeScript compiler. This is detailed in Sections 3.2.2-3.2.4.

3.2.1 Forking Bible OL on GitHub

If you plan to make any modifications to the software, I recommend that you fork Bible OL on GitHub before downloading it. In order to do this you must set up an account on https://github.com. After this, navigate to https://github.com/EzerIT/BibleOL and click the "Fork" label in the upper right section of the web page.

Please note that this document is not a manual on how to use Git and GitHub. You are expected to know this.

Section 3.1.3 tells you how to download Bible OL from the original repository. To download from your own fork, you should use these commands:

```
cd installation directory
git clone --recursive https://github.com/username/BibleOL
```

or

```
cd installation directory
git clone --recursive git@github.com:username/BibleOL.git
```

depending on which way you prefer to access GitHub. (Replace *username* in the commands above with your GitHub username.)

In either case, remember to execute these commands:

```
cd BibleOL
git-hooks/setup.sh
```

as described in Section 3.1.3.

Bible OL uses a number of Git submodules: CodeIgniter, ckeditor, zocial, jstree, and virtualkeyboard. The "--recursive" flag in the "git clone" command causes these submodules to be cloned from the "EzerIT" repository. You may want to replace them with your own copies of the repositories. The following table lists their names and their location on GitHub:

Submodule	My forked location	Original location
bootstrap	EzerIT/bootstrap	twbs/bootstrap
ckeditor	EzerIT/ckeditor-releases	ckeditor/ckeditor-releases
zocial	EzerIT/css-social-buttons	samcollins/css-social-buttons
jstree	EzerIT/jstree	vakata/jstree
virtualkeyboard	EzerIT/virtualkeyboard	*Not on GitHub*

Previous versions of Bible OL also used a submodule called CodeIgniter. This is no longer a submodule but an integrated part of the Bible OL software tree; it is located in the directory `CodeIgniter_Local`.

3.2.2 Installing Emdros

In order to install Emdros, you should follow the instructions given in in Section 3.1.2.

3.2.3 Installing Lessc

Lessc is the Less compiler. It runs under *nodejs*, which is a stand-alone JavaScript runtime system. For information about how to install and use the Less compiler, see http://lesscss.org. There, they recommend that *lessc* be installed using this shell command:

```
npm install -g less
```

You may not need the "-g" option. If you do include it, you may need to prefix the command by "sudo".

I use version 3.0.1 of *lessc*.

3.2.4 Installing Tsc

Tsc is the TypeScript compiler. It runs under *nodejs*, which is a stand-alone JavaScript runtime system. For information about how to install and use the TypeScript compiler, see http://www.typescriptlang.org. There, they recommend that *tsc* be installed using this shell command:

```
npm install -g typescript
```

You may not need the "-g" option. If you do include it, you may need to prefix the command by "sudo".

I use version 2.8.1 of *tsc*.

Chapter 4

Programming Languages and Frameworks

As a developer, you must read this chapter.

A considerable number of programming languages and other specification languages are used in the creation and execution of Bible OL. This chapter gives a brief overview of these languages and points you to where you may learn more about them.

4.1 PHP

The main language used on the server side is PHP[1], which is a popular general-purpose scripting language that is especially suited to web development.

In order to execute Bible OL, the PHP implementation on the server must be enhanced with features to execute MQL commands. This is described in Section 3.1.2.

4.2 CodeIgniter

Bible OL uses a PHP framework known as *CodeIgniter.*[2] More information about this is given in Chapter 16.

4.3 SQL

SQL[3] is a language for manipulating a relational database. Bible OL uses the MySQL database[4] system to store information about users who have an account on the Bible OL website. Bible OL uses the SQLite3 database system to store the "Words Database" (see chapter 11) and the "Hints Database" (see Chapter 12).

SQL commands are executed from PHP code through CodeIgniter.

4.4 MQL

MQL is a language for manipulating Emdros[5] text databases. The PHP implementation on the server must be extended with function to execute MQL commands. This is described in Section 3.1.2.

MQL and Emdros are described in greater detail in Chapter 6.

[1] http://php.net.
[2] http://www.codeigniter.com.
[3] See https://en.wikipedia.org/wiki/SQL.
[4] http://www.mysql.com.
[5] http://emdros.org.

4.5 HTML

The generated web pages use HTML version 5.[6]

4.6 CSS

CSS (Cascading Style Sheets)[7] is a language for specifying the layout style of a web page. However, only a small part of the Bible OL styles are written directly in CSS. Most styling is written in Less which is then compiled into CSS.

4.7 Less

Less[8] is a CSS pre-processor, meaning that it extends the CSS language, adding features that allow variables, mixins, functions and many other techniques that allow you to make CSS that is more maintainable, themable and extendable.

Although Less style files can be compiled when used in a browser, the Bible OL implementation compiles Less files only once and stores the resulting CSS files.

More information about how Less is used is given in Chapter 19.

4.8 JavaScript

On the client side (that is, in the user's browser) the software is loaded as JavaScript[9] code. However, only a small part of Bible OL is written directly in JavaScript. Most client-side software is written in TypeScript which is then compiled into JavaScript.

4.9 TypeScript

TypeScript[10] is a superset of JavaScript that adds strong typing and proper classes to JavaScript.

Most of the client-side software of Bible OL is written in TypeScript which is then compiled into JavaScript.

More information about how TypeScript is used is given in Section 20.1.

4.10 JSON

JSON[11] is a text-based data-interchange format. It is used to transfer data between the server and the client.

4.11 jQuery and jQuery UI

On the client side Bible OL uses a JavaScript/TypeScript framework known as *jQuery*[12] and its associate user interface functions *jQuery UI*.[13]

[6] See https://en.wikipedia.org/wiki/HTML5.
[7] See https://en.wikipedia.org/wiki/Cascading_Style_Sheets
[8] http://lesscss.org.
[9] See https://developer.mozilla.org/en-US/docs/Web/JavaScript.
[10] http://www.typescriptlang.org.
[11] http://json.org.
[12] https://jquery.com.
[13] https://jqueryui.com.

4.12 Bootstrap

On the client side Bible OL uses a JavaScript framework known as *Bootstrap.*[14] Currently, Bible OL uses Bootstrap version 3.3.5 with a few minor modifications.

4.13 RGraph

Bible OL can display graphs showing statistics about how students are doing. The graphs are constructed using *RGraph.*[15] Currently, Bible OL uses RGraph version 4.63.

4.14 What You Must Know

If you plan to modify the Bible OL server-side code, you must know how to program in PHP, and you must understand the CodeIgniter framework. You may also need to have a good understanding of HTML, Less, CSS, SQL, MQL, and JSON.

If you plan to modify the Bible OL client-side code, you must know how to program in TypeScript, and you must understand the jQuery framework, the Bootstrap framework, and, perhaps, the jQuery UI functions. You may also need to have a good understanding of HTML, CSS, JavaScript, and JSON.

Obviously, you also need a good understanding of how Bible OL works from a user's perspective.

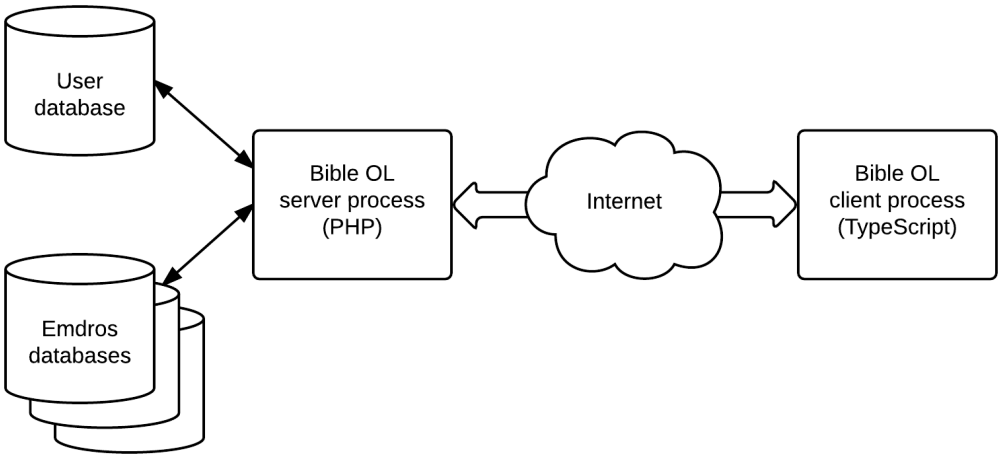
[14] https://getbootstrap.com.
[15] https://www.rgraph.net.

Chapter 5

High-level System Architecture

As a developer, you must read this chapter.

Bible OL consists of two main components, the server and the client, as shown in the following illustration.



The server process runs on a Linux computer, but can probably quite easily be ported to a Windows server that supports PHP, MySQL, and Emdros. It is programmed in PHP and has access to a number of databases:

- The user database which contains information about registered users and translations of Bible OL into various languages.[1]
- A number of Emdros databases, which contain the text and grammatical information for the Old and New Testaments (and potentially other texts as well).

The client is a web browser that accesses the server over an internet connection. It requests the server to provide an exercise or a portion of a text, which it then displays in the browser window. The client code is primarily written in TypeScript, which has been translated to JavaScript so that the browser can execute it.

Most of the layout is done in the client. The server generates HTML code for the main items in the window, but the actual text to be displayed is stored in variables in the client code, and the layout of this information is performed by the client.

[1] The term "user database" is thus somewhat misleading, but the name is retained in this document, nevertheless.

The data exchange between the client and the server is either a two-step or a three-step exchange. If the client merely requests a text to be displayed, the data exchange is:

1. The client requests the server to load a particular URL. The text to display is coded into this URL.
2. The server sends the requested text to the client.

If the client requests an exercise to be displayed, the data exchange is:

1. The client requests the server to load a particular URL. The URL contains information about which quiz template to use and how many questions to generate.
2. The server sends the requested exercise to the client.
3. When the user clicks on the "GRADE task" or "SAVE outcome" button, the client sends the user's answers to the server.

Note that there is no data exchange between the client and the server when the user presses the "Check answer" or "Show answer" buttons. These buttons execute code that is local on the server. (This means that in theory a student can find the correct answers to the questions by looking at the source code for the web page. This is, however, quite difficult to do, and it is not considered a serious flaw in the design since it would require considerable knowledge on the part of the student.)

Chapter 6

Understanding Emdros

As a developer, you must read this chapter. If you need to use MQL, you should read the whole chapter and also some of the documentation that comes with Emdros.

Emdros is a database system for storing and retrieving annotated text. Emdros was developed by Ulrik Sandborg-Petersen. A short introduction to Emdros for linguists is found at http://emdros.org/petersen-emdros-COLING-2004.pdf. More documents are available at http://emdros.org/docs.html.

Here, only a very brief description of the system will be given. Emdros divides a text into *objects.* Typical objects are *word, clause,* and *sentence.* For biblical texts, objects such as *book, chapter,* and *verse* are also used. Each occurrence of the smallest object, typically a word, is identified by a number, called a *monad* in Emdros terminology.

As an example, consider this text: "The boy, who had red hair, was sitting on the floor." This sentence consists of 11 words. We can also identify two clauses, "The boy...was sitting on the floor" and "who had red hair". Emdros assigns a monad (a positive integer) to each word object:

Monad	Word
1	The
2	boy,
3	who
4	had
5	red
6	hair,

Monad	Word
7	was
8	sitting
9	on
10	the
11	floor.

Additionally, sets of monads are used to identify clause objects: Monads { 1-2, 7-11 } identify one clause, and monads { 3-6 } identify another. Finally, the monads { 1-11 } identify a sentence object.

Associated with each object are a number of *features* that describe various characteristics of the object. For a word object, typical features could be *part of speech, gender, number, tense,* and *mood.* In the above sentence, word object 4 (the word "had") could, for example, have these features:

Feature	Value
Text	"had"
Part of speech	Verb
Tense	Past
Number	Singular
Person	3rd
Lexeme	"have"

Similarly, the two clause objects could have a feature called *type* with the values *main* and *subordinate,* respectively.

The exact set of objects and features available in a database is entirely up to the person who creates the database.

In addition to the monads, Emdros objects can also be identified by an ID_D. Like a monad, the ID_D is an integer, but every object in the Emdros database has a unique ID_D. So a single ID_D may refer to a single word or a clause or a sentence. In the above example, the word "who" with monad 3 may have ID_D=8, and the clause "who had red hair" with monads { 3-6 } may have ID_D=12.

Emdros comes with a query language, called MQL[1], that allows a user to search a corpus for objects with various features. MQL queries can be quite simple, such as "find all verbs in the past tense," or very complex, such as "find all sentences containing a singular pronoun, followed by at most three words, followed by a verb in the present tense, except cases where the verb is derived from 'to be'."

The exact syntax of MQL queries can be quite arcane and is beyond the scope of this paper, but examples can be found in http://emdros.org/petersen-emdros-COLING-2004.pdf.

[1] Mini Query Language. (Quite a misnomer since this is a very powerful language.)

Chapter 7

Source Code Tree Overview

As a developer, you must read this chapter.

The source code contains the following directories (in alphabetical order):

Name	Contents
`bootstrap`	A JavaScript framework used on the client side of Bible OL. Webpage: https://getbootstrap.com. Source: https://github.com/twbs/bootstrap.git. License: MIT.
`ckeditor`	An HTML text editor. It is used in the server code to allow the user to edit the description of an exercise. Webpage: http://ckeditor.com. Source: https://github.com/ckeditor/ckeditor-releases. License: A choice between GPL, LGPL, and MPL.
`CodeIgniter_Local`	The CodeIgniter framework used by PHP code on the server side of Bible OL. Webpage: http://www.codeigniter.com. Source: https://github.com/bcit-ci/CodeIgniter. License: MIT.
`culmus-fonts`	A collection of Hebrew fonts. These files are not used directly by the server, but a few of the font files from the subdirectory `Squirrel` have been copied to the `styles/fonts` directory. Webpage: http://culmus.sourceforge.net. Source: http://sourceforge.net/projects/culmus/files/culmus/0.130[1] License: GPL.
`db`	The Emdros databases and associated description files.
`images`	Image files used by the server.
`jquery-ui-1.10.2.custom`	A customized version of *jQuery UI*. Used by the client to display components of the user interface. Webpage: http://jqueryui.com. Source: http://jqueryui.com/download. License: MIT.

(Continued...)

[1] This does not include the `Squirrel` subdirectory. Unfortunately, I don't recall the origin of that directory.

Name	Contents
`js`	JavaScript files from various sources. The files `ol.js`, `editquiz.js`, `handle_legend.js`, and `fontselector.js` are the output form compiling TypeScript files. These JavaScript files should therefore never be edited.
`js/jquery-1.9.1.min.js` `js/jquery.min.map`	These files are part of jQuery. Webpage: http://jquery.com. Source: http://jquery.com/download. License: MIT.
`jstree`	A JavaScript component used by the server to display a hierarchy of books, chapters, and verses of the Bible. Webpage: http://jstree.com. Source: https://github.com/vakata/jstree. License: A choice between MIT and GPL.
`myapp`	The server code. Chapter 17 gives more information. License: MIT, except for the file `myapp/controllers/ctrl_upload.php`, which contains code take from `valums-file-uploader-b3b20b1` mentioned below.
`quizzes`	Quiz templates available for the user. This directory is used only by the runtime system. Development files should not be stored here, and the contents of the directory is not under Git control.
`quiz_templates`	Sample quiz templates to be copied to the `quizzes` directory in a new installation.
`RGraph`	A collection of JavaScript files that aid in drawing graphs. Webpage: https://www.rgraph.net. Source: https://www.rgraph.net/download. License: MIT.
`SILfonts`	A collection of Hebrew, Greek and phonetic fonts. These files are not used directly by the server, but a few of the font files have been copied to the `styles/fonts` directory. Webpage: http://scripts.sil.org. Source: Search the http://scripts.sil.org website for relevant fonts. License: SIL Open Font License.
`styles`	CSS, Less, and fonts files from various sources.
`techdoc`	The technical documentation.
`ts`	TypeScript source files for the client.
`ts/jquery`	The TypeScript definitions of the interfaces and variables found in *jQuery*. Webpage: http://definitelytyped.org. Source: https://github.com/borisyankov/DefinitelyTyped/tree/master/jquery. License: MIT.
`ts/jqueryui`	The TypeScript definitions of the interfaces found in *jQuery UI*. Webpage: http://definitelytyped.org. Source: https://github.com/borisyankov/DefinitelyTyped/tree/master/jqueryui. License: MIT.

(Continued...)

Name	Contents
`valums-file-uploader-b3b20b1`	An old version of a file upload mechanism, used to upload exercise files to the server. This code was released under a GPL license. Since this code was copied to Bible OL, it ownership and licensing has changed. It is now known as *Fine-Uploader* and is available from these sources: Webpage: http://fineuploader.com. Source: https://github.com/FineUploader/fine-uploader License: Widen Commercial License.[2] (I cannot tell this from their license, but according to their website the license allows royalty-free use for non-commercial purposes.)
`virtualkeyboard`	A JavaScript-based virtual keyboard for typing Greek and Hebrew in the client. Website: http://allanguages.info. Source: http://freecode.com/projects/jsvk. License: LGPL.
`zocial`	Icon and styles for setting up a Google or Facebook login button. Website: http://zocial.smcllns.com. Source: https://github.com/samcollins/css-social-buttons. License: MIT.

[2] https://github.com/FineUploader/fine-uploader/blob/master/LICENSE

Chapter 8

Emdros Databases in Bible OL

Read this chapter if you are going to work with code that accesses the Emdros databases on the server or displays text and exercises in the client.

Bible OL currently supports two text databases:

- ETCBC4, which contains the Hebrew and Aramaic version of the Old Testament.
- nestle1904, which contains the Greek version of the New Testament.

These two databases are described in detail in Appendices A and D. This chapter gives only the most important information. One way to learn more about them is to look at the MQL code used for generating these databases. The MQL code for generating the first 1,000 words of an MQL database can be printed by this command:

```
mqldump --batch-create-objects --start 1 --end 1000 database
```

where *database* should be the name of the Emdros database file.

Previous versions of Bible OL have used two other databases: *WIVU* for the Old Testament and *tisch* for the New Testament. The *WIVU* database was protected by a more restrictive copyright than ETCBC4, and the *tisch* database was based on Tischendorf's Greek New Testament, which used peculiar spellings in a number of places. However, traces of these databases can still be found in the system as described in Section 9.2.

In the Bible OL source tree, the Emdros databases are found in the directory `db`.

8.1 The *visual* Feature

The Emdros databases use various feature names to describe the actual text of the corpus. In ETCBC4, the name of the feature is *g_word_utf8* when the Hebrew alphabet is used and *g_word_translit* when a transliterated alphabet is used; in nestle1904, the name of the feature is *surface.*

In order to establish a uniform way to reference this important feature, the Bible OL server and client code uses the name *visual* as an alias for the text feature of the current Emdros database.

8.2 ETCBC4

This section describes a number of features of the ETCBC4 Hebrew/Aramaic database. A more detailed description can be found in Appendix A.

Text in the database comes in three different alphabets:

- Hebrew/Aramaic characters encoded in UTF-8. (In the following text, this will be known as the *native* alphabet.)

- Latin transliteration of the text, encoded in UTF-8. (In the following text, this will be known as the *transliterated* alphabet.)
- Hebrew/Aramaic characters in *ETCBC4 transcription*. This transcription is defined in the document `ETCBC4-transcription.pdf` which is located together with the current document or can be downloaded from `http://shebanq.ancient-data.org/shebanq/static/docs/ETCBC4-transcription.pdf`. (In the following text, this will be known as the *transcribed* alphabet.)

For example, using these three encodings, the three different encodings of the word "created" from Genesis 1 : 1 is encoded as:

- *Native:* בָּרָא
- *Transliterated:* bārāʾ
- *Transcribed:* B.@R@74>

The transcribed characters should never be displayed to users, but they can be useful for internal use because they only use a limited set of ASCII characters.

8.2.1 Object Types

The ETCBC4 database contains these object types:

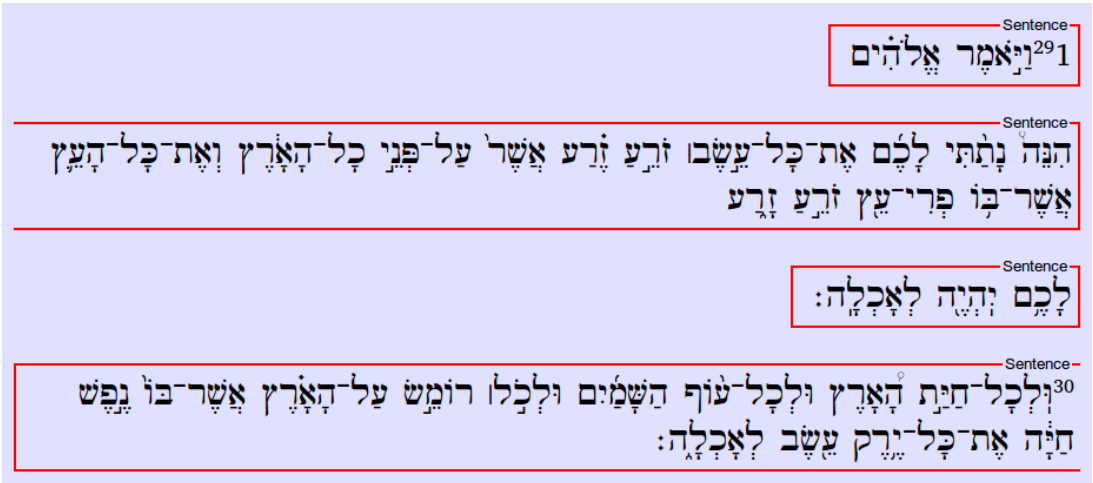
- word
- sentence
- sentence_atom
- clause
- clause_atom
- subphrase
- phrase
- phrase_atom
- book
- chapter
- verse
- half_verse

8.2.1.1 Syntactic Object Types

The object types *word, sentence, sentence_atom, clause, clause_atom, subphrase, phrase,* and *phrase_atom* describe the syntactic composition of the text.

The basic object type is the *word.* Each *word* corresponds to a single monad.

The top-level syntactic element is the *sentence.* A sentence may be built from sets of noncontiguous monads. Each contiguous part of a sentence is a *sentence_atom* object. Consider, for example, Genesis 1 : 29-30:



The sentence starting with the word הִנֵּה consists of two parts. One sentence_atom starts at הִנֵּה and ends at זָרַע, another sentence_atom starts at וּלְכָל־חַיַּת and ends at לְאָכְלָה. Together, these two sentence_atoms make up a single sentence. If a sentence is contiguous, it contains a single sentence_atom.

As illustrated above, Bible OL displays noncontiguous sentences using boxes where one of the sides is missing.

Sentence objects consist of *clause* objects, which – like sentences – are comprised of *clause_atom* objects.

Clause objects consists of *phrase* objects, which are comprised of *phrase_atom* objects.

Phrase objects may contain *subphrase* objects. Note the word "may"; not all words belong to a subphrase. Subphrase objects are always built from contiguous monads. Subphrases may contain other subphrases; for example, in Genesis 1 : 16 the words וַיַּעַשׂ אֱלֹהִים אֶת־שְׁנֵי הַמְּאֹרֹת הַגְּדֹלִים contain three subphrases:

- שְׁנֵי
- הַמְּאֹרֹת הַגְּדֹלִים
- הַגְּדֹלִים

Note how the third subphrase is part of the second subphrase.

8.2.1.2 Editorial Object Types

The object types *book, chapter, verse,* and *half_verse* describe the editorial composition of the text.

The objects *book, chapter,* and *verse* are self-explanatory. The *half_verse* objects identify a subdivsion of verses into two halves, labelled A and B. For example, in Genesis 1 : 1, the two half_verse objects correspond in English to:

A: In the beginning God created
B: the heavens and the earth.

8.2.2 What Is a Word?

In most western languages, a space is inserted between two words. In Hebrew, some words are strung together as one. For example, the text "וַיְהִי־אוֹר׃" ("and there was light") in Genesis 1 : 3 consists of the three words וַ ("and"), יְהִי ("there was"), and אוֹר ("light").

In ETCBC4 this problem is handled by associating an Emdros feature called *suffix* with each word. The suffix feature contains

- a space if a space should be inserted between this word and the next,
- an empty string if this word should be strung together with the following word,
- a ־ (Unicode value 05BE) if a *maqaf* (hyphen) should to be inserted between this word and the next,
- punctuation characters, such as the a ׃ (Unicode value 05C3), which is the verse termination character, *sof pasuq*.

So for the text "וַיְהִי־אוֹר׃", mentioned above, we have these features for the three words:

Text	Suffix
וַ	Empty string
יְהִי	*Maqaf*
אוֹר	*Sof pasuq* followed by space

The actual name of the *suffix* feature varies from one Emdros database to another. (See "suffixFeature" on page 34.)

Note: The *suffix* feature mentioned here must not be confused with the grammatical suffix that can be added to a Hebrew word. For example, in Genesis 1 : 12 the word מִינֵ֣הוּ, which derives from the lexeme מִין, has a grammatical suffix indicating *3rd person, masculine, singular.*

8.3 Nestle1904

This section describes a number of features of the nestle1904 Greek database. A more detailed description can be found in Appendix D.

8.3.1 Object Types

The nestle1904 database contains these object types:

- word
- sentence
- clause1
- clause2
- book
- chapter
- verse

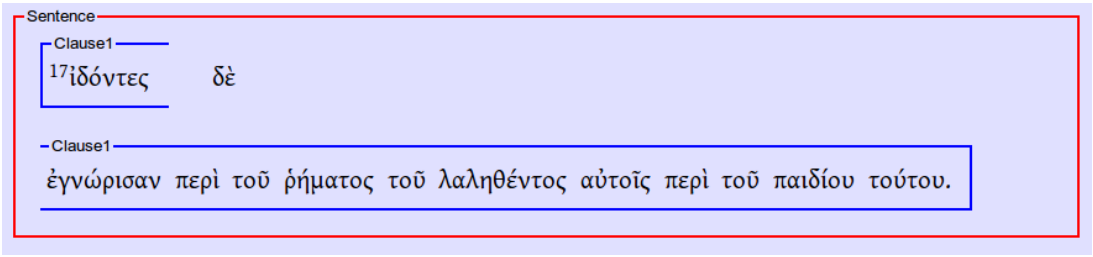
8.3.1.1 Syntactic Object Types

The object types *word, sentence, clause1,* and *clause2* describe the syntactic composition of the text.

The basic object type is the *word.* Each *word* corresponds to a single monad. In contrast to the Hebrew database, the Greek database has no concept of a *suffix* feature (see Section 8.2.2).

The top-level syntactic element is the *sentence.* A sentence is a set of contiguous monads.

Sentence objects contain *clause1* objects. A *clause1* object may be built from sets of noncontiguous monads. Consider, for example, Luke 2 : 17:



This sentence contains a single *clause1* object, but the word *δὲ* is not considered part of that object. As illustrated above, Bible OL displays noncontiguous *clause1* objects using boxes where one of the sides is missing.

Clause1 objects may contain *clause2* objects, which – like *clause1* objects – may be noncontiguous.

8.3.1.2 Editorial Object Types

The object types *book, chapter,* and *verse* describe the editorial composition of the text.

8.4 A Note on Greek Accents in Unicode

Classical Greek used polytonic accents over vowels. For example, the letter alpha could have these polytonic accents:

Character	Greek accent name	English accent name
ά	Oxia (οξεία)	Acute
ὰ	Varia (βαρεία)	Grave
ᾶ	Perispomeni (περισπωμένη)	Circumflex

Modern Greek is not a polytonic language, and in 1982 the polytonic accents were replaced by a single, monotonic accent: the *tonos* (τόνος). In the early years of the monotonic system, particularly when reformers wished to differentiate their system from the polytonic, the tonos on letters was a novel sign: typically a dot or wedge: ά. However, the Greek government decreed in 1986 that the tonos shall be the acute. So you must now write ά instead of ά.[1]

This confusion has had an impact on the representation of the oxia and tonos accents in Unicode. Because of the decision from 1982, Unicode distinguishes between the oxia and the tonos; but because of the decision from 1986, a change was made in Unicode version 3.0 stating that the character ά should be encoded with the Unicode value for GREEK SMALL LETTER ALPHA WITH TONOS, even when writing ancient Greek.

The affected Unicode characters are:

	Unicode value with	
Character	tonos	oxia
ά	03AC	1F71
έ	03AD	1F73
ή	03AE	1F75
ί	03AF	1F77
ό	03CC	1F79
ύ	03CD	1F7B
ώ	03CE	1F7D
ΐ	0390	1FD3
ΰ	03B0	1FE3

So nowadays the correct way to encode the character ά is to use the value 03AC, regardless of whether the accent is an oxia or a tonos.

But here's the catch: *The nestle1904 database uses the oxia encodings, not the recommended tonos encodings.* (The reason is probably a desire to emphasize that polytonic accents are used.)

This necessitates a conversion between tonos encoding and oxia encoding in a few places in Bible OL.

[1] Source: http://www.opoudjis.net/unicode/unicode_gkbkgd.html, accessed 24 April 2018.

Chapter 9

Database Description Files

Read this chapter if you are going to work with code that accesses the Emdros databases on the server or displays text and exercises in the client.

Each database is associated with a number of files that describe details about the database. They are collectively known as the *Database Description Files*. This chapter describes these files in detail.

In the Bible OL source tree, the Database Description Files are found in the directory `db`.

The names of Database Description Files consist of a so-called *primary name* and a suffix. For example, one Database Specification File is `ETCBC4-translit.db.json`. Here the primary name is "`ETCBC4-translit`" and the suffix is "`.db.json`".

The following table lists the suffixes of various Database Description Files:

Suffix	Type	Described in Section
`.db.json`	Database Specification File	9.1
`.typeinfo.json`	Database Type Information File	9.3
`.bookorder`	Database Book Order File	9.4

In the following sections the string "PRIM" is used to denote the primary name of a file.

Most of the files are JSON files. A JSON file contains key/value pairs, where the value can be a string, a number, a Boolean value, an array of values, or another collection of key/value pairs.

A JSON file can either be "ugly" or "pretty". This is an example of an ugly JSON file:

```
{"alpha":8,"beta":{"gamma":true,"delta":["ten","eleven"]}}
```

This is the same data in pretty format:

```
{
    "alpha": 8,
    "beta": {
        "gamma": true,
        "delta": [
            "ten",
            "eleven"
        ]
    }
}
```

Both of these listings describe the same object. The object contains two key/value pairs:

- "alpha" with the numerical value 8.
- "beta" with a value that is a collection of key/value pairs.

The key "beta" has a value that contains two key/value pairs:

- "gamma" with the Boolean value *true*.
- "delta" with a value that is an array containing the two strings "ten" and "eleven".

Bible OL works equally well with ugly and pretty JSON files, but the ugly format is normally preferred because it takes up less space (and makes reverse engineering slightly more difficult for the uninitiated). The pretty format is, of course, easier for humans to understand is therefore useful while debugging the system.

The script *json_pretty_print.php* can be used to convert between the ugly and the pretty format. If the file "xxx.json" contains JSON data (either ugly or pretty), the command

```
php json_pretty_print.php -p xxx.json
```

will write the data in pretty format to standard output; and the command

```
php json_pretty_print.php -u xxx.json
```

will write the data in ugly format to standard output.

The directory `db` contains JSON files in both the ugly and the pretty format. For example, the file `ETCBC4.db.json` is the ugly version of `ETCBC4.db.pretty.json`. The developer should only modify the `.pretty.json` files and then run the *Makefile* in the top source directory, which will generate the corresponding ugly JSON files.

9.1 Database Specification File: PRIM.db.json

On the server the *Database Specification File* has a name that ends in `.db.json`. For Bible OL, this is the main point of access to the Emdros databases. When Bible OL needs to list the available databases, it searches for files with names that end in `.db.json`.

On the client the contents of the Database Specification File is available in a variable called *configuration*. Its structure is described in TypeScript as the *Configuration* interface in the file `ts/configuration.ts`.

The Database Specification File describes how the individual parts of an Emdros database are used by Bible OL. It describes what features are available for exercises and what grammatical features the user can choose to display.

Multiple Database Specification Files can refer to the same Emdros database. For example, `ETCBC4.db.json` and `ETCBC4-translit.db.json` both reference the ETCBC4 Emdros database, but the former displays text using the native alphabet, whereas the latter displays text using the transliterated alphabet.

The Database Specification File is a JSON file containing the following key/value pairs:

Key	Value
version	A number which identifies the layout used by this file. This number is currently ignored.
databaseName	The name of the Emdros database file. Currently, this is either "ETCBC4" or "nestle1904". This is also the primary name of the Database Type Information File (see Section 9.3) and Database Book Order File (see Section 9.4).
propertiesName	The name of the Grammar Localization Struture (see Section 21.4).

(Continued...)

Key	Value
databaseVersion	A string containing the version number of the database. This number is only used for display purposes. Whenever an Emdros database is changed, this number should also be changed.
granularity	The name of the Emdros object type defining the amount of text to display in an exercise. Typically, this name is "sentence".
surfaceFeature	The actual name of the *visual* feature (see Section 8.1). For ETCBC4 using the native alphabet this value is "g_word_utf8", for ETCBC4 using the transliterated alphabet the value is "g_word_translit", for nestle1904 the value is "surface".
objHasSurface	The name of the Emdros object type that contains the *surfaceFeature*. (Typically, "word".)
suffixFeature	The actual name of the *suffix* feature (see Section 8.2.2). For ETCBC4 using the native alphabet this value is "g_suffix_utf8", for ETCBC4 using the transliterated alphabet the value is "g_suffix_translit", for nestle1904 the value is *null*.
charSet	The name of the character set for the text. For ETCBC4 using the native alphabet this value is "hebrew", for ETCBC4 using the transliterated alphabet the value is "transliterated_hebrew", for nestle1904 the value is "greek".
objectSettings	A collection of key/value pairs containing information about how Bible OL should treat Emdros object type. This is detailed in Section 9.1.1.

universeHierarchy	An array containing information about how the text references are structured. For the Bible, this hierarchy is book/chapter/verse. A typical value is
	`"universeHierarchy": [`
	`{`
	`"type": "book",`
	`"feat": "book"`
	`},`
	`{`
	`"type": "chapter",`
	`"feat": "chapter"`
	`},`
	`{`
	`"type": "verse",`
	`"feat": "verse"`
	`}`
	`]`
	This means that the top reference level is found in the *book* feature of Emdros objects of type *book*, the second reference level is found in the *chapter* feature of Emdros objects of type *chapter*, and the third reference level is found in the *verse* feature of Emdros objects of type *verse*.
	Note that in several locations, code in Bible OL is hard-coded to rely on the book/chapter/verse structure used in the Bible.

picDb	The URL of the resource website (see Section 23.1). This value may be *null*.
sentencegrammar	An array containing information about the grammar items available to the user. This is detailed in Section 9.1.3.

(Continued...)

Key	Value
subsetOf	If the Database Specification File describes a subset of a larger database, *subsetOf* gives information about the subset. This is detailed in Section 9.1.7. This value is always *null* for the ETCBC4 and nestle1904 databases.

9.1.1 The *objectSettings* Key

The *objectSettings* key in the Database Specification File gives detailed information about how the Emdros object types and their features should be treated by Bible OL. The *objectSettings* key has a value that is a collection of key/value pairs, where the keys are Emdros object type. The corresponding values give information about how the Emdros object should be treated.

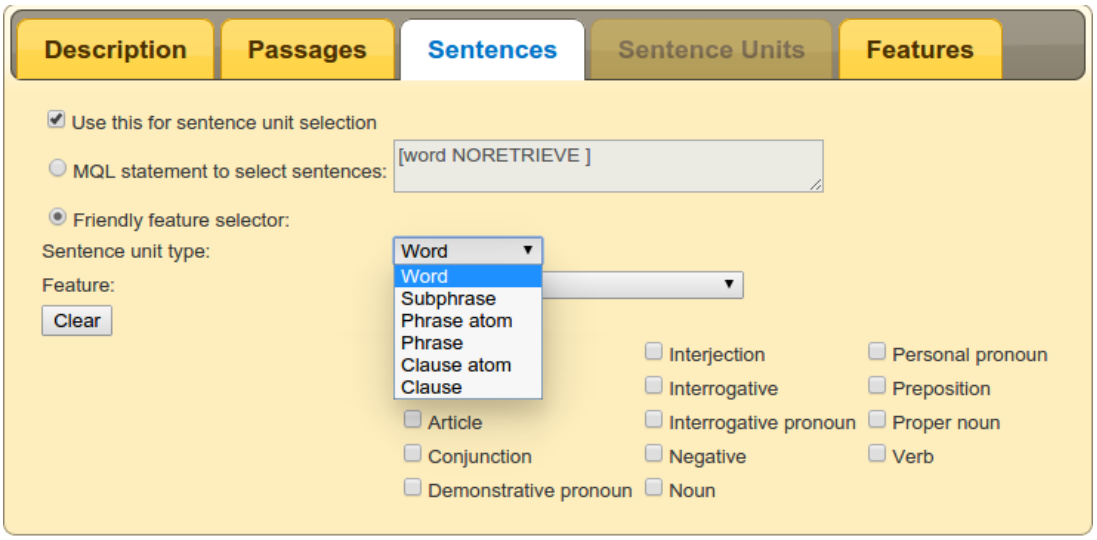
Listing 9.1 shows a subset of the *objectSettings* for the ETCBC4 database.

Listing 9.1: A sample objectSettings value

```
"objectSettings": {
    "book": {
    },
    "chapter": {
    },
    "verse": {
    },
    "word": {
        "mayselect": true,
        "additionalfeatures": [...],
        "featuresetting": {...}
    },
    "subphrase": {
        "mayselect": true,
        "featuresetting": {...}
    }
}
```

In this subset, the Emdros types *book, chapter, verse, word,* and *subphrase* are mentioned. No special information about *book, chapter,* and *verse* is provided, which means that these Emdros types cannot be made the subject of exercises. It is the presence of the key *mayselect* with the value *true* that signals to Bible OL that the associated Emdros object can be used when selecting quiz objects for an exercise. So in the example in Listing 9.1, the Emdros objects *word* and *subphrase* can be used for quiz object selection.

When you are creating an exercise with the ETCBC4 database, the “Sentences” and “Sentence Units” tabs allow you to specify a sentence unit (a.k.a. quiz object) type:



The values in the drop down box are the Emdros object that have *mayselect* set to *true* in *objectSettings*. As shown in Listing 9.1, these objects have one or two additional key/value pairs with the keys *featuresetting* and, optionally, *additionalfeatures*.

The value of *featuresetting* is another collection of key/value pairs, giving details about the features of the object. This is detailed in Section 9.1.2.

The *additionalfeatures* key identifies an array of features that should always be retrieved for this Emdros object, even though these features are not the subject of an exercise. They are used for accessing the multiple choice database (see Chapter 11).

9.1.2 The *featuresetting* Key

The *featuresetting* key under an Emdros object type gives detailed information about how the features of the Emdros object should be treated by Bible OL. Its value is a collection of key/value pairs, where the keys are feature names. The corresponding values give information about how the feature should be treated.

Listing 9.2 shows a subset of the *featuresetting* for the *word* object in the ETCBC4 database.

Listing 9.2: A sample featuresetting value

```
"featuresetting": {
    "lexeme_occurrences": {
        "ignoreShow": true,
        "ignoreRequest": true,
        "isRange": true
    },
    "english": {
        "ignoreSelect": true,
        "matchregexp": "\/^(.+[;,] +)?(\\(.*\\) *)?{0}( *\\(.*\\))?([,;].+)?$\/i",
    },
    "g_word_cons_utf8": {
        "hideWord": true,
        "foreignText": true
    },
    "g_word_nocant_utf8": {
        "alternateshowrequestDb": "ETCBC4_words.db",
        "alternateshowrequestSql": "SELECT DISTINCT word FROM texts,lextext,lexemes WHERE
lex='%s' AND lexid=lexemes.id AND textid=texts.id",
        "hideWord": true,
        "foreignText": true
    },
    "vt": {
```

```
            "hideValues": [
                "weyq"
            ]
        }
    }
```

In this subset, the features *lexeme_occurrences, english, g_word_const_utf8, g_word_nocant_utf8,* and *vt* are mentioned. The value associated with each of these keys is another collection of key/value pairs. Many of the values are Boolean, and an absent key is equivalent to a Boolean value of *false*. Thus, the absence of a *hideWord* key from *lexeme_occurrences* has the same meaning as if *hideWord* had been given the value *false*.

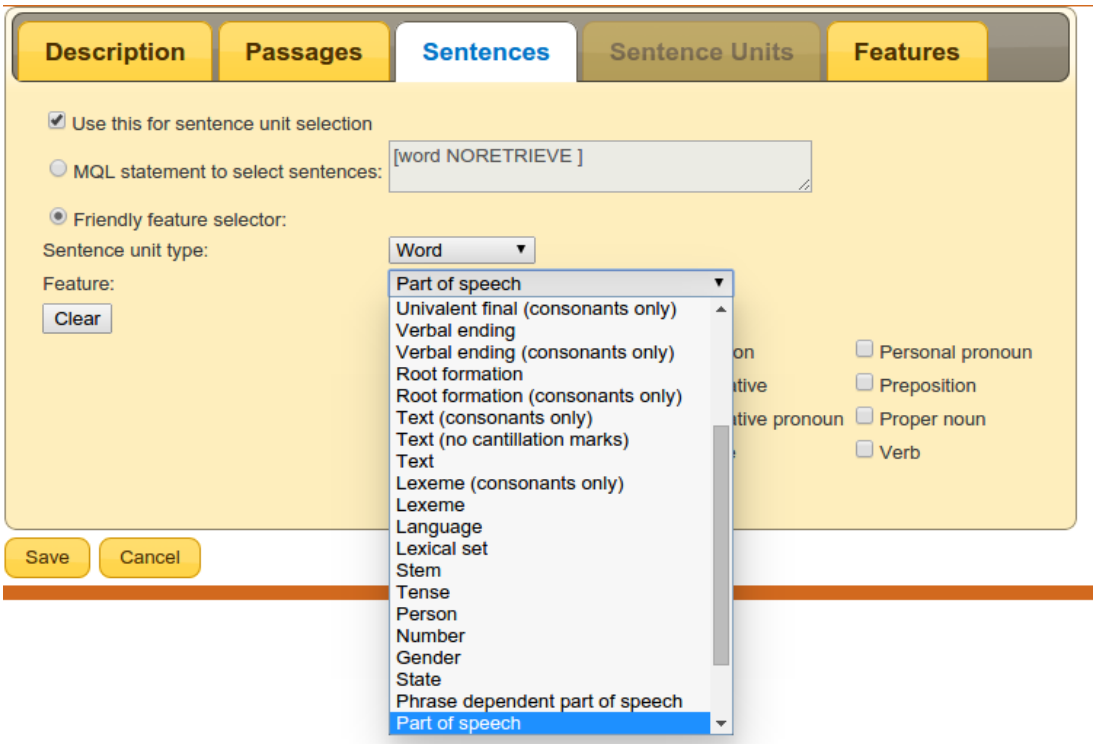
The following table lists the keys and values that can be associated with a feature of an Emdros object:

Key	Value
ignoreSelect	A value of *true* means: Do not use this feature for object selection (see below).
isDefault	This value must be *true* for exactly one feature of an Emdros object. It indicates that this feature is the initially displayed feature in the "Sentences" tab when creating an exercise (see below).
ignoreShow	A value of *true* means that this feature cannot be displayed as part of an exercise. In other words, there is no "Show" button for this feature on the "Features" tab when creating an exercise.
ignoreRequest	A value of *true* means that this feature cannot be requested as part of an exercise. In other words, there is no "Request" button for this feature on the "Features" tab when creating an exercise.
hideWord	If this value is *true* and the feature is a request feature for an exercise, the corresponding words should be replaced by a number in the displayed text (see below).
foreignText	A value of *true* means that this feature is written using a non-Latin alphabet.
transliteratedText	A value of *true* means that this feature is written using the transliterated Hebrew alphabet.
hideValues	Relevant only for enumeration features. It is an array of enumeration values that never occur in a text and should therefore be omitted from the user interface.
isRange	A value of *true* means that this feature represents range of integer values.
otherValues	An array of enumeration feature values that should be lumped together as "Other" in the user interface. (This is not currently used by any Emdros databases in Bible OL.)
matchregexp	A regular expression used to check if an answer provided by a learner matches the value of a feature. For example, the *english* feature for the Hebrew word אֶרֶץ has the value "land; territory, country; the earth". The regular expression in *matchregexp* is designed to ensure that a learner's answer is accepted, regardless of whether the answer is "land", "territory", "country", or "the earth".
alternateshowrequestDb	The name of a multiple choice database (see Chapter 11).

(Continued...)

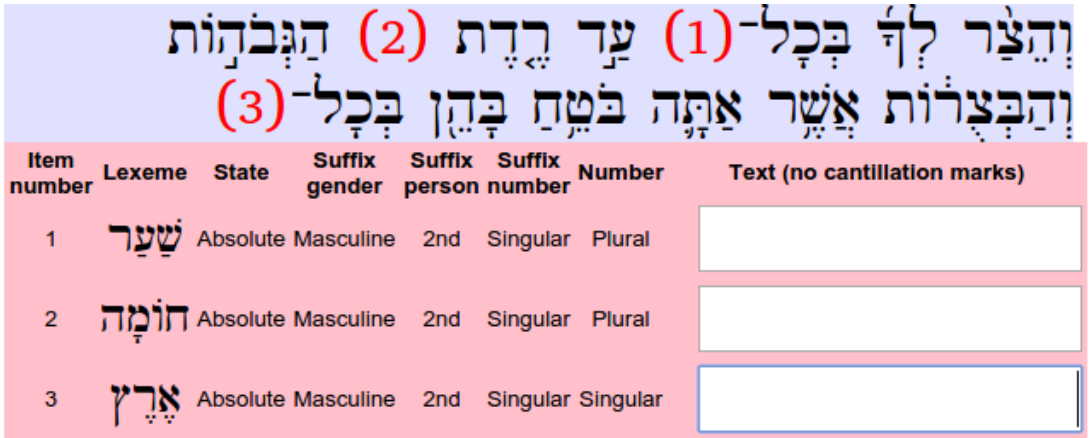
Key	Value
alternateshowrequestSql	An SQL statement used to access the multiple choice database (see Chapter 11).
indirdb, sql_command, sqlargs, multiple	These keys are used for pseudofeatures. See Section 9.1.2.1.

The keys *ignoreSelect* and *isDefault* control the contents of the feature selection menu when creating an exercise:



If *ignoreSelect* is set for a feature, then that feature is not shown in the selection menu. The feature with *isDefault* set is the selected feature when the dialog is first loaded.

If *hideWord* is *true* and the feature is used as a request feature in an exercise, the corresponding word is replaced by a number in the text. For example, in the following exercise the feature *text_nocant_utf8* (that is, "Text (no cantillation marks)") is used as a request feature. Consequently the corresponding words in the text are replaced by the numbers (1), (2), and (3) lest the words in the text give the answer to the questions:



9.1.2.1 Pseudofeatures

The features listed in the *featuresetting* information are normally genuine feature of Emdros objects. However, a few of them may be *pseudofeatures*. A pseudofeature is logically associated with an Emdros object just as ordinary features are, but the values of pseudofeatures are retrieved from other data sources.

As an example of a pseudofeature, consider Listing 9.3.

LISTING 9.3: Featuresetting syntax for a pseudofeature

```
"english": {
    "ignoreSelect": true,
    "matchregexp": "\/^(.+[;,] +)?(\\(.*\\) *)?{0}( *\\(.*\\))?([,;].+)?$\/i",
    "indirdb": "mysql",
    "sql_command": "SELECT gloss FROM {PRE}lexicon_%1$s h JOIN {PRE}lexicon_%1$s_en en ON
en.lex_id=h.id WHERE lex='%2$s' AND vs='%3$s'",
    "sqlargs": [ "language", "lex", "vs" ],
    "multiple": false
}
```

A pseudofeature uses the keys *indirdb, sql_command, sqlargs,* and *multiple.*

- *indirdb* gives the name of a database in which the pseudofeature can be found. The value is either the name of an SQLite3 database, or the string "mysql" if the pseudofeature is found in the user database (see Section 18).
- *sql_command* is the SQL command required to retrieve the value of the feature. A typical example is given in the listing above. The system will replace the string `{PRE}` with the correct database table prefix. The strings `%1$s`, `%2$s`, etc. indicate the position of the parameters required for the request. These parameters are specified under the key *sqlargs*. The string `%1$s` refers to the first parameter in *sqlargs*, `%2$s` refers to the second parameter in *sqlargs*, etc.
- *sqlargs* is an array of features whose values are to be used in place of the `%...$s` strings in *sql_command.*
- *multiple* is true if the database may contain more than one record for a given pseudofeature.

Taking the pseudofeature *english* from Listing 9.3 as an example, we see that the feature is found in the user database because *indirdb* has the value "mysql".

If we assume that the database table prefix[1] is "bol_", and if the *language, lex,* and *vs* features have the values "Hebrew", "B.@R@>", and "qal", respectively, the value of the pseudofeature is found by executing the SQL query

```
SELECT gloss FROM bol_lexicon_Hebrew h JOIN bol_lexicon_Hebrew_en en ON en.lex_id=h.id
    WHERE lex='B.@R@>' AND vs='qal'
```

The SQL query is expected to return a single value because *multiple* is *false.*

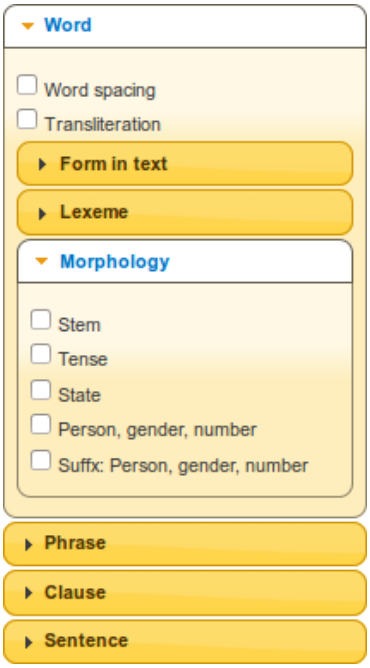
9.1.3 The *sentencegrammar* Key

The *sentencegrammar* key in the Database Specification File gives detailed information about the grammar items available to the user. Its value is an array, in which each entry corresponds to an Emdros object type.

Bible OL uses the information in *sentencegrammar* in two locations. One is the grammar selection box in the left part of the screen, the other is in the grammar information box in the right part of the screen.

The grammar selection box may look like this:

[1]The database table prefix can be set in the file `myapp/config/database.php` (See Section 3.1.4).



In the above illustration, each of the four main menus, *Word, Phrase, Clause,* and *Sentence,* of the grammar selection box corresponds to an entry at the top level of *sentencegrammar.*

The grammar information box may look like this:

Word	
Text	שָׁמַיִם
Transliteration	ššāmayim
Form in text:	
Root formation	-
Preformative	-
Verbal ending	-
Nominal ending	ם֑
Pronominal suffix	-
Univalent final	-
Qere	-
Lexeme:	
Lexeme	שָׁמַיִם
Lexeme (transliterated)	šāmayim
English	heaven
German	Himmel
Occurrences	421
Frequency rank	123
Part of speech	Noun
Phrase dependent part of speech	Noun
Lexical set	None
Verb class	N/A
Morphology:	
Stem	None
Tense	None
State	Absolute
Person, gender, number	-MPl
Suffx: Person, gender, number	---

In this illustration, the *sentencegrammar* for the *word* object has been used to determine what information to retrieve.

Listing 9.4 shows the *sentencegrammar* for the ETCBC4 database (taken from the file `db/ETCBC4.db.json`).

Listing 9.4: Condensed sentencegrammar value

```
"sentencegrammar": [
    {
        "mytype": "SentenceGrammar",
        "objType": "word",
        "items": [...]
    },
    {
        "mytype": "SentenceGrammar",
        "objType": "phrase",
        "items": [...]
    },
    {
        "mytype": "SentenceGrammar",
        "objType": "clause",
        "items": [...]
    },
```

```
    {
        "mytype": "SentenceGrammar",
        "objType": "sentence"
    }
]
```

As this example shows, *sentencegrammar* is an array whose elements have three key/value pairs:

- *mytype* with a value that is always "SentenceGrammar".
- *objType* with a value that identifies an Emdros object.
- *items* (optional), which is an array of *GrammarFeature, GrammarMetaFeature,* or *GrammarGroup* specifications, as detailed below.

If you compare the listing of *sentencegrammar* above, with the grammar selection box shown on page 40, you will notice that each element in the *sentencegrammar* corresponds to a top level box in the illustration.

For each Emdros object (that is, for each element in the *sentencegrammar* array) the value of *items* is an array that specified the features that can be displayed for the particular Emdros object. Each entry in the *items* array can have one of three forms: *GrammarFeature, GrammarMetaFeature,* or *GrammarGroup*, as specified in their *mytype* value.

Listing 9.5 shows a typical *items* array.

Listing 9.5: A sample items value

```
"items": [
    {
        "mytype": "GrammarFeature",
        "name": "g_word_translit"
    },
    {
        "mytype": "GrammarGroup",
        "name": "form_in_text",
        "items": [...]
    },
    {
        "mytype": "GrammarMetaFeature",
        "name": "pgn",
        "items": [...]
    }
]
```

In this example, the *items* array has two elements, one with *mytype=GrammarFeature*, one with *mytype=GrammarGroup*, and one with *mytype=GrammarMetaFeature.*

9.1.4 GrammarFeature

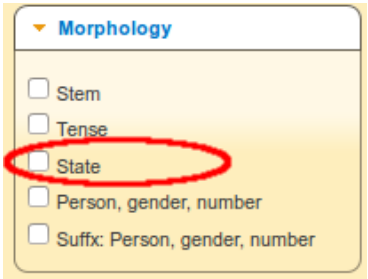
A *sentencegrammar* item with *mytype=GrammarFeature* describes a single Emdros feature. The format is given in Listing 9.6.

Listing 9.6: GrammarFeature syntax

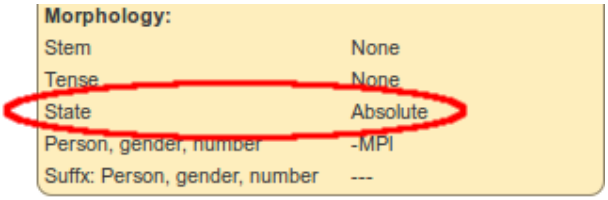
```
{
    "mytype": "GrammarFeature",
    "name": an Emdros feature name
}
```

The *name* value may simply be the name of a feature of the current Emdros object type, or it may be a string in the form "objectType:featureName", if the feature belongs to another object type.

In the grammar selection box, a GrammarFeature is displayed thus:



In the grammar information box, a GrammarFeature is displayed thus:



9.1.5 GrammarMetaFeature

A *sentencegrammar* item with *mytype=GrammarMetaFeature* describes a combined value of a number of Emdros features. Its format is given in Listing 9.7.

Listing 9.7: GrammarMetaFeature syntax

```
{
    "mytype": "GrammarMetaFeature",
    "name": the name of the GrammarMetaFeature,
    "items": [
        {
            "mytype": "GrammarSubFeature",
            "name": an Emdros feature name
        },
        {
            "mytype": "GrammarSubFeature",
        },
        ... Additional GrammarSubFeatures
    ]
}
```

In the *items* array the Emdros features that make up the GrammarMetaFeature are listed with a *mytype* value of "GrammarSubFeature".

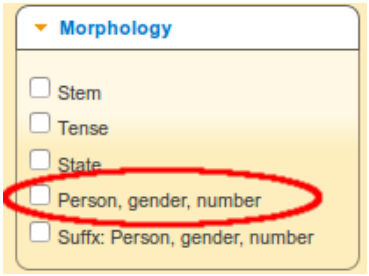
As an example, Bible OL combines the person, gender, and number of a word to a single item, such as "2FSg" (which means 2nd person, feminine, singular). This is specified as indicated in Listing 9.8.

Listing 9.8: A GrammarMetaFeature combining person, gender, and number

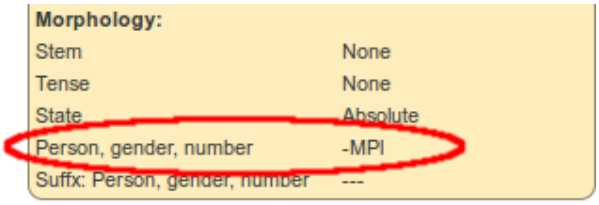
```
{
    "mytype": "GrammarMetaFeature",
    "name": "pgn",
    "items": [
        {
            "mytype": "GrammarSubFeature",
            "name": "ps"
        },
        {
```

```
            "mytype": "GrammarSubFeature",
            "name": "gn"
        },
        {
            "mytype": "GrammarSubFeature",
            "name": "nu"
        }
    ]
},
```

The *ps, gn,* and *nu* features represent the person, gender, and number of a word, respectively. In the grammar selection box, a GrammarMetaFeature is displayed thus:



In the grammar information box, a GrammarMetaFeature is displayed thus:



9.1.6 GrammarGroup

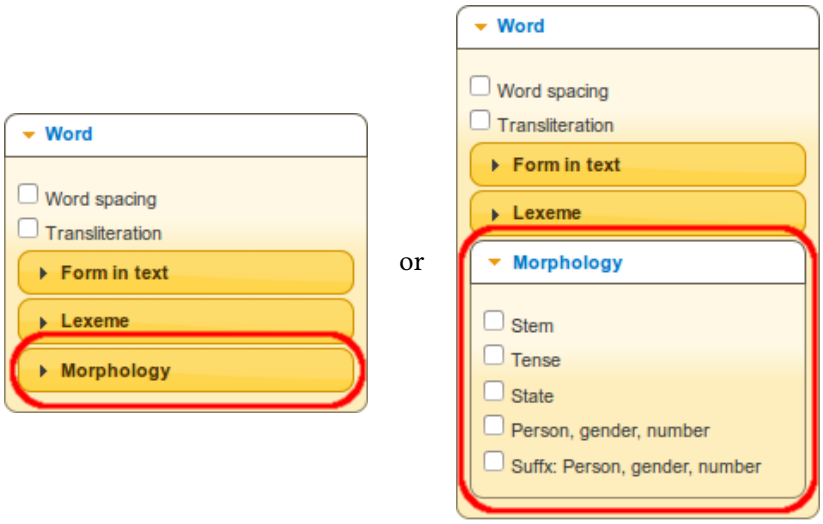
A *sentencegrammar* item with *mytype=GrammarGroup* groups GrammarFeatures and GrammarMetaFeatures into logical units, such as "Features that describe the lexeme" or "Features that describe the morphology". Its format is given in Listing 9.9.

Listing 9.9: GrammarGroup syntax

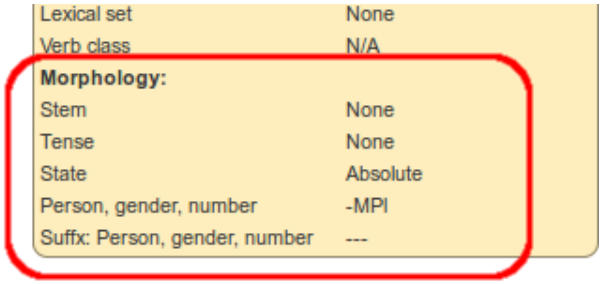
```
{
    "mytype": "GrammarGroup",
    "name": the name of the GrammarGroup,
    "items": [...]
}
```

The *items* array here contains a collection of GrammarFeatures and GrammarMetaFeatures in the format described above.

In the grammar selection box, a GrammarGroup (called “Morphology” in this case) is displayed thus:



In the grammar information box, a GrammarGroup is displayed thus:



9.1.7 The *subsetOf* Key

The *subsetOf* value is present for historical reasons. With the current ETCBC4 and nestle1904 databases it is not required; however, the previously used WIVU database, was available in two versions: the entire Old Testament and a subset containing about 20 per cent of the text.

The *subsetOf* key specifies the relationship between the subset and the superset. It is used to enable exercises that have been created for the subset to be used with the superset.

The format of *subsetOf* is illustrated by this example taken from the subset Database Specification File:

```
"subsetOf": {
    "name": "WIVU",
    "properties": "WIVU",
    "provides": [
        "Genesis",
        "Exodus:1",
        "Exodus:2",
        "Exodus:3",
        ... (Additional Bible references omitted)
    ]
}
```

Here, *name* is the name of the Emdros database file for the superset, and *properties* is the name of the Grammar Localization Struture (see Section 21.4) for the superset.

The *provides* array identifies the books and chapters provided by the subset. In this example, the subset includes the entire book of Genesis and the first three chapters of Exodus.

If *subsetOf* has the value *null*, the current database is not a subset of any other database.

9.2 Database Description Files for Old Databases

Currently, Bible OL uses two databases, the Hebrew *ETCBC4* database and the Greek *nestle1904* database. Previous versions of Bible OL used the Hebrew *WIVU* database (and various variants of it) and the Greek *tisch*[2] database.

The statistics tables in the user database (see Section 18.12) may still contain data from these older databases. If description and localization information for the old databases are not available, the corresponding statistics will not be displayed.

9.3 Database Type Information File: PRIM.typeinfo.json

The Emdros databases in Bible OL have an associated Database Type Information File. This file contains information about the Emdros object types and enumeration types. **On the server**, if the name of the Emdros database is PRIM, the name of the associated Database Type Information File is `PRIM.typeinfo.json`.

On the client the contents of the Database Type Information File is available in a variable called *typeinfo*. Its structure is described in TypeScript as the *TypeInfo* interface in the file `ts/configuration.ts`.

The Database Type Information file is a JSON file. Its contents can be automatically generated from the Emdros database itself. The PHP script `myapp/controllers/ctrl_maketypeinfo.php` contains code to do that. Running the command

```
php index.php maketypeinfo index databasename
```

from the base of the source code tree will generate the type information form the specified Emdros database and write it to standard out in ugly JSON format.

Note: The Database Type Information File must also contain information about pseudofeatures (see Section 9.1.2.1). This information is not generated automatically by the above command, but must be added manually.

The Database Type Information File contains the following key/value pairs:

Key	Value
objTypes	An array containing the names of all Emdros object types.
obj2feat	A collection of key/value pairs that list the names and types of the features associated with each Emdros object type (see Section 9.3.1).
enymTypes	An array containing the names of all enumeration types in the database.
enum2values	A collection of key/value pairs that list the values of all enumeration types (see Section 9.3.2).

9.3.1 The *obj2feat* Key

The value of the *obj2feat* key is a collection of key/value pairs that list the names and types of the features associated with each Emdros object type.

[2] Short for *Tischendorf*.

Listing 9.10 shows a subset of *obj2feat* for the ETCBC4 database (taken from the file `db/ETCBC4.typeinfo.json`).

Listing 9.10: A sample obj2feat value

```
1   "obj2feat": {
2       "word": {
3           "frequency_rank": "integer",
4           "g_suffix_utf8": "string",
5           "nu": "number_t",
6           "gn": "gender_t",
7           "sp": "part_of_speech_t",
8           "verb_class": "list of verb_class_t"
9           ... (Additional features omitted)
10      },
11      "clause_atom": {
12          "code": "integer",
13          "dist": "integer",
14          "is_root": "boolean_t",
15          "typ": "clause_atom_type_t"
16          ... (Additional features omitted)
17      },
18      ... (Additional object types omitted)
19  }
```

For example, line 3 shows that the *word* object has a feature called *frequency_rank* of type *integer*.

9.3.2 The *enum2values* Key

The value of the *enum2values* key is a collection of key/value pairs that list the values of all enumeration types.

Listing 9.11 shows a subset of *enum2values* for the ETCBC4 database (taken from the file `db/ETCBC4.typeinfo.json`).

Listing 9.11: A sample enum2values value

```
1   "enum2values": {
2       "boolean_t": [
3           "false",
4           "true"
5       ],
6       "number_t": [
7           "NA",
8           "du",
9           "pl",
10          "sg",
11          "unknown"
12      ],
13      ... (Additional enumeration types omitted)
14  }
```

For example, lines 2-5 show that the enumeration type *boolean_t* has the values *false* and *true*.

9.4 Database Book Order File: PRIM.bookorder

The Emdros databases in Bible OL have an associated Database Book Order File. If the name of the Emdros database is PRIM, the name of the associated Database Book Order File is `PRIM.bookorder`. This information is only available **on the server**.

This is a text file that lists the names of the books in the database in the order in which they should be presented. I also lists the chapters available in each book.

Listing 9.12 shows a subset of the Database Book Order File for the ETCBC4 database (taken from the file `db/ETCBC4.bookorder`).

Listing 9.12: A subset of the ETCBC4 Book Order File

```
Genesis/1-50
Exodus/1-40
Leviticus/1-27
Numeri/1-36
Deuteronomium/1-34
Josua/1-24
Judices/1-21
Samuel_I/1-31
Samuel_II/1-24
... (Additional books omitted)
```

This file defines the Hebrew order of the books of the Old Testament.[3] Each line consists of the name of the book (as it is defined in the database), followed by a slash and the list of available chapters.

For the ETCBC4 and nestle1904 databases, the chapters are always given as a simple range, such as "1-50" for Genesis, but as mentioned in Section 9.1.7, it is possible to define a subset of a database. In that case a line of the Database Book Order File may look like this:

```
Leviticus/2,6-9,23
```

which indicates that only chapters 2, 6-9, and 23 of Leviticus are available.

[3] This differs from the order of books used in Christian Bibles.

Chapter 10

Quiz Templates

Read this chapter if you need to understand how exercises are stored in the server.

A quiz template (or an exercise template) is an XML file that describes how Bible OL should generate an exercise. It always has a filename that ends in `.3et`.

Listing 10.1 shows a typical quiz template.

Listing 10.1: Quiz template sample

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <questiontemplate version="3">
3    <desc><![CDATA[Which gender is this?]]></desc>
4    <database>ETCBC4</database>
5    <properties>ETCBC4</properties>
6    <path>Genesis:1:1</path>
7    <path>Genesis:1:2</path>
8    <path>Genesis:3</path>
9    <path>Exodus</path>
10   <sentenceselection version="1">
11     <questionobject>word</questionobject>
12     <featurehandlers version="2">
13       <enumfeature version="1">
14         <name>sp</name>
15         <comparator>equals</comparator>
16         <value>subs</value>
17         <value>prps</value>
18       </enumfeature>
19       <enumfeature version="1">
20         <name>gn</name>
21         <comparator>differs</comparator>
22         <value>NA</value>
23         <value>unknown</value>
24       </enumfeature>
25     </featurehandlers>
26     <useforquizobjects>true</useforquizobjects>
27   </sentenceselection>
28   <quizfeatures version="3">
29     <show>visual</show>
30     <request>gn</request>
31   </quizfeatures>
32 </questiontemplate>
```

The *version* attribute used in several elements identifies what elements may legally appear within other elements. For example, the current version of Bible OL allows the elements `<show>`, `<request>`, `<requestdd>`, and `<dontshow>` within the `<quizfeatures>` element (see line 28 above). If a future

version of Bible OL changes the legal content of `<quizfeatures>`, the `version="3"` string should be changed to `version="4"`.

The top level of the quiz template is the `<questiontemplate>` element. It contains these elements:

Element	Contents
`<desc>`	A *CDATA* string describing the exercise. This string may contain HTML code. This is the text entered under the "Description" tab when creating a quiz template.
`<database>`	The name of the Emdros database and the primary name of the Database Specification File.
`<properties>`	The name of the Grammar Localization Struture.
`<path>`	This element may occur several times in the file. It describes a component of the passages used for the exercise. This is the data entered under the "Passages" tab when creating a quiz template. In Listing 10.1, lines 6-9 specify Genesis chapter 1 verses 1-2, Genesis chapter 3 (all verses), and the book of Exodus (all chapters).
`<sentenceselection>`	A description of how Bible OL should select sentences. See Section 10.1.
`<quizobjectselection>`	A description of how Bible OL should select sentence units (a.k.a. quiz objects). See Section 10.2.
`<quizfeatures>`	The display features and request features. See Section 10.3.
`<maylocate>`	A Boolean value indicating if a Bible reference may be displayed when showing an exercise question. If this XML element is absent, it is assumed have the value *true*.

10.1 `<sentenceselection>`

The `<sentenceselection>` element contains the information entered under the "Sentences" tab when creating a quiz template.

The `<sentenceselection>` element contains these elements:

Element	Contents
`<questionobject>`	The Emdros object type that is used for sentence selection. This value of this element is irrelevant if an MQL string is to be used for sentence selection.
`<mql>`	This element is only present if an MQL string is to be used for sentence selection. The element contains the MQL string. (See Section 10.4.)
`<featurehandlers>`	This element is not present if an MQL string is to be used for sentence selection. It contains a description of the features used for sentence selection. See Section 10.1.1.
`<useforquizobjects>`	A Boolean value which is *true* if the contents of the `<sentenceselection>` element is also used for sentence unit selection, and is *false* is sentence unit select is specified separately. This is controlled by the check box "Use this for sentence unit selection" under the "Sentences" tab.

10.1.1 `<featurehandlers>`

The `<featurehandlers>` element contains descriptions of the Emdros features used for selecting a sentence or a sentence unit. The `<featurehandlers>` element contains one or more of these elements,

each of which describes an Emdros feature and how it is used for selections:

Element	Emdros feature type
`<stringfeature>`	String. See Section 10.1.1.1.
`<integerfeature>`	Integer. The selector specifies distinct integer values. See Section 10.1.1.2.
`<rangeintegerfeature>`	Integer. The selector specifies a range of values for the Emdros feature. See Section 10.1.1.3.
`<enumfeature>`	Enumeration. See Section 10.1.1.4.
`<enumlistfeature>`	List of enumeration values. See Section 10.1.1.5.
`<qerefeature>`	A string representing a Hebrew qere form. See Section 10.1.1.6.

An implicit logical *AND* is assumed between these selector specifiers, meaning that only objects that match all of the specified selectors are chosen.

10.1.1.1 `<stringfeature>`

The `<stringfeature>` element contains a description of how an Emdros feature of type string is used for selecting a sentence or a sentence unit.

The `<stringfeature>` element contains these elements:

Element	Contents
`<name>`	The name of the Emdros feature.
`<comparator>`	The string "equals", "differs", or "matches". If the string is "equals", the Emdros feature must be equal to one of the `<value>` elements mentioned below; if the string is "differs", the Emdros feature must not be equal to any of the `<value>` elements mentioned below; if the string is "matches", the Emdros feature must match one of the regular expressions given in the `<value>` elements below.
`<value>`	This element may occur several times. It contains a string that is compared to the value of the Emdros feature.

10.1.1.2 `<integerfeature>`

The `<integerfeature>` element contains a description of how an Emdros feature of type integer is used for selecting a sentence or a sentence unit.

The `<integerfeature>` element contains these elements:

Element	Contents
`<name>`	The name of the Emdros feature.
`<comparator>`	The string "equals" or "differs". If the string is "equals", the Emdros feature must be equal to one of the `<value>` elements mentioned below; if the string is "differs", the Emdros feature must not be equal to any of the `<value>` elements mentioned below.
`<value>`	This element may occur several times. It contains an integer that is compared to the value of the Emdros feature.

10.1.1.3 `<rangeintegerfeature>`

The `<rangeintegerfeature>` element contains a description of how an Emdros feature of type integer is used for selecting a sentence or a sentence unit.

The `<integerfeature>` element contains these elements:

Element	Contents
<name>	The name of the Emdros feature.
<valuelow>	This element is optional. If it is present, it contains an integer. The value of the Emdros feature must be greater than or equal to this value.
<valuehigh>	This element is optional. If it is present, it contains an integer. The value of the Emdros feature must be less than or equal to this value.

10.1.1.4 <enumfeature>

The <enumfeature> element contains a description of how an Emdros feature of enumeration type is used for selecting a sentence or a sentence unit.

The <enumfeature> element contains these elements:

Element	Contents
<name>	The name of the Emdros feature.
<comparator>	The string "equals" or "differs". If the string is "equals", the Emdros feature must be equal to one of the <value> elements mentioned below; if the string is "differs", the Emdros feature must not be equal to any of the <value> elements mentioned below.
<value>	This element may occur several times. It contains an enumeration value name that is compared to the value of the Emdros feature.

10.1.1.5 <enumlistfeature>

The <enumlistfeature> element contains a description of how an Emdros feature of type "list of enumeration type" is used for selecting a sentence or a sentence unit.

The <enumlistfeature> element contains these elements:

Element	Contents
<name>	The name of the Emdros feature.
<listvalues>	This element may occur several times. Each specifies a separate selection mechanism. A logical *OR* is assumed between each selection. See below for further information.

The <listvalues> element specifies which enumeration values must occur in the Emdros feature. The <listvalues> element contains these elements:

Element	Contents
<yes>	This element may occur zero or more times. Each contains an enumeration value that must be present in the Emdros feature.
<no>	This element may occur zero or more times. Each contains an enumeration value that must not be present in the Emdros feature.

10.1.1.6 <qerefeature>

The presense of a <qerefeature> element indicates that Hebrew words with a qere form should be omitted in the selection.

The <qerefeature> element contains these elements:

Element	Contents
<name>	The name of the Emdros feature containing the qere form.
<value>	A Boolean value which is always *true*. This indicates that words with a qere form should be omitted from the selection. This value is never *false*, as a false value is indicated by the absense of the <qerefeature> element.

10.2 <quizobjectselection>

The <quizobjectselection> element contains the information entered under the "Sentence Units" tab when creating a quiz template. This element is not present in the quiz template if the <useforquizobjects> element under the <sentenceselection> element is *true*. (See Section 10.1.)

The <quizobjectselection> element contains these elements:

Element	Contents
<questionobject>	The Emdros object type that is used for sentence unit selection.
<mql>	This element is only present if an MQL string is to be used for sentence unit selection. The element contains the MQL string. (See Section 10.4.)
<featurehandlers>	This element is not present if an MQL string is to be used for sentence selection. It contains a description of the features used for sentence unit selection. The format is the same as for <sentenceselection> elements. See Section 10.1.1.

10.3 <quizfeatures>

The <quizfeatures> element lists the display features and request features of the quiz. These must be features of the Emdros object specified in the <questionobject> element of the <quizobjectselection> element (see Section 10.2).

The <quizfeatures> element may contain some or all of the following elements. Each element may occur several times.

Element	Contents
<show>	The name of a display feature.
<request>	The name of a request feature.
<requestdd>	The name of a request feature of string type which should be asked as a multiple-choice question (see Chapter 11).
<dontshow>	The name of a feature that must not be available in the grammar selection box and the grammar information box.
<dontshowobject>	The name of an Emdros object whose features must not be available in the grammar selection box and the grammar information box. If, however, this XML element has an attribute named show, whose value is feature of the object, then that feature will be available. For example, <dontshowobject show="qere_utf8">word</dontshowobject> means that all word features except *qere_utf8* will be unavailable.

10.4 Templates Using MQL for Selection

When creating a quiz template, a facilitator can use a user-friendly selector for specifying which feature values to use when selecting sentences and sentence units. However, for more complex selection criteria, an MQL query string can be specified.

When a quiz template contains an MQL string for *sentence selection,* Bible OL will surround the MQL string by `[sentence ...]`, where the three dots are replaced by the contents of the `<mql>` element under the `<sentenceselection>` element. This means that Bible OL will search for a sentence containing whatever is specified in the MQL statement. It is recommended that facilitators include the word `NORETRIEVE` in the MQL statement as this will cause the program to run considerably faster.

When a quiz template contains an MQL string for *sentence unit selection,* Bible OL will surround the MQL string by `[ttt ...]`, where `ttt` is the contents of the `<questionobject>` element under the `<quizobjectselection>` element, and the three dots are replaced by the contents of the `<mql>` element under the `<quizobjectselection>` element. This means that Bible OL will look in the chosen sentence for sentence units that can be described as specified in the MQL statement. Here, the MQL statement must not contain the characters `[` and `]`; so it is only possible to specify one sentence unit. Furthermore, the word `NORETRIEVE` must not be included in the statement.

The reason for handling the `<mql>` element in these different ways is that the MQL for sentence selection may include multiple query elements, whereas the MQL for sentence unit selection must refer to a single object.

This can be illustrated by the following example: Assume that an exercise is intended to test a student's knowledge of the case of Greek nouns when they follow a preposition. Sentence selection would therefore look for sentences that contain a preposition followed by a noun. This is achieved by this `<sentenceselection>` element:

```
<sentenceselection version="1">
  <questionobject>word</questionobject>
  <mql>[word NORETRIEVE psp=preposition][word NORETRIEVE psp=noun]</mql>
  <useforquizobjects>false</useforquizobjects>
</sentenceselection>
```

Note that the `<mql>` element contains two `[word ...]` blocks.

The sentence units (question objects) should be nouns. This is achieved by this `<quizobject-selection>` element:

```
<quizobjectselection version="1">
  <questionobject>word</questionobject>
  <mql>psp=noun</mql>
</quizobjectselection>
```

Note that the `<mql>` element contains just the `psp=noun` specification.

The student will be asked to provide the case of the noun, so the `<quizfeatures>` element will contain this:

```
<quizfeatures version="3">
  <show>visual</show>
  <request>case</request>
</quizfeatures>
```

When a student runs the exercise, Bible OL will select sentences using this MQL query:

```
[sentence [word NORETRIEVE psp=preposition][word NORETRIEVE psp=noun]]
```

Once a sentence has been chosen, Bible OL will select sentence units using this MQL query:

```
[word psp=noun GET case]
```

(When running this exercise, the student will be asked about the case of all nouns, not just the nouns that follow prepositions; but each sentence is guaranteed to contain at least one preposition/noun pair.)

Chapter 11

Multiple-Choice Questions

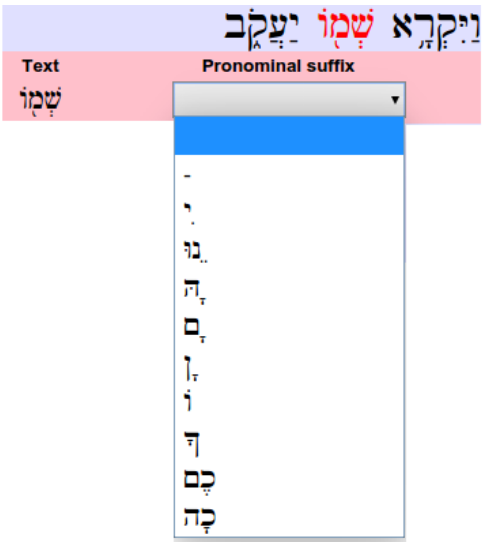
Read this chapter if you need to understand or modify the way Bible OL automatically generates multiple-choice questions for a few features.

If a request feature in an exercise has an enumeration type, the request feature is displayed as a drop-down box and thus becomes a multiple choice question:

But in some cases it is desirable that features of string type are also displayed as multiple choice. For example, in the current version of Bible OL, this is possible for the features *g_word_nocant_utf8*[1] and *g_prs_utf8*[2].

A separate "Words Database" exists with the information necessary for this to work.

The Words Database is an SQLite3 database containing all possible values for the relevant feature. Bible OL then constructs a drop-down box containing at most ten of the possible values, chosen at random but guaranteed to contain the correct answer:



[1] That is, *Text (no cantillation marks).*

[2] That is, *Pronominal suffix.*

The strings in the drop-down box are chosen based on three keys from the *objectSettings* of the Database Specification file. (See Section 9.1.1.)

- The name of the Words Database, found under the *alternateshowrequestDb* key.
- An SQL statement that extracts all possible values of the feature. This is found under the *alternateshowrequestSql* key.
- Features to be used as parameters in the SQL statement. These are found under the *additionalfeatures* key.[3]

Let us look at an example from the ETCBC4 database.[4] For the *g_prs_utf8* feature (the pronominal suffix) of the *word* object, the following values are specified under *objectSettings* in the Database Specification File:

Key	Value
additionalfeatures	`["lex"]`
alternateshowrequestDb	`ETCBC4_words.db`
alternateshowrequestSql	`SELECT DISTINCT suffix FROM suffixes,lexsuf,lexemes WHERE lex='%s' AND lexid=lexemes.id AND sufid=suffixes.id`

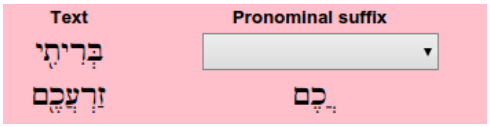
These values will direct Bible OL to replace "%s" in *alternateshowrequestSql* with the value of the *lex* feature retrieved from the Emdros database and then execute the SQL statement on the `ETCBC4_words.db` database.

In the case of the sentence וַיִּקְרָא שְׁמוֹ יַעֲקֹב in the illustration above, the *lex* feature of the word שְׁמוֹ has the value "CM/". Bible OL will therefore execute this SQL statement:

```
SELECT DISTINCT suffix FROM suffixes,lexsuf,lexemes
    WHERE lex='CM/' AND lexid=lexemes.id AND sufid=suffixes.id
```

This will yield a collection of all possible pronominal suffixes associated with the *lex* value "CM/". Bible OL will then choose ten of these at random, while still ensuring that the correct answer (וֹ) is among them, and present them in a drop-down box.

If the SQL statement yields only a single value, it is not turned into a drop-down box as with the second word here:



[3] Currently, only one value is allowed in *additionalfeatures*.

[4] For a detailed description of the Words Database for ETCBC4 see Appendix B.

Chapter 12

Hints

Read this chapter if you need to understand or modify the way Bible OL hinting for some Hebrew verbs.

In some cases, a Hebrew verb form may have several interpretations depending on the context. For example, the word תֵּרְאֶה (Genesis 1 : 9) can be either 2nd person, singular, masculine or 3rd person, singular, feminine. A student looking merely at the word form, will not be able to determine the correct interpretation. In the context of Genesis 1 : 9 the word is 3rd person, singular, feminine. A teacher can choose to display the *hint* feature in exercises to help the student select the correct interpretation. For example, Genesis 1 : 9 may be displayed thus:

A separate "Hints Database" exists with the information necessary for this to work.

The Hints Database is an SQLite3 database containing all possible values for the relevant feature. For a detailed description of the Hints Database see Appendix C.

The *hint* feature is implemented as a pseudofeature (see Section 9.1.2.1). Hints are only available for verbs, and only in situations where the verb form is ambiguous.

Chapter 13

Data Exchange

Read this chapter if you are going to work with code that accesses the Emdros databases on the server or displays text and exercises in the client.

This chapter describes the data exchange between the client (web browser) and the server.

13.1 Displaying Text

A user requests Bible OL to display a particular passage by accessing a URL in one of these formats:

`http://`*hostname*`/text/show_text/`*dsfname*`/`*book*`/`*chapter*
`http://`*hostname*`/text/show_text/`*dsfname*`/`*book*`/`*chapter*`/`*verse*
`http://`*hostname*`/text/show_text/`*dsfname*`/`*book*`/`*chapter*`/`*firstverse*`/`*lastverse*

The first variant retrieves an entire chapter, the second variant retrieves a single verse, and the third variant retrieves a range of verses.

The *dsfname* in the URLs is the primary name of the Database Specification File (Section 9.1); so if the *dsfname* is "ETCBC4-translit", the server will access the Database Specification File `ETCBC4-translit.db.json`.

So, for example, to retrieve Genesis 1 : 2-5 from ETCBC4-translit on the server with hostname `bibleol.3bmoodle.dk`, you can use this URL: `https://bibleol.3bmoodle.dk/text/show_text/ETCBC4-translit/Genesis/1/2/5`.

The server will always expand the requested range of verses to contain a complete set of sentences; so a request for Genesis 1 : 16-17 will automatically be expanded to include verse 18, because verses 17 and 18 comprise a single sentence.

When the server has interpreted the components of the URL, it queries the relevant Emdros database, and based on the result it generates an HTML document containing little more than the relevant headers, HTML code to lay out the menu, and these JavaScript variables:

Variable	Contents
useToolTip	A Boolean value of *true* if the grammar information box should be displayed as a tooltip under the mouse, *false* if the grammar information box should be displayed at the right side of the browser window. This value is configurable on a per-user basis, but currently there is no user interface to change its value.
configuration	A JavaScript object whose value is the contents of the Database Specification File (Section 9.1).
l10n	A JavaScript object whose value is the Grammar Localization Structure (Section 21.4).

(Continued...)

Variable	Contents
l10n_js	A JavaScript object whose key/value pairs provide localized text for the user interface. (Chapter 21.)
typeinfo	A JavaScript object whose value is the contents of the Database Type Information File (Section 9.3).
site_url	The base part of the URL of the website. (For example, `https://bibleol.3bmoodle.dk/`.)
dictionaries	The text to display, including grammar information. This is a JavaScript object in a format defined in Chapter 14.
quizdata	This variable is *null*, indicating that we are displaying text, not running an exercise.

Included in the HTML document that the server sends to the client is a link to a number of CSS and JavaScript files, including `ol.js`, which contains the main piece of code that is to run on the client.

When the client (the web browser) has read the HTML file, the JavaScript code directs it to construct the visual appearance of the text. This involves building the central text layout, adding grammatical information to each word, phrase, clause, etc., and constructing the grammar selection box and the grammar information box.

13.2 Running an Exercise

A user requests Bible OL to start a particular exercise by accessing a URL in one of these formats:

`http://`*hostname*`/text/show_quiz?quiz=`*excercisename*`&count=`*numberOfQuestions*
`http://`*hostname*`/text/show_quiz_univ?quiz=`*excercisename*`&count=`*numberOfQuestions*

The first variant executes an exercise based on the set of Bible passages specified in the `<path>` elements of the quiz template; the second variant asks the user which Bible passages to use, and then starts the exercise. In both cases the *exercisename* is the path name (relative to the `quizzes` directory) of the file containing the exercise; the *numberOfQuestions* is a positive integer indicating the maximum number of questions to ask in the exercise.[1]

So, for example, to run an exercise consisting of ten questions from the quiz template file `Nestle 1904/demo/case.3et` on the server with hostname `bibleol.3bmoodle.dk`, you can use this URL: `https://bibleol.3bmoodle.dk/text/show_quiz?quiz=Nestle%201904/demo/case.3et&count=10`. This will use the pre-defined set of Bible passages stored in the quiz template file.

If the user chooses the second variant of the URL, the server will display a tree of books of the Bible. The user can open book nodes to display a list of chapters, and they can open chapter nodes to display a list of verses. The client retrieves the number of chapters in each book and the number verses in each chapter as needed using AJAX requests from the *jstree* package (see page 25). Once the user clicks the "Start quiz" button, the client sends an HTTP POST request to the server containing the quiz template filename, the number of questions to ask, and the list of selected passages.

The server uses information in the quiz template file and the list of passages (either the list specified by the user or the pre-defined list from the template) to generate a query for the relevant Emdros database. Based on the result of the query, the server generates an HTML document containing little more than the relevant headers, HTML code to lay out the menu, and these JavaScript variables:

Variable	Contents
useToolTip	The same as in Section 13.1.

(Continued...)

[1] If `count` is omitted or if the number is illegal, five questions will be asked.

Variable	Contents
configuration	The same as in Section 13.1.
l10n	The same as in Section 13.1.
l10n_js	The same as in Section 13.1.
typeinfo	The same as in Section 13.1.
site_url	The same as in Section 13.1.
dictionaries	For each question, this variable contains the text to display, including grammar information. This is a JavaScript object in a format defined in Chapter 14.
quizdata	A JavaScript object containing information about the exercise being run. This includes information about the display features and the request features and their (correct) values. This variable is described in Chapter 15.

Included in the HTML document that the server sends to the client is a link to a number of CSS and JavaScript files, including `ol.js`, which contains the main piece of code that is to run on the client.

When the client (the web browser) has read the HTML file, the JavaScript code directs it to construct the visual appearance of the questions. This involves building the central text layout, adding grammatical information to each word, phrase, clause, etc., building the question/answer panel, and constructing the grammar selection box and the grammar information box.

As the user answers the questions, the client keeps track of the answers, but no communication with the server takes place before the user presses the "GRADE task" or "SAVE outcome" button. What happens when one of these buttons is pressed, depends on whether the user is logged in or not. If the user not logged in, the client instructs the server to display the *Select a Quiz* web page. But if the user is logged in, the client sends the result to the server URL `http://`*hostname*`/statistics/update_stat`, which updates the statistics for the user. The record in the user statistics table has a field called *grading*; this field is set to 0 if the user pressed the "SAVE outcome" button, the field is set to 1 if the use pressed the "GRADE task" button. The *grading* field is used by Learning Journey (see Section 22.1).

Chapter 14

The *dictionaries* Variable

Read this chapter if you are going to work with code that accesses the Emdros databases on the server or displays text and exercises in the client.

All information about the text to display, the associated feature values, and the phrase, clause, and sentence structure of the text is communicated between the server and the client in the JavaScript variable *dictionaries*.

This means that the primary task of the server is to convert data from the Emdros database into a format that can be stored in the *dictionaries* variable, and the primary task of the client is to convert the contents of the *dictionaries* variable into displayed text.

In the PHP files executed by the server, the value is described by the *Dictionary* class in the file `myapp/libraries/Dictionary.php`. In the TypeScript files executed by the client, the value is described by *DictionaryIf* interface in the file `ts/dictionary.ts`.

Note: There is also a class in the client called *Dictionary*. This is not the same as the *Dictionary* class in the server (although the two are related, as we shall see in Section 20.2).

14.1 The TypeScript *DictionaryIf* and PHP *Dictionary* Classes

There is a one-to-one mapping of classes and data fields in the server and in the client, but the names differ a little, as you can see from the following.

This is a simplified overview of the relevant TypeScript definitions in the client:

```
interface DictionaryIf {
    sentenceSets : MonadSet[];
    monadObjects: MonadObject[][][];
    bookTitle : string;
}

interface MonadSet {
    segments : MonadPair[];
}

interface MonadPair {
    low : number;
    high : number;
}

interface MonadObject {
    mo : MatchedObject;
    children_idds : number[];
}

interface MatchedObject {
```

```
    id_d : number;
    name : string;
    monadset : MonadSet;
    features : {[key : string] : string;};
    sheaf : any;
}

interface SingleMonadObject extends MonadObject {
    text : string;
    suffix : string;
    bcv : string[];
    bcv_loc : string;
    sameAsNext : boolean[];
    sameAsPrev : boolean[];
    pics : number[];
    urls : any[];
}

interface MultipleMonadObject extends MonadObject {
    subobjects : MatchedObject[][];
}
```

This is a simplified overview of the corresponding PHP definitions in the server:

```
class Dictionary {
    public $sentenceSets; // An array of OlMonadset objects
    public $monadObjects; // An array of arrays of arrays of MonadObject objects
    public $bookTitle;
}

class OlMonadSet implements Iterator {
    public $segments;   // An array of MonadPair objects
}


class MonadPair {
    public $low;
    public $high;
}

abstract class MonadObject {
    public $mo;         // An OlMatchedObject object
    public $children_idds;
}

class OlMatchedObject {
    public $id_d;
    public $name;
    public $monadset;   // An OlMonadSet object
    public $features;   // Maps feature name to feature value
    public $sheaf;
}


class SingleMonadObject extends MonadObject {
    public $text;
    public $suffix;
    public $bcv;        // An array of strings or integers
    public $bcv_loc;    // Localized version of bcv
    public $sameAsNext; // An array of Booleans
    public $sameAsPrev; // An array of Booleans
    public $pics;       // An array of integers
    public $urls;       // An array of string pairs
```

```
}

class MultipleMonadObject extends MonadObject {
    public $subobjects; // MatchedObjects for any subobjects retrieved for this
    MultipleMonadObject
}
```

In the following text, the names from the client definitions are used.
The fields of the *DictionaryIf* interface are:

Field	Contents
sentenceSets	Each element in this array specifies a lump of text, defined by the constituent monads. When using the "Display text" feature of Bible OL, *sentenceSets* contains only a single lump of text, and the array therefore has a single element. When running an exercise, each question uses a separate lump of text, and there are as many entries in the array as there are questions in the exercise. If, for example *sentenceSets[2]* has the value `{"segments": [{"low": 8868, "high": 8877}]}`, it means that the third question (index 2) is based on monads 8868-8877 from the Emdros database.
monadObjects	An array of arrays of arrays of *MonadObject* objects. The first index of this array corresponds to the index in the *sentenceSets* array. The second index selects the level in the grammatical hierarchy (for example, word – phrase – clause – sentence). At the third level we find the actual objects. For example, *monadObjects[0][2][4]* is the *MonadObject* that describes the fifth (index 4) clause (index 2) of the first (index 0) lump of text.
bookTitle	The title of the current book of the Bible. It is used by the client to display an appropriate heading on the webpage when displaying text.

The grammatical information about each object is structured in a grammatical hierarchy. At the lowest level we have *words.* The names of the objects above the words are named *phrase, clause,* and *sentence* in ETCBC4; they are named *clause level 2, clause level 1,* and *sentence* in nestle1904. The names and the depth of the hierarchy can be chosen freely by the database, although the user interface can only display a limited number of levels.

Above the top level, all sentences in a lump of text are grouped together in a so-called "patriarch" object.

Each Emdros object is described by a *MonadObject* in the *monadObjects* array mentioned above. The second index of *monadObjects* identifies the level in the grammatical hierarchy. *MonadObject* is an abstract class, its concrete subclasses are *SingleMonadObject* and *MultipleMonadObject*.

At the lowest level (the word level) the *MonadObjects* belong to the class *SingleMonadObject,* which represents Emdros objects that correspond to a single monad. At the higher levels in the grammatical hierarchy, the *MonadObjects* belong to the class *MultipleMonadObject,* which represents Emdros objects that correspond to a multiple monads.

Consider, for example, the middle sentence of Genesis 1 : 7:[1]

wayyavdēl bên hammayim ʔᵃšer mittaḥat lārāqîₐʕ ûvên hammayim ʔᵃšer mēʕal lārāqîₐʕ [2]

[1] I am using a transliterated text here because a left-to-right orientation will make the following illustrations easier to read.

[2] In English: "and separated the waters that were under the expanse from the waters that were above the expanse."

At grammatical level 0, the words are assigned these monads:

wa-	yyavdēl	bên	ha-	mmayim	ʔᵃšer	mi-	ttaḥat	lā-	-	rāqîₐʕ
101	102	103	104	105	106	107	108	109	110	111

û-	vên	ha-	mmayim	ʔᵃšer	mē-	ʕal	lā-	-	rāqîₐʕ
112	113	114	115	116	117	118	119	120	121

Assuming that the *dictionaries* variable contains just this one lump of text (that is, *sentenceSet* contains only one element), the 21 words are represented as *SingleMonadObjects* in *monadObjects[0][0][0..20]* in the *dictionaries* variable. (The "words" with monads 110 and 120 a null forms of the definite article.)

At grammatical level 1, the words are grouped into phrases thus:

wa-	yyavdēl	bên hammayim ûvên hammayim	ʔᵃšer	mittaḥat lārāqîₐʕ	ʔᵃšer	mēʕal lārāqîₐʕ
101	102	103-105, 112-115	106	107-111	116	117-121

Note how the third phrase consists of two lumps of contiguous monads. The seven phrases are represented as *MultipleMonadObjects* in *monadObjects[0][1][0..6]* in the *dictionaries* variable.

At grammatical level 2, the phrases are grouped into clauses thus:

wayyavdēl bên hammayim ûvên hammayim	ʔᵃšer mittaḥat lārāqîₐʕ	ʔᵃšer mēʕal lārāqîₐʕ
101-105, 112-115	106-111	116-121

The three clauses are represented as *MultipleMonadObjects* in *monadObjects[0][2][0..2]* in the *dictionaries* variable.

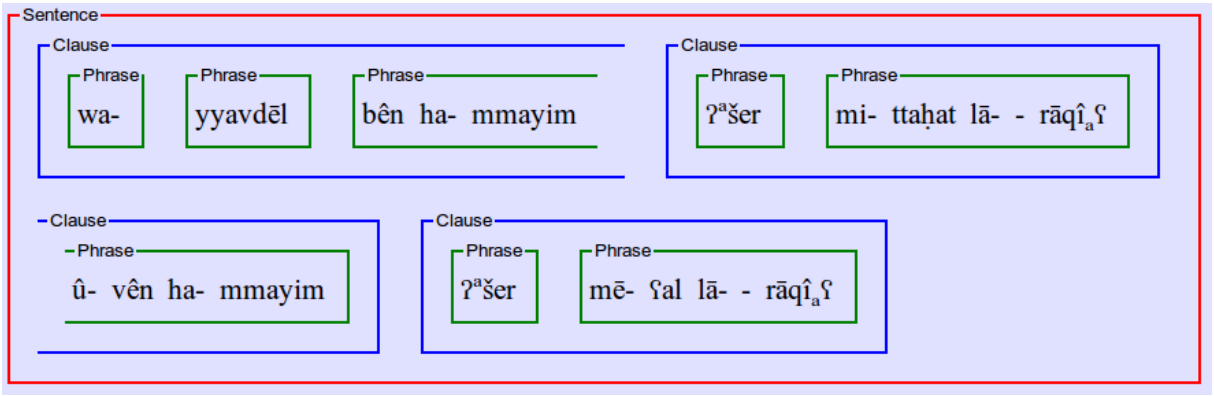
At grammatical level 3, the clauses are grouped into sentences thus:

wayyavdēl bên hammayim ʔᵃšer mittaḥat lārāqîₐʕ ûvên hammayim ʔᵃšer mēʕal lārāqîₐʕ
101-121

In this example, only a single sentence is present. This sentence is represented as a *MultipleMonadObject* in *monadObjects[0][3][0]* in the *dictionaries* variable.

Finally, at grammatical level 4, the sentences are grouped into a single patriarch object. In this example the patriarch contains the same monads as the sentence object. The patriarch is represented as a *MultipleMonadObject* in *monadObjects[0][4][0]* in the *dictionaries* variable.

In Bible OL the grammatical hierarchy can be displayed thus:



Note how the frames around the split phrase and clause are drawn.

14.2 The *MonadObject* Class and Its Subclasses

The previous section describes how objects of class *MonadObject* are used to represent Emdros objects in the *monadObjects* field in the *dictionaries* variable. (Strictly speaking, *MonadObject* is an interface in TypeScript and an abstract class in PHP, but that is irrelevant to the following discussion.)

The *MonadObject* class has these members:

Field	Contents
mo	An object of class *MatchedObject* (*OlMatchedObject* in PHP). A *MatchedObject* is a representation of data about a single Emdros object. Details are given in Section 14.3.
children_idds	An array of integers containing the ID_Ds of the constituent Emdros objects at a lower level in the grammar hierarchy. Consider, for example, the three clauses at grammatical level 2 in the example on page 65. The clause "ʔᵃšer mittaḥat lārāqîₐʕ" contains the two phrases "ʔᵃšer" and "mittaḥat lārāqîₐʕ". If these two phrases have ID_Ds 357 and 362, then *children_idds* of the clause will be the array [357, 362].

At the lowest level (the word level) in the grammar hierarchy, where every Emdros object corresponds to a single monad, the *SingleMonadObject* subclass of *MonadObject* is used to represent an Emdros object.

In addition to the fields mentioned above, a *SingleMonadObject* has these members:

Field	Contents
text	A string containing the text used to display the word.
suffix	A string containing the suffix feature, if any, of the word. (See Section 8.2.2.)
bcv	An array containing the Bible reference for the verse containing the word. For a word in Genesis 1 : 7, *bcv* is the array ["Genesis", 1, 7].
bcv_loc	A localized version of the Bible reference.
sameAsNext	An array of Booleans. *sameAsNext[0]* is *true* if this word belongs to the same book as the next word; *sameAsNext[1]* is *true* if this word belongs to the same chapter as the next word; *sameAsNext[2]* is *true* if this word belongs to the same verse as the next word. For the last word in a collection, all three Booleans are *false*.

(Continued...)

Field	Contents
sameAsPrev	An array of Booleans. *sameAsPrev[0]* is *true* if this word belongs to the same book as the previous word; *sameAsPrev[1]* is *true* if this word belongs to the same chapter as the previous word; *sameAsPrev[2]* is *true* if this word belongs to the same verse as the previous word. For the first word in a collection, all three Booleans are *false*.
pics	An array of integers identifying pictures on the resource website (see Section 23.1) that are relevant for this word. The first three values in the array are the book number, chapter number, and verse number. The remaining elements are IDs of pictures on the resource website.
urls	An array of references to URLs that are relevant for the word. Each entry in the array is itself an array with two elements: the URL and a string identifying the icon to show in Bible OL. If, for example an element in *url* has the value ["http://example.com/here.html", "v"], Bible OL will show a hyperlink to `http://example.com/here.html` in the form of a V icon. For more information see Section 23.1.

In addition to the fields mentioned above, a *MultipleMonadObject* has this member:

Field	Contents
subobjects	Subobjects (for example, *clause_atom* as a subobject of *clause*) in cases where a sentencegrammar refers to such objects.

14.3 The *MatchedObject* Class

A *MatchedObject* (called an *OlMatchedObject* in PHP) is a representation of data about a single Emdros object. It has the following members:

Field	Contents
id_d	The ID_D of the Emdros object
name	The name of the Emdros object type.
monadset	A *MonadSet* (*OlMonadSet* in PHP) object listing the monads belonging to this object. For example, the first clause at grammatical level 2 in the example on page 65 is represented by this value:

```
"monadset": {
    "segments": [
        {
            "low": 101,
            "high": 105
        },
        {
            "low": 112,
            "high": 115
        }
     ]
}
```

features	An associative array mapping feature name to feature value.

(Continued...)

Field	Contents
sheaf	Always null. (This is used by complex MQL searches that should not occur in Bible OL.)

Chapter 15

The *quizdata* Variable

Read this chapter if you are going to work with code that generates exercises on the server or displays exercises in the client.

All information about the questions and correct answers to an exercise is communicated between the server and the client in the JavaScript variable *quizdata*.

In the server, the data is generated by the class *Quiz_data* in the file `myapp/libraries/Quiz_data.php`; in the client the data is described in the TypeScript *QuizData* interface in the file `ts/quizdata.ts`. Th

15.1 The TypeScript *QuizData* and PHP *Quiz_data* Classes

There is a one-to-one mapping of classes and data fields in the server and in the client, but the names differ a little, as you can see from the following.

This is a simplified overview of the relevant TypeScript definitions in the client:

```
interface QuizData {
    quizid : number;
    quizFeatures : ExtendedQuizFeatures;
    desc : string;
    maylocate : boolean;
    monad2Id : number[];
    id2FeatVal : string[][];
}

interface ExtendedQuizFeatures {
    showFeatures : string[];
    requestFeatures : {name : string; usedropdown : boolean; }[];
    dontShowFeatures : string[];
    dontShowObjects : { content : string; show? : string; } [];
    objectType : string;
    dontShow : boolean;
    useVirtualKeyboard : boolean;
}
```

This is a (very) simplified overview of the corresponding PHP definitions in the server:

```
class Quiz_data {
    public $quizid;
    public $quizFeatures;     // An ExtendedQuizFeatures object
    public $desc;
    public $maylocate;
    public $monad2Id;         // An array of integers
    public $id2FeatVal;       // An array of arrays of strings
```

```
}

class ExtendedQuizFeatures {
    public $showFeatures;     // An array of strings
    public $requestFeatures;  // An array of string/Boolean pairs
    public $dontShowFeatures; // An array of strings
    public $dontShowObjects;  // An array of string/string pairs
    public $objectType;
    public $dontShow;
    public $useVirtualKeyboard;
}
```

In the following text, the names from the client definitions are used.
The fields of the *QuizData* interface are:

Field	Contents
quizid	An integer used to identify the entry in the *sta_quiz* table in the user database (see Section 18.12.1) where statistics about this exercise is to be stored. If the user is not logged in, *quizid* is -1 and statistics will not be stored.
quizFeatures	An object of class *ExtendedQuizFeatures*. It contains information about how the exercise should be presented to the user. Details are given below.
desc	The description of the exercise from the quiz template file.
maylocate	A Boolean value indicating if the exercise may display a “Locate” checkbox to the user.
monad2Id	An array that maps monads to the ID_Ds of a quiz object. If a quiz object covers more than one word, several entries in this array will have the same value. (Note that this assumes that a word is not part of more than one quiz object.)
id2FeatVal	An array of arrays of strings. For each quiz object, this array holds information about the values of the display features and the correct values of the request features. If, for example, a quiz object has an ID_D of 1234, and one of the display or request features is *case* with the value *genitive*, then *id2FeatVal[1234]['case']* will have the value “genitive”.

The *ExtendedQuizFeatures* interface has these members:

Field	Contents
showFeatures	An array of strings containing the names of the display features.

requestFeatures	An array of objects describing the request features. Each object has this layout:
	`{`
	`  name : string;`
	`  usedropdown : boolean;`
	`}`
	The *name* field contains the name of the request feature, the *usedropdown* field is *true* if the request feature is of type string and the question should be asked as a multiple choice question (see Section 11).

dontShowFeatures	Array of strings naming features that must not be available in the grammar selection box and the grammar information box.

dontShowObjects	An array of objects naming Emdros object types that must not be available in the grammar selection box and the grammar information box. Each object has this layout:
	`{`
	`  content : string;`
	`  show? : boolean;`
	`}`
	The *content* field contains the name of the Emdros object type, the optional *show* names a feature of that Emdros object that may be displayed nonetheless. Currently, this is only used to display features that are qere forms of Hebrew words.

objectType	The Emdros type of the quiz objects.
dontShow	*True* if the quiz objects should be replaced with (1), (2), (3), etc. in the displayed text.
useVirtualKeyboard	*True* if the client should display a virtual keyboard to facilitate the typing of text in a foreign alphabet.

The *quizFeatures* field of the *QuizData* interface has the additional fields *useDropdown, additionalFeatures,* and *allFeatures.* They are not used by the client software.

Chapter 16

The CodeIgniter Framework

Read this chapter if you are going to understand or modify the server code.

Bible OL uses a PHP framework known as *CodeIgniter*, which provides a simple decoding of URLs, forces a model-view-controller approach to the software structure, and provides a large library for performing a number of tasks. Currently, Bible OL uses CodeIgniter version 3.1.7.

If you are going to modify server code, you will need a good understanding of how CodeIgniter works, and you should therefore read the documentation at http://www.codeigniter.com/docs. The following sections provide a few examples of what CodeIgniter can do for the programmer.

16.1 URL Decoding

When a user accesses a URL such as, for example, `http://website/aaaaa/bbb`, this will cause CodeIgniter to call the PHP function *bbb* in the class *Ctrl_aaaaa*, which is located in the file *ctrl_aaaaa.php*.

It is also possible to access these functions from the shell command line on the server. The shell command

```
php index.php aaaaa bbb
```

is equivalent to accessing `http://website/aaaaa/bbb`.

If the function name is omitted, it defaults to *index*.

16.2 Model-view-controller Structure

It is customary in many large programming projects to split functionality into three groups:

- Models, which are responsible for providing the data that is to be displayed to the user.
- Views, which handle the actual layout on the computer screen.
- Controllers, which handle the flow of data between the models and the views.

CodeIgniter makes structuring PHP code into model-view-controller groups easy.

Continuing with the example from the previous section, the function *bbb* contains the *controller* code.

This controller function may load one or model *models*. If, for example, *bbb* executes this code:

```
$this->load->model('mod_users');
$this->mod_users->get_user_by_id(8);
```

the model class *Mod_users* is loaded from the file *mod_users.php* and the function *get_user_by_id(8)* is called in that class. The *Mod_users* class handles the data exchange with the underlying user database.

Once the controller has retrieved the relevant data, it may load a *view* class which handles the generation of the HTML code presented to the browser. If, for example, *bbb* executes this code:

```
$this->load->view('view_main_page', array('name' => $user_name,
                                          'email' => $user_email));
```

the view file *view_main_page.php* will be loaded and the variables `$user_name` and `$user_email` will be transferred to the file. The code in the view file (mostly HTML) will then be sent to the browser.

16.3 Library Functions

The CodeIgniter library provides a large set of library functions. One of the most important is a set of functions that enable easy and safe construction of SQL statements. For example, instead of the PHP/SQL statement

```
SELECT * from {$db_prefix}user WHERE name=$username AND age=$userage ORDER BY id;
```

you can write this PHP code:

```
$this->db->select('*')
     ->from('user')
     ->where('name', $username)
     ->where('age', $userage)
     ->order_by('id');
```

16.4 Adding Code

Almost all PHP code resides in the directory `myapp`. Controller, model, and view classes are found in the directories `myapp/controllers`, `myapp/models`, and `myapp/views`, respectively. Functions can be added to existing classes, or files containing new classes can be added to the subdirectories.

The code for CodeIgniter itself resides in the directory `CodeIgniter`.

CodeIgniter is configured through definitions in the files located in the directory `myapp/config`. I have added a file `ol.php` to this directory containing these configuration variables:

Index in $config	Value
pw_salt	Salt for storing the password in the user database.
mql_driver	Set to `'native'` to select a built-in MQL driver. Set to or `'extern'` to run MQL commands external to the PHP interpreter.
mail_sender_address	The sender address on email sent by Bible OL to registered users.
mail_sender_name	The sender name on email sent by Bible OL to registered users.
google_client_id	Used by Google login (see Section 17.3).
google_client_secret	Used by Google login (see Section 17.3).
facebook_client_id	Used by Facebook login (see Section 17.3).
facebook_client_secret	Used by Facebook login (see Section 17.3).

Chapter 17

Server Code

Read this chapter if you are going to understand or modify the server code.

This chapter describes some of the techniques and tools you will find in the server code. The server code is written in PHP, and almost all the server code resides in the directory `myapp`. The server code uses the CodeIgniter framework (see Section 16), and a good understanding of how CodeIgniter works is essential to understanding the server code.

17.1 Models, Views, and Controllers

Almost all PHP code used by the server resides in the directory `myapp`. The directories `models`, `views`, and `controllers` hold the main components of the MVC structure supported by CodeIgniter.

The following controllers exist:

Name	Function
classes	Manages classes (that is, groups of users).
config	Allows users to change their font preferences.
file_manager	Management of files and directories in the `quizzes` directory.
lang	Switches user interface language.
login	Handles login using the local user database.
main_page	Displays the main page.
maketypeinfo	This controller can only be accessed from the command line, not through the web interface. It is used to create the Database Type Information File as described in Section 9.3.
migrate	Handles upgrading from one version of Bible OL to another.
oauth2	Handles login using a Google or Facebook account.
pic2db	This controller can only be accessed from the command line, not through the web interface. It is used to retrieve information from the resource website (see Section 23.1).
privacy	Displays the privacy policy.
shebanq	Handles import of MQL from the SHEBANQ website (see Section 20.3.1).
statistics	Updates and displays statistics about the exercises executed by the user.
text	Displays text or exercises. Also handles editing of exercises.

(Continued...)

Name	Function
translate	Allows translators to provide localization of Bible OL.
upload	Receives and stores uploaded exercises.
urls	Manages hyperlinks associated with lexemes.
userclass	Manages a user's relationship to a class.
users	Manages users.

The following models exist:

Name	Function
mod_askemdros	Retrieves Emdros-related data.
mod_classdir	Manages class permissions for an exercise directory.
mod_classes	Manages classes (that is, groups of users).
mod_config	Manages users' font preferences.
mod_intro_text	Generates the text on the front page.
mod_localize	Generates localization information for JavaScript code (see Section 21.3).
mod_quizpath	Exercise directory operations.
mod_statistics	Manages user statistics.
mod_translate	Manages translating Bible OL into different languages.
mod_urls	Manages hyperlinks associated with lexemes.
mod_userclass	Manages a user's relationship to a class.
mod_users	Manages users.

There is no reason to go through the various views here. Their function is best learned by looking for calls like `$this->load->view(...)` in the controller code.

In addition to the model/view/controller classes, the following modules are worth noting:[1]

File	Contents
`core/MY_Controller.php`	Customized version of CodeIgniter's *CI_Controller* class.
`helpers/quiztemplate_helper.php`	XML parser for quiz template files.
`helpers/sheaf_helper.php`	Classes that model data in Emdros replies.
`helpers/sheaf_xml_helper.php`	XML parser for Emdros replies.
`helpers/xmlhandler_helper.php`	Superclass and functions for XML parser.
`libraries/DB_config.php`	Classes for handling Emdros Database Description Files.
`libraries/Dictionary.php`	The *Dictionary* class (see Chapter 14).
`libraries/include/dataexception.inc.php`	Exception classes.
`libraries/include/monadobject.inc.php`	The *MonadObject* class and its subclasses (see Section 14.2).
`libraries/include/typeinfo.inc.php`	The *TypeInfo* class (see Section 9.3).
`libraries/Mql/Mql.php`	The *Mql* class which handles MQL requests (see Section 17.2).

(Continued...)

[1] Note that the list is not complete. Consult the comments in the individual files for more information.

File	Contents
libraries/Mql/drivers/Mql_extern.php	Driver for executing MQL commands through an external MQL command.
libraries/Mql/drivers/Mql_native.php	Driver for executing MQL commands through an MQL library in PHP.
libraries/picdb.php	Class for retrieving picture references and URL references from the resource website (see Section 23.1).
libraries/Quiz_data.php	The *Quiz_data* class and associated functions and class (see Chapter 15).
libraries/Suggest_answers.php	Class for accessing the Words Database (see Chapter 11).
libraries/Universe_tree.php	Classes used together with *jstree* (see page 25) to display a hierarchy of books, chapters, and verses of the Bible.

17.2 MQL Requests in the Server

The server code can be configured to execute MQL requests in one of two ways:

- Adding an MQL library to PHP and calling the MQL API directly from PHP.
- Executing the command line version of MQL from within PHP code.

A driver layer in Bible OL protects the programmer for having to worry about this in most cases; this is described in Section 17.2.1. Information about the two drivers are found in Sections 17.2.2 and 17.2.3.

17.2.1 The MQL Interface to Bible OL

The Driver Library mechanism of CodeIgniter is used to hide the MQL implementation from the programmer in most situations. The programmer must specify the desired way to interact with MQL by setting *$config['mql_driver']* in `myapp/config/ol.php` to either *'native'* (for using the MQL PHP library) or *'extern '* (for using an external MQL command).

The programmer can access MQL in the following way. First, the appropriate MQL driver must be loaded:

```
$this->load->driver('mql',array('db' => $this->db_config->emdros_db,
                                'driver' => $this->config->item('mql_driver')));
```

This is typically done in the *setup* function of the *mod_askemdros* module. Here, *$this->db_config ->emdros_db* is the name of the Emdros database.

After this, MQL requests can be executed like in this example:

```
$emdros_data = $this->mql->exec("SELECT ALL OBJECTS WHERE [word sp=subs GET text] GOqxqxqx");
```

It is important that each Emdros command be terminated by “GOqxqxqx” rather than simply “GO”. The reason is that the MQL driver needs to split a string of several Emdros commands into individual commands. It does this by looking for the string “GOqxqxqx”. If the string “GO” had been used instead, the occurrence of a “GO” inside an MQL command would cause the command splitting to fail. The string “GOqxqxqx” is chosen because it is highly unlikely to occur inside an MQL command.

The call to the *exec* function executes the MQL command and returns the result as an array of *TableOrSheaf* objects. The *TableOrSheaf* class is defined in `myapp/helpers/sheaf_helper.php`.

If the result of the MQL query is a table, the *get_table* function of *TableOrSheaf* will return the table as an *OlTable* object. This object has functions such as *rows, cols, get_header* and *get_cell* which allow you to access various parts of the table.

If the result of the MQL query is a sheaf or a flat sheaf, the *get_sheaf* function of *TableOrSheaf* will return the table as an *OlSheaf* object. This object has functions such as *get_straws, get_first_straw,* and *number_of_straws* which allow you to access the straws within the sheaf.

If you know that the MQL request will return a sheaf and you are only interested in the monads of that sheaf, the so-called "quick harvest" method can be used. In this case you must add a Boolean argument with the value *true* to the call to *exec*:

```
$emdros_data = $this->mql->exec("SELECT ALL OBJECTS WHERE [word sp=subs] GOqxqxqx", true);
```

As before, *exec* returns an array of *TableOrSheaf* objects, but in this case all the objects represent sheafs. As before, calling *get_sheaf* on one of these objects returns an *OlSheaf* object, but in this case the *OlSheaf* only contains *OlMonadSet* objects, and the functions for accessing straws do not work. The function *has_monadset* returns *true* if any *OlMonadSets* are available; and the function *get_monadset* returns an array of *OlMonadSets*.

17.2.2 Driver for Native MQL

The driver for native MQL assumes that MQL support has been added to PHP. Section 3.1.2 explains how to do this.

Using the native MQL driver, the MQL API is available directly from within PHP. A C++ version of the MQL API is described in the Emdros Programmer's Reference Guide[2].

17.2.3 Driver for External MQL

The driver for external MQL relies on the existence of an MQL command line tool on the server. The driver (located in file `myapp/libraries/Mql/drivers/Mql_extern.php`) contains this variable definition:

```
private $command_line = '/usr/local/bin/mql --xml';
```

If the MQL command line program is located in some other directory, this variable definition must be changed accordingly.

Note that there is almost no error reporting from MQL when the external MQL command is used.

17.3 Google and Facebook Login

The server provides two different login mechanism. One uses a local list of users (see Section 18.1), the other relies on a user's Google or Facebook login. Both Google and Facebook provide authentication using the OAuth 2 protocol.

Google's description of how their OAuth implementation works can be found here: https://developers.google.com/accounts/docs/OAuth2WebServer. Facebook's implementation works in a similar manner.

The following is a brief description of the mechanism as it is set up on the Bible OL installation that runs at https://bibleol.3bmoodle.dk.

Step 1. The Browser Sends Authentication Request to Google or Facebook

Google:

When a user clicks "Sign in with Google+" on the login page, the browser sends an HTTP GET request to `https://accounts.google.com/o/oauth2/auth` with the following GET parameters:

[2] http://emdros.org/progref/current

Name	Value
response_type	`'code'`
client_id	Our Google client ID, configured in the file `myapp/config/ol.php`.
redirect_uri	`'https://bibleol.3bmoodle.dk/oauth2/google_callback'`
scope	`'https://www.googleapis.com/auth/userinfo.profile https://www.googleapis.com/auth/userinfo.email'`
state	A random value, stored in CodeIgniter's session mechanism.

Facebook:

When a user clicks "Sign in with Facebook" on the login page, the browser sends an HTTP GET request to `https://www.facebook.com/dialog/oauth` with the following GET parameters:

Name	Value
response_type	`'code'`
client_id	Our Facebook app ID, configured in the file `myapp/config/ol.php`.
redirect_uri	`'https://bibleol.3bmoodle.dk/oauth2/facebook_callback'`
scope	`'email'`
state	A random value, stored in CodeIgniter's session mechanism.

Step 2. Google/Facebook Responds

Google or Facebook checks if the user can be logged in to Bible OL. Google/Facebook then sends a response to the browser, directing it to send a new HTTP GET command to the *redirect_uri* specified in the request above. The parameters for the new GET request depend on whether Google/Facebook approved access or not.

Step 3. Browser Sends Authentication Information to the Bible OL Server

As directed by the response from Google or Facebook, the browser sends an HTTP GET request to `https://bibleol.3bmoodle.dk/oauth2/*_callback` with the following GET parameters:

Name	Value
error	Error information if access is denied. If access is approved, this parameter is not present.
state	The value of the *state* parameter from Step 1.
code	An authentication code generated by Google/Facebook.

Step 4. The Bible OL Server Requests an Access Token from Google/Facebook

In Bible OL, the request from the browser is handled by the function *callback* in the *Ctrl_oauth2* controller (in the file `myapp/controllers/ctrl_oauth2.php`).

Google:

If Google approved access, Bible OL does not respond immediately to the client, but sends an HTTP POST request to `https://accounts.google.com/o/oauth2/token` with the following POST parameters:

Name	Value
code	The authentication code received in the *code* parameter in Step 3.
client_id	Our Google client ID, configured in the file `myapp/config/ol.php`.
client_secret	Our Google client secret, configured in the file `myapp/config/ol.php`.
redirect_uri	`'https://bibleol.3bmoodle.dk/oauth2/google_callback'`
grant_type	`'authorization_code'`

In response to the HTTP POST request, Google replies with a JSON string containing an *access_token*.

Facebook:

If Facebook approved access, Bible OL does not respond immediately to the client, but sends an HTTP GET request to `https://graph.facebook.com/v2.4/oauth/access_token` with the following GET parameters:

Name	Value
code	The authentication code received in the *code* parameter in Step 3.
client_id	Our Facebook app ID, configured in the file `myapp/config/ol.php`.
client_secret	Our Facebook app secret, configured in the file `myapp/config/ol.php`.
redirect_uri	`'https://bibleol.3bmoodle.dk/oauth2/facebook_callback'`

In response to the HTTP GET request, Google replies with a JSON string containing an *access_token*.

Step 5. The Bible OL Server Requests User Information from Google/Facebook

Google:

The Bible OL server sends an HTTP GET request to `https://www.googleapis.com/oauth2/v1/userinfo` with the following GET parameter:

Name	Value
access_token	The *access_token* received from Google in Step 4.

In response to this request, Google replies with a JSON string containing *id, given_name, family_name,* and *email* for the user. If this is the first time the user logs in to Bible OL, the user information is stored in the user table (Section 18.1) of the user database. Bible OL generates a username by concatenating the string "ggl_" with the user *id* received from Google.

Facebook:

The Bible OL server sends an HTTP GET request to `https://graph.facebook.com/v2.4/me` with the following GET parameters:

Name	Value
access_token	The *access_token* received from Google in Step 4.
appsecret_proof	A SHA256 hash of the *access_token* and the Facebook app secret.
fields	`'id,first_name,last_name,email'`

In response to this request, Facebook replies with a JSON string containing *id, first_name, last_name,* and *email* for the user. If this is the first time the user logs in to Bible OL, the user information is stored in the user table (Section 18.1) of the user database. Bible OL generates a username by concatenating the string "fcb_" with the user *id* received from Facebook.

Step 6. User is Logged In

The Bible OL server now responds to the request sent by the browser in Step 3. The response consists of web page informing the user that they are now logged in to the system.

17.4 Account Expiry

A cron job must be set up to run daily. The cron entry must execute this command:

```
php index.php users expire_users
```

This command calls the *expire_users* function in the *Ctrl_users* controller (in the file `myapp/controllers/ctrl_users.php`). The function does four things:

- If a user has not logged in 48 hours after creating an account, the account is deleted.
- If a user has not logged in for nine months, the system emails them a warning.
- If a user has not logged in for seventeen months, the system emails them a warning.
- If a user has not logged in for eighteen months, the account is deleted.

Chapter 18

User Database

Read this chapter if you are going to set up a server installation or if you are going to use the user database

The user database is a MySQL database that contains information about

- Users and classes registered on the system,
- Statistics about exercises,
- Localization strings,
- URLs linked to glosses.

The user database contains these tables:

alphabet	language_comment	sta_displayfeature
bible_refs	language_LANG	sta_question
bible_urls	lexicon_Aramaic	sta_quiz
class	lexicon_Aramaic_LANG	sta_quiztemplate
classexercise	lexicon_greek	sta_requestfeature
db_localize	lexicon_greek_LANG	sta_universe
exercisedir	lexicon_Hebrew	user
exerciseowner	lexicon_Hebrew_LANG	userclass
font	migrations	userconfig
heb_urls	personal_font	

The characters "LANG" in the names above should be replaced with language codes, such as "en" for English or "da" for Danish.

Additionally, the user database contains these tables that are used by Learning Journey (see Section 22.1):

sta_grading sta_gradingfeature sta_gradingpath

The names are typically prefixed by a common text string found in *$db['default']['dbprefix']* in the file `myapp/config/database.php`. If that value is *'bol_'*, the database tables will be named *bol_alphabet, bol_bible_refs* etc.

The tables (except for those used by Learning Journey) are described in the following sections.

18.1 The *user* Table

The *user* table contains information of each registered user. It has the following fields:

Column	Type	Contents
id	Integer	Unique number identifying the user.
first_name	Text	User's first name.
last_name	Text	User's last name
family_name_first	Boolean	True if Chinese name order is used.
username	Text	Name used when logging in.
password	Text	Encrypted password.
reset	Text	Password reset code.
reset_time	Integer	UNIX time when password reset code was issued.
isadmin	Boolean	User is an administrator.
isteacher	Boolean	User is a teacher.
istranslator	Boolean	User is a translator.
email	Text	User's email address.
oauth2_login	Text	OAuth 2 service used to log user in.
created_time	Integer	UNIX time when the account was created.
last_login	Integer	UNIX time when the user last logged in.
warning_sent	Integer	Number of email warnings sent about account inactivity.
preflang	Text	User's preferred language.
accept_policy	Integer	UNIX time when the user accepted the privacy policy.
policy_lang	Text	The language of the accepted privacy policy.
acc_code	Text	Used in the policy acceptance handshake.
acc_code_time	Integer	UNIX time when `acc_code` was generated.

Local users have their username and password stored in this table. Their *oauth2_login* field is set to *NULL* (0).

Google users have usernames such as "ggl_106440263559736360192", where 106440263559736-360192 is the user ID provided by Google. Their *oauth2_login* field is set to *"google"*.

Facebook users have usernames such as "fcb_10206545794576996", where 10206545794576996 is the user ID provided by Facebook. Their *oauth2_login* field is set to *"facebook"*.

The *preflang* field is specified as either the two-letter ISO 639-1 code of the language[1] or "none" for no preferred language.

18.2 The *class* Table

The *class* table contains information of each class (as in "school class", a group of students). It has the following fields:

Column	Type	Contents
id	Integer	Unique number identifying the class.

(Continued...)

[1] For example, "en" for English and "da" for Danish.

Column	Type	Contents
classname	Text	Name of the class.
password	Text	An optional password required when a student enrols in a class.
enrol_before	Date	An optional deadline for enrolment.
ownerid	Integer	The *id* field from an entry in the *user* table. This field identifies the user who owns the class. Typically, this is the teacher who created the class. A field value of zero means that nobody owns the class.

18.3 The *userclass* Table

An entry in the *userclass* table indicates that a particular user is a member of a particular class. The table has these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
userid	Integer	The *id* field from an entry in the *user* table.
classid	Integer	The *id* field from an entry in the *class* table.
access	Boolean	*True* if the user has granted the class owner access to results that are marked as not intended for grading.

18.4 The *userconfig* Table

The *userconfig* table holds information about the configuration for a particular user. Currently, only one option is available, and there is no user interface for configuring it. The table has these fields:

Column	Type	Contents
user_id	Integer	The *id* field from an entry in the *user* table.
usetooltip	Boolean	*True* if the user wants the grammar information box to work as a tooltip instead of having a fixed position on the display.

18.5 The *alphabet* Table

The *alphabet* table contains a list of the foreign alphabets used by Bible OL. It has the following fields:

Column	Type	Contents
id	Integer	Unique number identifying the alphabet.
name	Text	The internal name of the alphabet.
direction	Text	"rtl" for right-to-left text, "ltr" for left-to-right text.
sample	Text	A sample text in the alphabet. This will be displayed when the user chooses fonts.
english	Text	The English name of the alphabet.

18.6 The *font* Table

The *font* table contains the users' font preferences for various alphabets. It has the following fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
user_id	Integer	The *id* field from an entry in the *user* table.
alphabet_id	Integer	The *id* field from an entry in the *alphabet* table.
font_family	Text	A comma-separated string of font names.
text_size	Integer	Font size when displaying text.
text_italic	Boolean	*True* if the font is italic when displaying text.
text_bold	Boolean	*True* if the font is bold when displaying text.
feature_size	Integer	Font size for interlinear text.
feature_italic	Boolean	*True* if the font is italic for interlinear text.
feature_bold	Boolean	*True* if the font is bold for interlinear text.
tooltip_size	Integer	Font size for text in the grammar information box.
tooltip_italic	Boolean	*True* if the font is italic for text in the grammar information box.
tooltip_bold	Boolean	*True* if the font is bold for text in the grammar information box.
input_size	Integer	Font size in input fields.
input_italic	Boolean	*True* if the font is italic in input fields.
input_bold	Boolean	*True* if the font is bold in input fields.

18.7 The *personal_font* Table

Each user can specify one personal font per alphabet. The personal font is listed on the font selection page together with the system fonts.

The *personal_font* table contains a user's personal fonts for a specific alphabet. It has the following fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
user_id	Integer	The *id* field from an entry in the *user* table.
alphabet_id	Integer	The *id* field from an entry in the *alphabet* table.
font_family	Text	The name of the font.

18.8 The *exercisedir* and *classexercise* Tables

Together, the *exercisedir* and *classexercise* tables control which classes have access to which exercise directories. The *exercisedir* assigns an ID to each exercise directory. It has the following fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
pathname	Text	The pathname of a directory, relative to the `quizzes` directory.

The *classexercise* has an entry for each class that is allowed to access a given directory. It has the following fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
classid	Integer	The *id* field from an entry in the *class* table. Users belonging to this class have access to the exercises in the directory identified by the field *pathid*. If the *classid* field is 0, everybody has access to the exercises in the directory identified by the field *pathid*.
pathid	Integer	The *id* field from an entry in the *exercisedir* table.

18.9 The *exerciseowner* Table

The *exerciseowner* table contains information about the owner of an exercise. It has the following fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
pathname	Text	The pathname of the exercise file, relative to the `quizzes` directory.
ownerid	Integer	The *id* field from an entry in the *user* table. This field identifies the user who owns the exercise. Typically, this is the teacher who created the exercise. A field value of zero means that nobody owns the exercise.

18.10 The *bible_refs* Table

The *bible_refs* table contains links between Bible verses and pictures on the resource website (see Section 23.1) that are relevant for the verse. The table contains these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
book	Text	The name of the book. (This is the name used internally in the Emdros database.)
booknumber	Integer	The number of the book. This is the same information held in the *book* field. The file `myapp/controllers/ctrl_pic2db.php` contains an array that translates between book name and book number.
chapter	Integer	The chapter.
verse	Integer	The verse.
picture	Integer	The number of a picture on the resource website.

18.11 The *bible_urls* Table

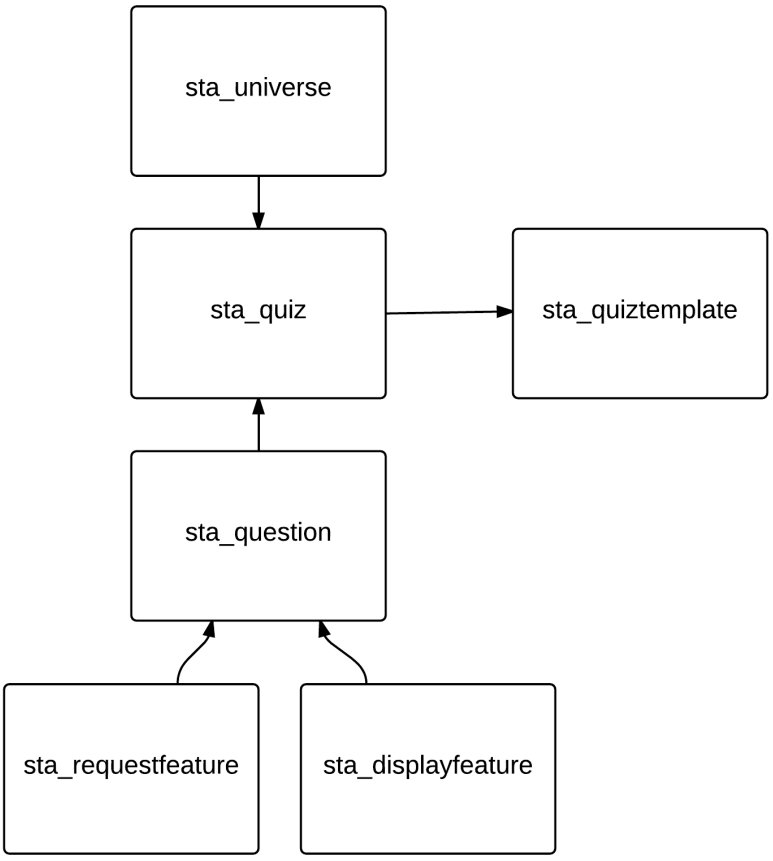
The *bible_urls* table contains links between Bible verses and URLs.[2] The table contains these fields:

[2] Currently, the URLs are configured on the resource website (see Section 23.1), but they could come from other sources.

Column	Type	Contents
id	Integer	Unique number identifying this entry.
book	Text	The name of the book. (This is the name used internally in the Emdros database.)
booknumber	Integer	The number of the book. This is the same information held in the *book* field. The file `myapp/controllers/ctrl_pic2db.php` contains an array that translates between book name and book number.
chapter	Integer	The chapter.
verse	Integer	The verse.
url	Text	The relevant URL.
type	Text	A single character identifying the type of icon to display. This character must be either D (for "Document"), V (for "Video"), or U (for "other URL").

18.12 The Statistics Tables

The six tables with names starting with *sta_* contain statistics about how well a user performed in an exercise. The following figure illustrates the relationship between the six tables.



In this illustration, each arrow indicates a many-to-one relationship, with one item at the arrow head and many items at the other end of the arrow.

18.12.1 The *sta_quiz* Table

Every time a user starts running an exercise, an entry is created in the *sta_quiz* table. It contains these fields:

Column	Type	Contents
id	Integer	Unique number identifying this execution of an exercise.
userid	Integer	The *id* field from an entry in the *user* table. This identifies the user running the exercise.
templid	Integer	The *id* field from an entry in the *sta_quiztemplate* table. This identifies the quiz template used for this exercise.
start	Integer	The start time (UNIX time[3]).
end	Integer	The end time (UNIX time). This value is NULL if the exercise is still running or if the exercise was aborted without saving the result.
valid	Boolean	*False* if the user has deleted the entry, *true* otherwise. (There is currently no user interface for deleting statistics.
grading	Boolean	*True* if this entry is to be used for grading purposes, *false* otherwise. (This is controlled by the user clicking "GRADE task" or "SAVE outcome" at the end of an exercise.)

18.12.2 The *sta_quiztemplate* Table

Every time a user starts running an exercise, the system checks if the quiz template is already stored in the *sta_quiztemplate* table. If not, an entry is created. A template is identified by its contents; if, therefore, a facilitator changes the contents of a quiz template, a new entry will be created in the *sta_quiztemplate* table the next time the exercise is run.

The table contains these fields:

Column	Type	Contents
id	Integer	Unique number identifying this quiz template.
userid	Integer	The *id* field from an entry in the *user* table. This identifies the user running the exercise. For historical reasons, each user has their own set of templates in this table.
pathname	Text	The full pathname of the quiz template file.
dbname	Text	The name of the Emdros database on which the quiz template is based.
dbpropname	Text	The name of the Grammar Localization Structure for the Emdros database on which the quiz template is based.
qoname	Text	The Emdros type name of the sentence unit (quiz object) on which the exercise is based.
quizcode	Text	The actual XML text of the quiz template. Since this text contains both the database name, the name of the Grammar Localization Structure, and the sentence unit, the table fields *dbname, dbpropname,* and *qoname* are actually superfluous, but they are included as separate fields to make decoding the XML text unnecessary in most cases.

(Continued...)

[3] That is, seconds since 00:00:00 UTC on 1 January 1970.

Column	Type	Contents
quizcodehash	Integer	A hash value of the *quizcode* field. It can be used to speed up the comparison of the quizcode field from different entries in this table: If the *quizcodehash* values are different, then the *quizcode* values will also be different.

18.12.3 The *sta_universe* Table

Each entry in the *sta_universe* table represents a single book, chapter, or verse from the Bible. Together, a number of entries with the same *quizid* field identify the passages used for generating a particular exercise.

The *sta_universe* table contains these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
userid	Integer	The *id* field from an entry in the *user* table.
quizid	Integer	The *id* field from an entry in the *sta_quiz* table.
component	Integer	A reference to a single book, chapter, or verse. The format is either "Genesis", "Genesis:3", or "Genesis:3:8".

18.12.4 The *sta_question* Table

Each entry in the *sta_question* table represents a single question, as defined in Section 1.3 (see Figure 1.1 on page 6). The table contains these fields:

Column	Type	Contents
id	Integer	Unique number identifying this question.
userid	Integer	The *id* field from an entry in the *user* table.
quizid	Integer	The *id* field from an entry in the *sta_quiz* table.
txt	Text	The text of the question. The quiz objects are enclosed between `<em>` and `</em>`.
location	Text	The Bible reference for the text, given in the format "Genesis, 3, 8".
time	Integer	The time (UNIX time) when this question was answered.

18.12.5 The *sta_displayfeature* Table

Each entry in the *sta_displayfeature* table lists a display feature that was shown for a question item (see Section 1.3 and Figure 1.1 on page 6 for a definition of "question item".)

The table contains these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
userid	Integer	The *id* field from an entry in the *user* table.
questid	Integer	The *id* field from an entry in the *sta_question* table.
qono	Integer	The index (starting from 1) of the question item within the question.
name	Text	The name of the feature.
value	Text	The value of the feature.

18.12.6 The *sta_requestfeature* Table

Each entry in the *sta_requestfeature* table lists a request feature that was required for a question item (see Section 1.3 and Figure 1.1 on page 6 for a definition of "question item".)

The table contains these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
userid	Integer	The *id* field from an entry in the *user* table.
questid	Integer	The *id* field from an entry in the *sta_question* table.
qono	Integer	The index (starting from 1) of the question item within the question.
name	Text	The name of the feature.
value	Text	The correct value of the feature.
answer	Text	The answer provided by the user. If the user tries to answer several times in a single exercise, only the first answer is recorded.
correct	Boolean	*True* if the user's answer is correct.[4]

18.13 The *heb_urls* Table

The *heb_urls* table keeps track of the hyperlinks associated with Hebrew or Aramaic lexemes.

The table contains these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
lex	Text	The value of the *lex* feature from the ETCBC4 database.
language	Text	The value of the *language* feature from the ETCBC4 database.
url	Text	The URL that is the destination of the hyperlink.
icon	Text	The name of an icon (typically taken from the list of Bootstrap icons) to use for this hyperlnk.

18.14 The *migrations* Table

The *migrations* table is maintained by the migration mechanism of CodeIgniter. This mechanism keeps track of system updates. It contains this field:

Column	Type	Contents
version	Integer	Current system version number as used by CodeIgniter.

18.15 The Localization Tables

The *db_localize* table and the tables with names starting with *language_* or *lexicon_* contain information for the localization of text.

[4] This is not the same as testing if *value=answer*. For example, when providing an English translation of a word, a correct *answer* may be "do" even if the *value* is "make, do; fix; deal with".

18.15.1 The *db_localize* Table

The *db_localize* table contains the Grammar Localization Structures, that is, the translation of grammar terms into various languages. For details, see Section 21.4.

The table contains these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
db	Text	The name of the Grammar Localization Structure.
lang	Text	The language code code for the translation ("en" for English, "da" for Danish, etc.). The special value "comment" is used for an entry providing context and formatting information used in the tranlator's interface (see Section 21.5).
json	Text	A JSON string containing the Grammar Localization Structure.

18.15.2 The *language_comment* Table

The *language_comment* table contains information describing the available translation strings for the user interface. This table is used by the translator's interface. Each *textgroup*/*symbolic_name* pair in this table matches an entry in each *language_LANG* table.

The table contains these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
textgroup	Text	See Section 21.2.
symbolic_name	Text	See Section 21.2.
comment	Text	A string providing the human translator with context for when the string is used.
format	Text	If this value is "keep_blanks", whitespace characters in the string are important. If the value is *NULL*, multiple contiguous whitespace characters are replaced by a single space.
use_textarea	Boolean	*True* if the translator's interface should use a `<textarea>` HTML element for input. *False* if an `<input type="text">` HTML element should be used.

18.15.3 The *language_LANG* Tables

The system contains a number of tables with names such as *language_en* for English, *language_da* for Danish, etc. They provide translations of the user interface.

The tables contain these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
textgroup	Text	See Section 21.2.
symbolic_name	Text	See Section 21.2.
text	Text	The relevant string translated into language *LANG*.

18.15.4 The *lexicon_Hebrew* and *lexicon_Aramaic* Tables

The *lexicon_Hebrew* table contains information about all the Hebrew lexemes used by the system. Similarly, the *lexicon_Aramaic* table contains information about all the Aramaic lexemes.

The tables contain these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
lex	Text	The lexeme in the *transcribed alphabet* (see page 28).
vs	Text	The verbal stem of the word. "NA" if the word is not a verb.
tally	Integer	The number of occurrences of this word in the text.
vocalized_lexeme_utf8	Text	The lexeme in Hebrew characters.
roman	Text	An optional roman numeral to differentiate between identically spelled lexemes.
sortorder	Text	A string that can be used to sort the words alphabetically.
firstbook	Text	
firstchapter	Integer	The book, chapter, and verse of the first occurrence of the word.
firstverse	Integer	

An entry is uniquely defined by its *lex* and *vs* fields.

18.15.5 The *lexicon_Hebrew_LANG* and *lexicon_Aramaic_LANG* Tables

The system contains a number of tables with names such as *lexicon_Hebrew_en* for English and *lexicon_Hebrew_de* for German. They provide translations for the lexemes listed in the *lexicon_Hebrew* table. Similar tables, such as *lexicon_Aramaic_en,* provide translations for the lexemes listed in the *lexicon_Aramaic* table.

The tables contain these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
lex_id	Integer	The *id* field of the entry in the *lexicon_Hebrew* or *lexicon_Aramaic* table for which the current entry provides a translation.
gloss	Text	The translation of the lexeme into language *LANG*.

18.15.6 The *lexicon_greek* Table

The *lexicon_greek* table contains information about all the Greek lexemes used by the system.

The tables contain these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
strongs	Integer	The Strong's number for the lexeme.
strongs_unreliable	Boolean	*True* if the indicated Strong's number is considered unreliable.
lemma	Text	The lexeme in accented Greek characters.

(Continued...)

Column	Type	Contents
tally	Integer	The number of occurrences of this word in the text.
sortorder	Text	The lexeme in unaccented, lowercase Greek characters, which can be used to sort the words alphabetically.
firstbook	Text	
firstchapter	Integer	The book, chapter, and verse of the first occurrence of the word.
firstverse	Integer	

18.15.7 The *lexicon_greek_LANG* Table

The system contains a number of tables with names such as *lexicon_greek_en* for English. They provide translations for the lexemes listed in the *lexicon_greek* table.

The tables contain these fields:

Column	Type	Contents
id	Integer	Unique number identifying this entry.
lex_id	Integer	The *id* field of the entry in the *lexicon_greek* table for which the current entry provides a translation.
gloss	Text	The translation of the lexeme into language *LANG*.

Chapter 19

Less Style Sheets

Read this chapter if you are going to understand or modify style sheets.

CSS style sheets can sometimes be unwieldy to work with. The *Less* program is a CSS preprocessor that allows a clearer way to structure style sheets.

Compare, for example, the CSS code in the left column below with the Less code in the right column:

CSS

```
ul.dropdown {
  padding: 0;
}

ul.dropdown a {
  text-decoration: none;
}

ul.dropdown li {
  display: inline-block;
  background: #f3d673;
  z-index: 1;
}

ul.dropdown li:hover {
  background: #c0c0c0;
  position: relative;
}

ul.dropdown li a {
  color: black;
  display: block;
}
```

Less

```
@beige: #f3d673;
@mediumgray: #c0c0c0;

ul.dropdown {
    padding: 0;

    a {
        text-decoration: none;
    }

    li {
        display: inline-block;
        background: @beige;
        z-index: 1;

        &:hover {
            background: @mediumgray;
            position: relative;
        }

        a {
            color: black;
            display: block;
        }
    }
}
```

Section 3.2.3 describes how to install the Less compiler *lessc*. Details of the Less language can be found at `http://lesscss.org`.

Although Less style files can be compiled when used in a browser, the Bible OL implementation compiles Less files only once and stores the resulting CSS files. This is achieved through the Makefile in the top directory. The command "make styles/ol.css" will compile the Less file.[1]

[1] The simple command "make" will compile all Less and TypeScript files.

At present, Bible OL uses only one Less file, namely `styles/ol.less` which compiles into `styles/ol.css`.

Chapter 20

Client Code

Read this chapter if you are going to understand or modify the client code.

The client code runs in a web browser. Most of it is written in TypeScript which is compiled into JavaScript.

There are three different TypeScript programs that can run as client code:

- *ol,* which displays text or runs exercises. (See Section 20.2.)
- *editquiz,* which edits a quiz template.
- *fontselector,* which allows a user to set font preferences.

20.1 TypeScript

TypeScript is a superset of JavaScript that adds strong typing and proper classes to JavaScript. The website `http://www.typescriptlang.org` contains a tutorial and the formal specification of the language.

Section 3.2.4 describes how to install the TypeScript compiler *tsc.*

The Bible OL implementation compiles TypeScript files only once and stores the resulting JavaScript files. This is achieved through the Makefile in the top directory. The command "make all" (or simply "make") will compile the TypeScript files (and the Less file). The Makefile checks that the *tsc* compiler version is 1.0.1.0. This is probably unnecessary, and should be removed as newer versions appear.

The TypeScript files are found in the directory `ts`; the resulting JavaScript files are stored in the directory `js`.

20.2 The *ol* Client Code

The *ol* client code is responsible for displaying text and running an exercise based on information provided by the server – primarily in the JavaScript variables *configuration, l10n, l10n_js, typeinfo, dictionaries,* and *quizdata,* which are described in detail in Chapters 13 and 14.

For text display, the *ol* program builds the text inside an HTML skeleton provided by the server. The skeleton looks like this (somewhat simplified):

```
<div class="grammarselector" id="gramselect"></div>
<div class="grammardisplay"></div>

<div id="textcontainer">
    <h1></h1>
    <div id="textarea"></div>
    <p><button id="togglemql">Toggle MQL</button></p>
    <pre class="mqlarea">SELECT ALL OBJECTS WHERE ...</pre>
</div>
```

When running an exercise, the *ol* program builds a question inside an HTML skeleton provided by the server. The skeleton looks like this (somewhat simplified):

```
<div class="grammarselector" id="gramselect"></div>
<div class="grammardisplay"></div>

<div id="textcontainer">
    <div id="quizdesc"></div>
    <div id="textarea"></div>
    <div id="virtualkbcontainer"><div id="virtualkbid"></div></div>
    <table id="quiztab"></table>
    <p><input id="locate_cb" type="checkbox">Locate: <span class="location"></span></p>

    <p class="inline">Progress:</p>
    <progress id="progress" value="0" max="1"></progress>
    <div id="progressbar"></div>
    <p id="progresstext" class="inline"></p>

    <div id="buttonlist1">
        <button id="check_answer" type="button">Check answer</button>
        <button id="show_answer" type="button">Show answer</button>
    </div>
    <div id="buttonlist2">
        <button id="next_question" type="button">Next</button>
        <button id="finish" type="button">GRADE task</button>
        <button id="finishNoStats" type="button">SAVE outcome</button>
    </div>

    <p><button id="togglemql">Toggle MQL</button></p>
    <pre class="mqlarea">SELECT ALL OBJECTS WHERE ...</pre>
</div>
```

The `<div class="grammarselector">` element is for the grammar selection box. The *ol* program generates the contents of this element using the class *GenerateCheckboxes*.

The `<div class="grammardisplay">` element is for the grammar information box. It is built by the function *toolTipFunc*, which is defined within the function *generateSentenceHtml* in the *Dictionary* class.

The `<div class="textcontainer">` element contains the text and, possibly, the question. The actual text is placed in the `<div class="textarea">` element. The question items are placed in the `<table id="quiztab">` table.

The `<button id="togglemql">` button and the `<pre class="mqlarea">` element are normally not shown to the user. They are intended for debugging only. The `<pre class="mqlarea">` element contains the MQL commands executed during the creation of the text or exercise. You can either inspect the element by looking at the HTML source sent to the browser, or you can enable the `<button id="togglemql">` button by removing "`display: none`" from this instruction in `styles/ol.less`:[1]

```
button#togglemql {
    display: none;
}
```

When the "`display: none`" line has been removed, a "Toggle MQL" button will appear in the browser. Clicking the button will display the MQL commands executed during the creation of the text or exercise.

* * *

[1] After modifying `styles/ol.less`, you must recompile the Less file. For simple debugging, you may prefer to edit the `styles/ol.css` file directly.

The most complicated task for *ol* is probably to build the contents of the *textarea*, and this will be described in some detail below.

As Chapter 14 explains, the *dictionaries* variable contains the field *sentenceSets,* which is an array of *MonadSet* objects, and the field *monadObjects,* which is an array of arrays of arrays of *MonadObject* objects. When Bible OL is displaying text, the array *sentenceSets* and the top array in *monadObjects* have only one element; but when Bible OL is displaying an exercise consisting of *n* questions, the two arrays have *n* elements.

The *ol* program converts the *dictionaries* variable (which is of interface class *DictionaryIf*) into one or more objects of class *Dictionary*, one for each entry in the *MonadSets/MonadObject* arrays. (Note the *Dictionary* class here must not be confused with the *Dictionary* class in the server. The server's *Dictionary* class corresponds to the client's *DictionaryIf* interface.)

As explained in Section 14.1, the *monadObjects* field of the *DictionaryIf* interface is an array of array of arrays. The middle array is indexed by the level in the grammatical hierarchy (word, phrase, clause, etc.). As part of creating a *Dictionary* from a given index in a *DictionaryIf*, the *constructor* function in the *Dictionary* class builds a parallel collection of arrays: Each *MonadObjects* is complemented by one or more *DisplayMonadObjects.* A *DisplayMonadObject* represents the physical appearance of an Emdros object in the browser. *DisplayMonadObject* has a member function, *generateHtml,* which is responsible for generating the HTML that renders the Emdros object.

Just as a *MonadObject* is either a *SingleMonadObject* or a *MultipleMonadObject*, a *DisplayMonadObject* is either a *DisplaySingleMonadObject* (typically representing a word) or a *DisplayMultipleMonadObject* (typically representing a phrase, clause, or sentence). There is, however, an important difference between a *MultipleMonadObject* and a *DisplayMultipleMonadObject.* If, for example, a clause consists of multiple noncontiguous parts, it is represented by one *MultipleMonadObject* but by multiple *DisplayMultipleMonadObjects,* one for each part of the clause.

When the *generateHtml* is called for a *DisplaySingleMonadObject,* its task is to create HTML code to represent a single word and all its features. The code generated is structured as shown in Listing 20.1.

LISTING 20.1: HTML display structure for a word object

```
<span class="textblock inline">
    <span class="textdisplay charset" data-idd="ID_D">text</span>

    <span class="wordgrammar dontshowit featurename charset">featurevalue</span>
    <span class="wordgrammar dontshowit featurename charset">featurevalue</span>
    <span class="wordgrammar dontshowit featurename charset">featurevalue</span>
    ...

</span>
```

Here,

charset	identifies the character set and hence the font and text direction. Valid values are `hebrew`, `hebrew_translit`, `greek`, `latin`, and `ltr`. The value `ltr` is used to force left-to-right for features in Latin script. The value `latin` is currently not used; it is reserved for corpuses that use the Latin alphabet.
ID_D	is the ID_D (see Chapter 6) of the word.
text	is the actual word.
featurename	is the non-localized name of the feature, that is, the name of the feature as it appears in the Emdros database.
featurevalue	is the localized value of the feature.

If a particular feature is turned on in the grammar selection box, the `dontshowit` class in the relevant `<span>` elements is changed to `showit`.

Two extra class values are added to the `<span class="textdisplay">` element to control the rendering of Hebrew word spacing.

The first class value is one of these and doesn't change:

Class value	Meaning
`cont`	The word must be followed immediately by the next word with no intervening space.
`contx`	The word ends in a *maqaf* (־) and must be followed immediately by the next word with no intervening space.

If neither `cont` nor `contx` is set on a word, a `<span>` element with class `wordspace` is inserted after the word.

The second class value is one of these, and it changes as the user switches between display and not displaying word spacing:

Class value	Meaning
`cont1`	The user has not requested word spacing. Use default rending of words.
`cont2`	The user has requested word spacing. Add a hyphen and a space to the end of the current word.
`cont2x`	The user has requested word spacing. The current word ends in a *maqaf*. Add a space to the end of the current word.

When the *generateHtml* is called for a *DisplayMultipleMonadObject,* its task is to create HTML code to represent a phrase, clause, or sentence object and all its features. The code generated is structured as shown in Listing 20.2.

LISTING 20.2: HTML display structure for a phrase/clause/sentence object

```
<span class="notdummy nolevel noseplin">
    <span class="gram dontshowit" data-idd="ID_D">loctype
         <span class="xgrammar dontshowit type_featurename">:featurevalue</span>
         <span class="xgrammar dontshowit type_featurename">:featurevalue</span>
         <span class="xgrammar dontshowit type_featurename">:featurevalue</span>
         ....
    </span>
    ... (Lower levels in the grammar hierarchy are inserted here)
</span>
```

Here,

nolevel	is one of `nolev1`, `nolev2`, `nolev3`, etc. The number within this name identifies the level in the grammar hierarchy: 1 for the level just above *word*, 2 for the next higher level, etc.
ID_D	is the ID_D (see Chapter 6) of the word.
loctype	is the localized name of the Emdros object type.
type	is the non-localized name of the Emdros object type, that is, the name of the type as it appears in the Emdros database.
featurename	is the non-localized name of the feature, that is, the name of the feature as it appears in the Emdros database. Note that *type* and *featurename* are strung together with an intervening underscore, thus forming a single class value.
featurevalue	is the localized value of the feature.

The `<span class="notdummy">` element may additionally have the class value `hasp` and/or `hass`. This indicates that the current *DisplayMultipleMonadObject* is part of a noncontiguous collection of monads. The class value `hasp` means that the *DisplayMultipleMonadObject* has a predecessor; the class value `hass` means that the *DisplayMultipleMonadObject* has a successor.

If "Show border" is selected in the grammar selection box for a particular level in the grammar hierarchy, the `dontshowit` class in the relevant `<span class="gram">` elements is changed to `showit`.

If “Separate lines” is selected in the grammar selection box for a particular level in the grammar hierarchy, the `noseplin` class in the relevant `<span class="notdummy">` elements is changed to `seplin`.

If a particular feature is turned on in the grammar selection box, the `dontshowit` class in the relevant `<span class="xgrammar">` elements is changed to `showit`.

Occasionally[2], a level in the grammar hierarchy is missing. If this is the case, the server code will insert a dummy object in the hierarchy, and the client will generated this HTML code:

```
<span class="nolevel noseplin">
    <span class="nogram dontshowit" data-idd="ID_D">loctype
    </span>
    ... (Lower levels in the grammar hierarchy are inserted here)
</span>
```

Note the absence of the `notdummy` class value.

At the top (patriarch) level the `notdummy` class name is omitted, and at this level the HTML simply looks like this:

```
<span class="nolevel noseplin">
    ... (Lower levels in the grammar hierarchy are inserted here)
</span>
```

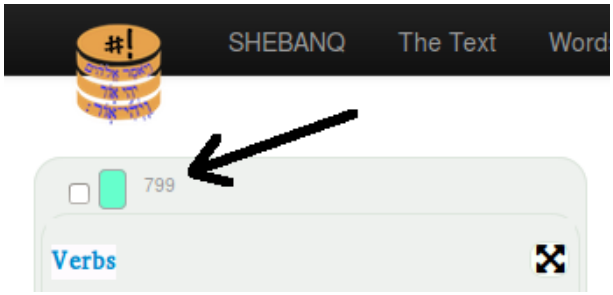
20.3 The *editquiz* Client Code

The *editquiz* client code is not described in detail here. Its purpose is to allow users to edit a quiz template. There is, however, one feature that requires some explanation: The ability to import queries from SHEBANQ.

20.3.1 Import from SHEBANQ

The SHEBANQ website (see Section 23.2) allows its users to create MQL queries using the ETCBC4 Hebrew database. Such queries can be imported into Bible OL.

First, the user must identify the query ID at the SHEBANQ website. The number is found in the upper left corner of a query page at SHEBANQ:



In this example, the query ID is 799.

When editing a quiz template on Bible OL, a user can request the client code to import this query from SHEBANQ.

The *editquiz* client code sends a request to the *import_shebanq* function in the *ctrl_shebanq* controller. This in turn sends a request to `https://shebanq.ancient-data.org/hebrew/query.json?id=799` which returns a JSON representation of the MQL query.

[2] Currently only in the nestle1904 database.

Chapter 21

Internationalization and Localization

As a developer, you must read this chapter.

21.1 CodeIgniter's Internationalization Mechanism

CodeIgniter provides a mechanism for developing internationalized software. The basic rule is never to write English text directly in the code. All text strings must be given a name, and a language-specific version of that text is stored in an array called *$lang*.

The language specific strings are located in the directory `myapp/language`. This directory has a subdirectory for each supported language. The name of the subdirectory is the international two-letter abbreviation of the language as specified in the ISO 639-1 standard. For example, the Danish translation is stored in the subdirectory `da`, and the German translation is stored in the subdirectory `de`. The only exception is English which is stored in a subdirectory called `english` rather than `en`.

Accessing the translation of a string is a two-stage process. First, the relevant file must be loaded from `myapp/language`; this is done thus:

```
$this->lang->load($filename, $language);
```

where *$filename* is the name of the relevant file in `myapp/language`, and *$language* is the language code.

Secondly, the actual string must be loaded:

```
$x = $this->lang->line($string_index);
```

where *$string_index* is a string that identifies the relevant text.

21.2 Bible OL's Modifications to CodeIgniter's Mechanism

CodeIgniter's mechanism is based on having translations located in PHP files. This is not convenient when online updates of the translations are required. In Bible OL the mechanism has therefore been modified so that the translation texts are primarily taken from the user database.

The file `myapp/core/MY_Lang.php` provides an extended mechanism for loading localization information. When executing the statement

```
$this->lang->load($filename, $language);
```

the system will look in the table *language_$language* in the user database (see Section 18.15.3). Here, it will search for all strings whose *textgroup* field is the value specified in the *$filename* parameter. All these strings are loaded into memory, and can be indexed using the value of the *symbolic_name* field as *$string_index*:

```
$x = $this->lang->line($string_index);
```

If the relevant *textgroup* is not found in the user database, the system reverts to the old mechanism.

21.3 Internationalization of the Client Code

This section deals with creating an internationalized and localized version of the TypeScript client code.

As an example, let us assume that we want the client to display the text "Elephants are big". In a non-internationalized version, this might be achieved by this code:

```
$('#xxx').text('Elephants are big');
```

(Here *xxx* is the ID of the HTML element we wish to modify.)

All localization information for the client is found in the *language_LANG* table as records with the *textgroup* field set to "js". Before passing control to the client, the server reads all these records and stores them as key/value pairs in the JavaScript variable *l10n_js*.

Now, if the translation of "Elephans are big" is found under the *symbolic_name* "elephant_size", the original TypeScript code must be replaced by:

```
$('#xxx').text(localize('elephant_size'));
```

The *localize* function loads the appropriate translation from the *l10n_js* variable.

21.4 Grammar Localization Structure

All keys and values in the Database Specification File (Section 9.1) are language independent. **On the server** the *propertiesName* key (page 33) of the Database Specification File contains the name of the so-called *Grammar Localization Structure*. To retrieve the structure, the system looks in the user database for a record in the *db_localize* table (see Section 18.15.1) whose *db* field is the name of the Grammar Localization Structure, and whose *lang* field is the requested target language. The *json* field will then contain the Grammar Localization Structure.

On the client the Grammar Localization Structure is available in a variable called *l10n*. The structure is described in TypeScript as the *Localization* interface in the file `ts/localization.ts`.

The Grammar Localization Structure is specified in JSON and contains the following key/value pairs:

Key	Value
dbdescription	A short description of the associated Emdros database.
dbcopyright	An HTML string containing copyright information for the associated Emdros database.
emdrosobject	A collection of key/value pairs containing the localized names for the Emdros object types and their features (see Section 21.4.1).
emdrostype	A collection of key/value pairs containing the localized names for the values in the Emdros enumeration types (see Section 21.4.2).
grammarfeature	A collection of key/value pairs giving the names of GrammarFeatures (see Section 21.4.3).
grammarmetafeature	A collection of key/value pairs giving the names of GrammarMetaFeatures (see Section 21.4.3).

(Continued...)

Key	Value
grammarsubfeature	A collection of key/value pairs giving the names of features within a GrammarMetaFeature (see Section 21.4.3).
grammargroup	A collection of key/value pairs giving the names of GrammarGroups (see Section 21.4.3).
universe	A collection of key/value pairs describing how to display book, chapter, and verse references (see Section 21.4.4).

21.4.1 The *emdrosobject* Key

The value of *emdrosobject* is a collection of key/value pairs containing the localized names for the Emdros object types and their features. As an example, Listing 21.1 shows a subset of the *emdrosobject* for English localization of the ETCBC4 database.

Listing 21.1: A sample emdrosobject value

```
1   "emdrosobject": {
2       "word": {
3           "_objname": "Word",
4           "vt": "Tense",
5           "sp": "Part of speech",
6           ... (Additional features omitted)
7       },
8       "phrase_atom": {
9           "_objname": "Phrase atom",
10          "det": "Determination",
11          "rela": "Relation",
12          ... (Additional feature omitted)
13      },
14      ... (Additional Emdros object types omitted)
15  }
```

Each key within the *emdrosobject* is the name of an Emdros object, so the above example gives information about the *word* and the *phrase_atom* Emdros objects.

Each key has a value which is a collection of key/value pairs. One of those keys is always *_objname*, and its value is the English name for the Emdros object; the remaining keys are features of the Emdros object, and their values are the English name for the feature.

So in the above example, lines 2-7 state that the English name of the Emdros object *word* is "Word", and that it has a feature called *vt* which in English should be rendered as "Tense". The *word* feature *sp* should be rendered "Part of speech" in English.

Similarly, lines 8-13 state that the English name of the Emdros object *phrase_atom* is "Phrase atom", and that it has a feature called *det* which in English should be rendered as "Determination". The *phrase_atom* feature *rela* should be rendered "Relation" in English.

If the value of the *_objname* key is long, it may not display well in the text area. An abbreviated version may be provided by appending the string *_abbrev* to the key in the *emdrosobject*. This can be seen in Listing 21.2, which is taken from the English localization of the *nestle1904* database.

Listing 21.2: An abbreviated emdrosobject name

```
1   "emdrosobject": {
2       "clause1": {
3           "_objname": "Clause level 1",
4           "typ": "Function"
5       },
6       "clause1_abbrev": {
7           "_objname": "Clause1"
8       },
```

```
9         ... (Additional Emdros object types omitted)
10    }
```

In this listing the Emdros object *clause1* is normally translated "Clause level 1", but in the text display area it is simply "Clause1", as shown in Figure 21.1. The *_objname* key is the only key under the abbreviated version.

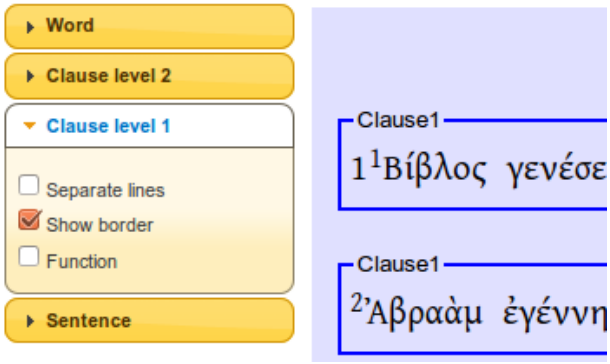


Figure 21.1: The term "Clause level 1" is used in the grammar selection box, but "Clause1" is used in the text area.

21.4.2 The *emdrostype* Key

The value of *emdrostype* is a collection of key/value pairs containing the localized names for the value of Emdros enumeration types. As an example, Listing 21.3 shows a subset of the *emdrostype* for English localization of the ETCBC4 database.

Listing 21.3: A sample emdrostype value

```
1     "emdrostype": {
2         "part_of_speech_t": {
3             "verb": "Verb",
4             "subs": "Noun",
5             "nmpr": "Proper noun",
6             ... (Additional values omitted)
7         },
8         "gender_t": {
9             "f": "#2 Feminine",
10            "m": "#1 Masculine",
11            "NA": "#3 None",
12            "unknown": "#4 Unknown"
13        },
14        ... (Additional enumeration types omitted)
15    }
```

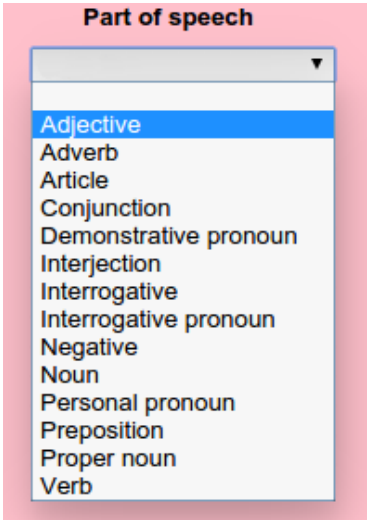
Each key within the *emdrostype* is the name of an Emdros enumeration type, so the above example gives information about the *part_of_speech_t* and the *gender_t* enumeration types.

Each key has a value which is a collection of key/value pairs, giving the names and the English translation of the values of the enumeration type.

In the above example, lines 2-7 indicate that the type *part_of_speech_t* has values such as *verb, subs,* and *nmpr*, whose English translations are "Verb", "Noun", and "Proper noun", respectively.

Lines 8-13 indicate that type *gender_t* has values *f, m, NA,* and *unknown*, whose English translations are "Feminine", "Masculine", "None", and "Unknown", respectively. The strings "#1", "#2" etc. indicate

the order in which these values should be sorted when presented to the user. Normally, the values would be sorted alphabetically thus:



But if the English translation starts with "#1", "#2" etc. these numbers indicate the sort order. So with the contents of *gender_t* given in Listing 21.3, genders are sorted thus:

If the translation of an enumeration value is long, it may not display well in the text area. Abbreviated versions may be provided by appending the string *_abbrev* to the key in the *emdrostype*. This can be seen in Listing 21.4, which is taken from the English localization of the *nestle1904* database.

Listing 21.4: Abbreviated emdrostype values

```
1   "emdrostype": {
2       "clause_type_t": {
3           "ADV": "Adverbial",
4           "CL": "Clause",
5           ... (Additional enumeration values omitted)
6       },
7       "clause_type_t_abbrev": {
8           "ADV": "ADV",
9           "CL": "CL",
10          ... (Additional enumeration values omitted)
11      },
12      ... (Additional enumeration types omitted)
13  }
```

In this listing the enumeration value *ADV* of type *clause_type_t* is normally translated "Adverbial", but in the text display area it is simply "ADV", as shown in Figure 21.2.

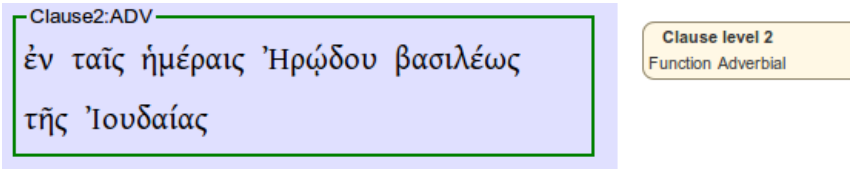


Figure 21.2: The term "Adverbial" is used in the grammar information box, but "ADV" is used in the text area.

21.4.3 The *grammarfeature, grammarmetafeature, grammarsubfeature,* and *grammargroup* Keys

Section 9.1.3 describes how the Database Specification File specifies how grammar information should be grouped in the grammar selection box and the grammar information box on the Bible OL webpage. Sections 9.1.4, 9.1.5, and 9.1.6 describe the GrammarFeature, GrammarMetaFeature, and GrammarGroup specifications and the GrammarSubFeature which is part of a GrammarMetaFeature.

In the Grammar Localization Structure, the *grammarfeature, grammarmetafeature, grammargroup,* and *grammarsubfeature* keys give the translation of these items, as detailed below.

21.4.3.1 *grammarfeature*

As an example, Listing 21.5 shows the *grammarfeature* for the English localization of the ETCBC4 database.

Listing 21.5: A sample grammarfeature value

```
"grammarfeature": {
    "word": {
        "text_translit": "Transliteration"
    }
}
```

Section 21.4.1 describes how the *emdrosobject* key is used to provide translations for Emdros object features. The *grammarfeature* in the above example gives an alternative translation. Normally the translation of a feature is taken from *emdrosobject*, but in the case of the grammar selection box and the grammar information box, the translation in *grammarfeature* is used, if present. If no translation is given in *grammarfeature*, the translation from *emdrosobject* is used.

21.4.3.2 *grammarmetafeature*

Listing 21.6 shows the *grammarmetafeature* for the English localization of the ETCBC4 database.

Listing 21.6: A sample grammarmetafeature value

```
1   "grammarmetafeature": {
2       "word": {
3           "pgn": "Person, gender, number",
4           "suffix_pgn": "Suffix: Person, gender, number"
5       }
6   }
```

In line 3 the GrammarMetaFeature *pgn* of the *word* object is given the English translation "Person, gender, number". (Listing 9.8 on page 43 specifies that the *word* object has a GrammarMetaFeature called *pgn*.)

21.4.3.3 *grammarsubfeature*

Listing 21.7 shows a subset of the *grammarsubfeature* for the English localization of the ETCBC4 database.

Listing 21.7: A sample grammarsubfeature value

```
"grammarsubfeature": {
    "word": {
        "ps": {
            "p1": "1",
            "NA": "-",
            "p2": "2",
            "p3": "3",
            "unknown": "?"
        },
        "gn": {
            "f": "F",
            "m": "M",
            "NA": "-",
            "unknown": "?"
        },
        "nu": {
            "du": "Du",
            "NA": "-",
            "pl": "Pl",
            "sg": "Sg",
            "unknown": "?"
        },
        ... (Additional features omitted)
    }
}
```

The example in Listing 9.8 on page 43 specifies that the *word* object has a GrammarMetaFeature called *pgn* which is made up of the GrammarSubFeatures *ps, gn,* and *nu.* The *grammarsubfeature* value in Listing 21.7 above specifies the English translation for these three GrammarSubFeatures. So if, for example, a word has features *ps=p2, gn=m,* and *nu=sg* (corresponding to second person, masculine, singular), the *pgn* GrammarMetaFeature should be rendered as "2MSg".

21.4.3.4 *grammargroup*

Listing 21.8 shows the *grammargroup* for the English localization of the ETCBC4 database.

Listing 21.8: A sample grammargroup value

```
"grammargroup": {
    "word": {
        "form_in_text": "Form in text",
        "lexeme": "Lexeme",
        "morphology": "Morphology"
    }
},
```

The Database Specification File for the ETCBC4 database defines three GrammarGroups for the *word* object with the names *form_in_text,*[1] *lexeme,* and *morphology.* The localization information above specifies the English names for these GrammarGroups. The two illustrations in Section 9.1.3 show these translations in the grammar selection box and the grammar information box.

[1]Shown in Listing 9.5 on page 42.

21.4.4 The *universe* Key

The value of *universe* is collection of key/value pairs describing how to display book, chapter, and verse references.

As an example, Listing 21.9 shows a subset of the *universe* for English localization of the ETCBC4 database.

Listing 21.9: A subset of the universe value

```
1   "universe": {
2       "book": {
3           "_label": "%s",
4           "Genesis": "Genesis",
5           "Exodus": "Exodus",
6           "Leviticus": "Leviticus",
7           "Numeri": "Numbers",
8           "Deuteronomium": "Deuteronomy",
9           "Josua": "Joshua",
10          "Judices": "Judges",
11          "Ruth": "Ruth",
12          "Samuel_I": "1 Samuel",
13          "Samuel_II": "2 Samuel",
14          ... (Additional books omitted)
15      },
16      "chapter": {
17          "_label": "Chapter %s"
18      },
19      "verse": {
20          "_label": "Verse %s"
21      },
22      "reference": {
23          "_label": "%s %d:%d",
24          "Genesis": "Gen",
25          "Exodus": "Ex",
26          ... (Additional book abbreviations omitted)
27      }
28  }
```

The *universeHierarchy* key of the Database Specification File (see page 34) defines the reference hierarchy of ETCBC4 as consisting of the Emdros object types *book*, *chapter*, and *verse*. The *universe* key in Listing 21.9 above defines how these three object types should be rendered:

The *_label* key presents the general format as a string where "%s" is to be replaced by the actual reference. So, when line 17 gives the *_label* key of *chapter* a value of "Chapter %s", it means that chapter 18 will be displayed as "Chapter 18".

For the *book* object, the *_label* key (line 3) is simply the string "%s", but additionally English translations of the book names used in ETCBC4 are given in lines 4-14.

The *reference* key specifies how a Bible reference should be written. The *_label* key here (line 23) specifies the format as "%s %d:%d". The "%s" will be replaced by the abbreviated book name, and the two "%d" strings will be replaced by the chapter and verse number, respecitively. Lines 24-26 contain abbreviations of the book names.

21.5 Grammar Localization Comments

In addition to the Grammar Localization Structures for various target languages, the *db_localize* table also contains comments for the different structures. These comments are found in entries where the *lang* field is "comment".

The grammar localization comments are structured exactly as the Grammar Localization Structures, but their contents are used in the translator's interface to guide in the translation of each item. Addi-

tionally, if a value starts with the string "f:textarea", the translator's interface will use a `<textarea>` HTML element for input. Otherwise, the translator's interface will use an `<input type="text">` HTML element.

21.6 Lexicon Localization

Sections 18.15.4, 18.15.5, 18.15.6, and 18.15.7 give information about how localized versions of the Hebrew, Aramaic, and Greek dictionaries are stored.

21.7 Importing and Exporting Translations

For a translator, it is often more convenient to work with translations offline. For this reason it is possble to import and export the various translation tables from and to a textual format.

21.7.1 The User Interface

Sections 18.15.2 and 18.15.3 describe how localization of the user interface is stored in the user database.

From the shell command line on the server the contents of the user interface tables can be exported thus:

```
php index.php translate if_db2php language-code destination-directory
```

Here *language-code* is, for example, "en" for English or "comment" to get the comment information (see Section 18.15.2). The exported files will be stored in a directory named *destination-directory/language-code*. (The directory *destination-directory* must exist, the directory *destination-directory/language-code* need not exist.)

To import the user interface tables from textual files, execute this command from the shell command line on the server:

```
php index.php translate if_php2db language-code source-directory
```

The imported files will be loaded from a directory named *source-directory*.

Note the difference in directory naming in the export and import commands. The language code is appended to the directory name on export, but not on import. To export the English texts to a directory called `abc` and import it again, use this sequence of commands:

```
php index.php translate if_db2php en abc
php index.php translate if_php2db en abc/en
```

The reason for this difference is that we want to enforce a particular directory structure when exporting, but we cannot rely on having that structure when importing.

21.7.2 Grammar Localization Structures

Section 21.4 describes how localization of grammar information is stored in the user database.

From the shell command line on the server the contents of all the grammar localization structures can be exported thus:

```
php index.php translate gram_db2prop destination-directory
```

This command will store the grammar localization structures for all Emdros databases and all languages in files in the directory *destination-directory* (which must exist). The data will be stored in "pretty" JSON format.

To import the grammar localization structures from textual files, place the JSON files in the directory `db/propterty_files` under the Bible OL main directory. Then execute this command from the shell command line on the server:

```
php index.php translate gram_prop2db
```

This command will read the files from the `db/propterty_files` directory and update the contents of the user database, where necessary. The command will print a list of tile files found plus information about whether the contents of the files caused the database to be updated.

Note the difference in directories used in the export and import commands. For export, the destination directory must be specified on the command line. For import, the files are always read from the `db/propterty_files` directory.

21.7.3 Lexicon

Sections 18.15.4, 18.15.5, 18.15.6, and 18.15.7 describe how localized versions of the dictionaries are stored.

From the shell command line on the server the contents of a lexicon can be exported thus:

```
php index.php translate download_lex source-language target-language
```

The *source-language* must be either "hebr", "aram", or "greek". The *destination-language* must be, for example, "en" for English. The lexicon will be written to the standard output as a comma-separated file, suitable for import into a spreadsheet program.

Note that a user with translator or administrator privileges can also download the lexicons from the Bible OL website by selecting the *Administration > Download lexicon* menu.

A lexicon can be imported from the shell command line on the server by executing this command:

```
php index.php translate import_lex source-language target-language CSV-file
```

The *source-language* and *destination-language* are described above. The *CSV-file* is the name of a comma-separated file containing the lexicon in the exact format generated by the export line above. This means that the imported CSV file must have the same number of fields and the same headings as the generated file.

Chapter 22

Plug-ins

Read this chapter if you want to modify Learning Journey or write a new plug-in.

CodeIgniter and Bible OL provide a limited mechanism for writing plug-ins (that is, extensions to the system).

PHP code for the plug-in must be stored in a directory hierarchy under `myapp/third_party/xxx`, where `xxx` is the name of the plug-in. The directory hierachy must contain the directories required by CodeIgniter, namely, `controllers`, `helpers`, `language`, `libraries`, `models`, and `views`.

The names of the controllers in the `controllers` directory must start with the string "Ctrl_XXX_", where "XXX" is the name of the plug-in in upper case letters. Similarly the models and views should have names that start with the string "Mod_XXX_" or "view_XXX_".

Localization of plug-ins is not quite streamlined yet. Localization information can be stored in the `language` directory.

To enable a plugin with the name "xxx", the following modifications must be made to the Bible OL code:

In the file `myapp/config/config.php` add the following line:

```
$config['xxx_enabled'] = true;
```

The value can be set to *true* or *false* depending on whether the plug-in is installed or not.

In the file `myapp/config/autoload.php` add the following lines:

```
if (config_item('xxx_enabled'))
    $autoload['packages'][] = APPPATH.'third_party/xxx';
```

In the file `myapp/config/routes.php` add the following lines:

```
if ($this->config->item('xxx_enabled')) {
    $route['(xxx)/(.+)'] = function ($name, $path) {
        $this->directory="../third_party/$name/controllers/"; // $this is the CI_Router object
        return "Ctrl_$path";
    };
}
```

This ensures that URLs such as, for example, `'https://bibleol.3bmoodle.dk/xxx/XXX_foobar` is sent to the controller at `myapp/third_party/xxx/controllers/Ctrl_XXX_foobar`. Thus, all URLs directed at plug-in "xxx" should start with the string "/xxx/" after the hostname.

If additional code needs to be added to Bible OL (for example, to add special menu items if a plug-in is available), the following can be added to the existing Bible OL code:

```
if ($this->config->item('xxx_enabled')) {
    ...(Code that uses the plug-in)
}
```

At present, the only available plug-in is *Learning Journey*.

22.1 The Learning Journey Plug-in

Bible OL collects information about the answers users give to exercises and how much time they spend on each exercise. The Learning Journey plug-in[1] accesses this information and provides statistics about each user's performance.

No further information about Learning Journey is given in this document.

[1] Developed by Judith Gottschalk.

Chapter 23

Complementary Websites

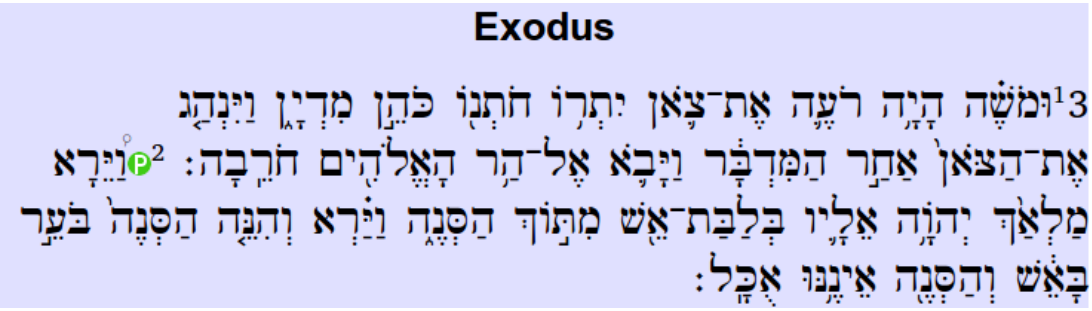
Read this chapter if you want to.

A few additional websites complement the function of Bible OL: The resource website and the SHEBANQ website.

23.1 The Resource Website

The resource web site is a collection of photos from the Middle East. Many of them relate to events and places described in the Bible. The photos have descriptive texts that contain Bible references. The URL of the resource website is `http://resources.3bmoodle.dk`.

Bible OL can use information from the resource website to add picture links to Bible passages. If a photo in the resource website refers to, for example, Exodus 3 : 2, and a user ticks the "Show link icons" checkbox when displaying Exodus chapter 3, a green "P" icon will appear in the text next to verse 2:



Clicking on the icon will cause the web browser to display the relevant photo. If there are more than one photo, the icon will be blue rather than green.

Bible OL gathers information about the photos on the resource web site by executing the command

```
php index.php pic2db
```

in a cron job. This command calls Bible OL's *Ctrl_pic2db* controller (in the file `myapp/controllers/ctrl_pic2db.php`), which requests information from the resource database.

By accessing the URL `http://resources.3bmoodle.dk/jsonrefs.php`, the *Ctrl_pic2db* controller receives a JSON object from the resources website containing information about the photos and the Bible verses to which they refer. Bible OL stores this information in the *bible_refs* table in the user database (see Section 18.10).

In addition to the pictures, the resource website may also provide URLs associated with various Bible verses. These URLs are configured in the resource website but are otherwise unrelated to the

functioning of that website. The URLs are intended to identify videos, documents, or other resources that may relevant for studying a particular verse.

Information about these URLs is also retrieved by the cron job above. By accessing the URL http://resources.3bmoodle.dk/jsonurls.php, the *Ctrl_pic2db* controller receives a JSON object from the resources website containing information about the URLs and the Bible verses to which they refer. Bible OL stores this information in the *bible_urls* table in the user database (see Section 18.11). Links to these urls are displayed as "V", "D", or "U" icons.

23.2 The SHEBANQ Website

SHEBANQ (System for HEBrew text: ANnotations for Queries and markup) is a website that uses the ETCBC4 database for displaying text and grammar information for the Hebrew Bible. The URL is http://shebanq.ancient-data.org.

When Bible OL displays a text from the Old Testament, an icon in the upper right corner of the text area provides a link to the same chapter at the SHEBANQ website. A similar link to Bible OL is found on the SHEBANQ website. Also, when a teacher is creating an exercise in Bible OL, they can import MQL queries from the SHEBANQ website (see Section 20.3.1).

Appendix A

ETCBC4 Details

The *ETCBC4* Emdros database contains the Hebrew and Aramaic text for the Old Testament.

The database comes from the *Eep Talstra Center for Bible and Computer* and is made available under a Creative Commons Attribution-NonCommercial 4.0 International License.[1] When describing the database, a text similar to this one should be used: "The database itself can be found through this persistent identifier: `urn:nbn:nl:ui:13-048i-71`." The identifier should be a hyperlink pointing to `http://www.persistent-identifier.nl/?identifier=urn:nbn:nl:ui:13-048i-71`.

Prior to using this database in Bible OL, I have added additional features to the information from its original creators. More details about this is given in Section A.1.20.

A.1 The *word* Object

The basic object type is the *word*. Each *word* corresponds to a single monad. The following sections give details about the features of the object.

Many of the features exist in several different encodings. The encodings are indicated by the name of the feature. A feature *XXX* may exists in these variants:

Feature name	Encoding
XXX	Transcribed alphabet
XXX_utf8	Native alphabet
XXX_translit	Transliterated alphabet
XXX_cons_utf8	Consonants only, native alphabet
XXX_nocant	Cantillation marks omitted, transcribed alphabet
XXX_nocant_utf8	Cantillation marks omitted, native alphabet
XXX_nopunct_translit	Punctuation omitted, transliterated alphabet

Except where otherwise noted, all features of the *word* object are string features.

In cases where a feature is an enumeration, the enumeration type may contain values that are not actually used. Such values are not listed in the following sections.

A.1.1 Features: g_word, g_suffix, and g_cons

These features relate to the visual appearance of a word.

The *g_word* features contain the actual text of the word. These variants exist:

- g_word
- g_word_utf8

[1] `http://creativecommons.org/licenses/by-nc/4.0`.

- g_word_translit
- g_word_cons_utf8
- g_word_nocant
- g_word_nocant_utf8
- g_word_nopunct_translit

The *g_word* features work together with the *g_suffix* features as described in Section 8.2.2. The *g_suffix* features exist in these variants:

- g_suffix
- g_suffix_utf8
- g_suffix_translit

The *g_suffix* features contain characters that follow a word. Possible suffixes are:

- An empty string
- A space
- ־
- ׀ followed by a space
- : followed by a space
- ׃ : followed by a space
- ס ׃ : followed by a space
- פ ׃ : followed by a space
- ס : followed by a space
- פ : followed by a space
- ס surrounded by spaces
- פ surrounded by spaces

Note: Do not confuse the *g_suffix* features with the features *suffix_gender, suffix_number,* and *suffix_person* which are described in Section A.1.13.

The *g_cons* features contain the consonants of the word. They are not used by Bible OL because their information is available by other means.

The *g_cons* features exist in these variants:

- g_cons
- g_cons_utf8

A.1.2 Features: g_lex, lex, and g_voc_lex

The "lexeme" or "lemma" of a word is the version of the word found in a dictionary. For example, the English word "mice" has the lexeme "mouse" because in a dictionary, the word is found under the entry "mouse".

ETCBC4 has three different lexeme features with different characteristics.

The *g_voc_lex* is the word commonly taken to be the lexeme of a Hebrew word. Except for verbs, this lexeme contains vowels. The *g_voc_lex* features exist in these variants:

- g_voc_lex
- g_voc_lex_utf8
- g_voc_lex_translit
- g_voc_lex_cons_utf8

The *g_lex* and *lex* features contain various other ways to write the lexeme. Compare, for example, these three lexemes for אֱלֹהִים:

Feature name	Content
g_voc_lex_utf8	אֱלֹהִים
g_lex_utf8	אֱלֹה
lex_utf8	אלהים

I have not studied all the differences between the three lexemes. Suffice it to say that *g_voc_lex_utf8* is the one normally shown to users, and *lex* is useful internally in the system because it only contains consonants and uses the transcribed alphabet.

The *g_lex* features exist in these variants:

- g_lex
- g_lex_utf8
- g_lex_cons_utf8

The *lex* features exist in these variants:

- lex
- lex_utf8
- lex_cons_utf8

A.1.3 Features: lexeme_occurrences and frequency_rank

ETCBC4 contains information about the frequency of various lexemes. The feature *lexeme_occurrences* is an integer feature containing the number of times this lexeme occurs in the Old Testament. The feature *frequency_rank* is an integer feature containing the rank of each lexeme: The most frequent lexeme has rank 1, the second most frequent lexeme has rank 2, etc.

These two features are counted separately for Hebrew and Aramaic parts of the Old Testament.

A.1.4 Features: pfm, g_pfm

The *pfm* feature describes the paradigmatic form of the preformative. The following values are possible:

- The empty string
- >
- absent[2]
- H
- J
- L
- M
- N
- n/a
- T=
- T

The *g_pfm* features contain the graphical representation of the preformative. These features exist in these variants:

- g_pfm
- g_pfm_utf8
- g_pfm_translit
- g_pfm_cons_utf8

A.1.5 Features: vbs, g_vbs

The *vbs* feature describes the paradigmatic form of the root formation morpheme. The following values are possible:

- >
- absent[3]

[2] This is the actual string “absent”.

[3] This is the actual string “absent”.

- C
- H
- HCT
- HT
- N
- n/a
- NT
- >T
- T

The *g_vbs* features contain the graphical representation of the root formation morpheme. These features exist in these variants:

- g_vbs
- g_vbs_utf8
- g_vbs_translit
- g_vbs_cons_utf8

A.1.6 Features: vbe, g_vbe

The *vbe* feature describes the paradigmatic form of the verbal ending. The following values are possible:

- The empty string
- H=
- H
- J
- JN
- N>
- N
- n/a
- NH
- NW
- T==
- T=
- T
- TJ
- TM
- TN
- TWN
- W
- WN

The *g_vbe* features contain the graphical representation of the verbal ending. These features exist in these variants:

- g_vbe
- g_vbe_utf8
- g_vbe_translit
- g_vbe_cons_utf8

A.1.7 Features: nme, g_nme

The *nme* feature describes the paradigmatic form of the nominal ending. The following values are possible:

- The empty string
- absent[4]
- H
- J=
- J
- JM=
- JM
- JN=
- JN
- N
- n/a
- T=
- T
- TJ
- TJM
- TJN
- W=
- W
- WT
- WTJ

The *g_nme* features contain the graphical representation of the nominal ending. These features exist in these variants:

- g_nme
- g_nme_utf8
- g_nme_translit
- g_nme_cons_utf8

A.1.8 Features: uvf, g_uvf

The *uvf* feature describes the paradigmatic form of the univalent final. The following values are possible:

- >
- absent[5]
- H
- J
- N
- W

The *g_uvf* features contain the graphical representation of the univalent final. These features exist in these variants:

- g_uvf
- g_uvf_utf8
- g_uvf_translit
- g_uvf_cons_utf8

[4] This is the actual string "absent".
[5] This is the actual string "absent".

A.1.9 Features: prs, g_prs

The *prs* feature describes the paradigmatic form of the pronominal suffix. The following values are possible:

- H
- H=
- HJ
- HM
- HN
- HW
- HWN
- J
- K
- K=
- KM
- KN
- KWN
- M
- MW
- n/a
- N
- N>
- NJ
- NW
- W

The *g_prs* features contain the graphical representation of the pronominal suffix. These features exist in these variants:

- g_prs
- g_prs_utf8
- g_prs_translit
- g_prs_cons_utf8

A.1.10 Feature: qere

The *qere* features contain the qere reading for the current word, if any.

The *qere* features exist in these variants:

- qere
- qere_utf8
- qere_translit

A.1.11 Feature: language

The *language* feature indicates if the word is Hebrew or Aramaic. It is an enumeration feature of type *language_t* whose value is either *Hebrew* or *Aramaic.*

A.1.12 Features: sp and pdp

The *sp* feature indicates the part of speech of the word; the *pdp* feature indicates the phrase dependent part of speech. Both are enumeration features of type *part_of_speech_t*. They can have these values:

Value	Meaning
adjv	Adjective
advb	Adverb
art	Article
conj	Conjunction
inrg	Interrogative
intj	Interjection
nega	Negative
nmpr	Proper noun
prde	Demonstrative pronoun
prep	Preposition
prin	Interrogative pronoun
prps	Personal pronoun
subs	Noun
verb	Verb

A.1.13 Features: ps, nu, gn, suffix_person, suffix_number, suffix_gender, prs_ps, prs_nu, prs_gn

The *ps, nu,* and *gn* features indicate the person, number, and gender, respectively, of the word. They are enumeration features of type *person_t, number_t,* and *gender_t* respectively.

The *suffix_person, suffix_number,* and *suffix_gender* features indicate the person, number, and gender of an optional suffix on the word. (Note: Do not confuse these features with the suffix features described in Section A.1.1.)

The *prs_ps, prs_nu,* and *prs_gn* features are identical to the *suffix_person, suffix_number,* and *suffix_gender* features, respectively.

The *person_t* enumeration has these values:

Value	Meaning
p1	First person
p2	Second person
p3	Third person
unknown	Unknown person
NA	Not applicable

The *number_t* enumeration has these values:

Value	Meaning
sg	Singular
du	Dual
pl	Plural
unknown	Unknown number
NA	Not applicable

The *gender_t* enumeration has these values:

Value	Meaning
f	Feminine
m	Masculine
unknown	Unknown number
NA	Not applicable

A.1.14 Feature: ls

The *ls* feature indicates the lexical set of the word. It is an enumeration feature of type *lexical_set_t* whose value is one of the following:

Value	Meaning
afad	Anaphoric adverb
card	Cardinal
cjad	Conjunctive adverb
focp	Focus particle
gntl	Gentilic
mult	Noun of multitude
nmcp	Copulative noun
nmdi	Distributive noun
none	None
ordn	Ordinal
padv	Potential adverb
ppre	Potential preposition
ques	Interrogative particle
quot	Quotation verb
vbcp	Copulative verb

A.1.15 Feature: vs

The *vs* feature indicates the verbal stem of the word. It is an enumeration feature of type *verbal_stem_t* whose value is one of the following:

Value	Meaning	Used in language
afel	Afel	Aramaic
etpa	Etpaal	Both
etpe	Etpeel	Aramaic
haf	Hafel	Aramaic
hif	Hifil	Hebrew
hit	Hitpael	Hebrew
hof	Hofal	Both
hotp	Hotpaal	Hebrew
hsht	Hishtafal	Both
htpa	Hitpaal	Aramaic
htpe	Hitpeel	Aramaic
htpo	Hitpoal	Hebrew
nif	Nifal	Hebrew
nit	Nitpael	Hebrew
pael	Pael	Aramaic
pasq	Passive Qal	Hebrew
peal	Peal	Aramaic
peil	Peil	Aramaic
piel	Piel	Hebrew
poal	Poal	Hebrew
poel	Poel	Hebrew
pual	Pual	Hebrew
qal	Qal	Hebrew
shaf	Shafel	Aramaic
tif	Tifal	Hebrew
NA	Not applicable	

A.1.16 Feature: vt

The *vt* feature indicates the verbal tense of the word. It is an enumeration feature of type *verbal_tense_t* whose value is one of the following:

Value	Meaning
coho	Cohortative
emim	Emphatic imperative
impf	Imperfect
impv	Imperative
infa	Infinitive absolute
infc	Infinitive construct
juss	Jussive
perf	Perfect
ptca	Participle
ptcp	Passive participle
wayq	Wayyiqtol
NA	Not applicable

A.1.17 Feature: st

The *st* feature indicates the state of the word. It is an enumeration feature of type *state_t* whose value is one of the following:

Value	Meaning
a	Absolute
c	Construct
e	Emphatic
NA	Not applicable

A.1.18 Feature: verb_class

The *verb_class* feature indicates the verb classes to which a word belongs. It is list of values of the enumeration type *verb_class_t* whose values are:

- analog_i_nun
- analog_i_waw
- four_consonants
- geminate
- hjh_xjh
- i_aleph
- i_guttural
- i_nun
- i_waw
- i_yod
- ii_guttural
- ii_waw
- ii_yod
- iii_aleph
- iii_guttural
- iii_he
- regular

A.1.19 Features: number, distributional_parent, functional_parent, lexeme_count

These features are currently not used by Bible OL and are not documented here.

A.1.20 The Origin of the Features

The following features of the *word* object were part of the ETCBC4 database as I received it from its creators:

distributional_parent	g_suffix	lex	ps
functional_parent	g_suffix_utf8	lexeme_count	qere
g_cons	g_uvf	lex_utf8	qere_utf8
g_cons_utf8	g_uvf_utf8	ls	sp
g_lex	g_vbe	nme	st
g_lex_utf8	g_vbe_utf8	nu	suffix_gender
gn	g_vbs	number	suffix_number
g_nme	g_vbs_utf8	pdp	suffix_person
g_nme_utf8	g_voc_lex	pfm	uvf
g_pfm	g_voc_lex_utf8	prs	vbe
g_pfm_utf8	g_word	prs_gn	vbs
g_prs	g_word_utf8	prs_nu	vs
g_prs_utf8	language	prs_ps	vt

The following word features were generated by the program *emdros_updater* which is available at https://github.com/EzerIT/ETCBC4BibleOL.

frequency_rank	g_uvf_cons_utf8	g_word_nocant
g_lex_cons_utf8	g_uvf_translit	g_word_nocant_utf8
g_nme_cons_utf8	g_vbe_cons_utf8	g_word_nopunct_translit
g_nme_translit	g_vbe_translit	g_word_translit
g_pfm_cons_utf8	g_vbs_cons_utf8	lex_cons_utf8
g_pfm_translit	g_vbs_translit	lexeme_occurrences
g_prs_cons_utf8	g_voc_lex_cons_utf8	qere_translit
g_prs_translit	g_voc_lex_translit	verb_class
g_suffix_translit	g_word_cons_utf8	

The *emdros_updater* and associated programs also performs a number of modifications to the database, including adding the verbal tenses *jussive, cohortative,* and *emphatic imperative*. See the file https://github.com/EzerIT/ETCBC4BibleOL/blob/master/README.md for details.

A.2 The Other Object Types

For information about the other object types in the ETCBC4 database, please consult the MQL code used for generating the database. The MQL code can be seen by running the *mqldump* program as described on page 27.

Appendix B

ETCBC4 Words Database

The Words Database (see Chapter 11) for the ETCBC4 Emdros database has the name `ETCBC4_words.db`. Its structure is defined by these SQL statements:

```
CREATE TABLE lexemes (id integer primary key, lex text);
CREATE TABLE lexsuf (id integer primary key, lexid integer, sufid integer);
CREATE TABLE lextext (id integer primary key, lexid integer, textid integer);
CREATE TABLE suffixes (id integer primary key, suffix blob, suffix_translit blot);
CREATE TABLE texts (id integer primary key, word blob, word_translit blob);

CREATE INDEX ixlexemes on lexemes(lex);
CREATE INDEX ixlexsuf on lexsuf(lexid);
CREATE INDEX ixlextext on lextext(lexid);
CREATE INDEX ixsuffixes on suffixes(suffix);
CREATE INDEX ixsuffixes2 on suffixes(suffix_translit);
CREATE INDEX ixtexts on texts(word);
CREATE INDEX ixtexts2 on texts(word_translit);
```

The *lexemes* table has an entry for every possible value of the *lex* feature in the Hebrew parts (that is, not the Aramaic parts) of the Old Testament. Each entry consists of an ID number and the *lex* value.

The *suffixes* table has an entry for every possible pair of values of the *g_prs_utf8* and *g_prs_translit* features in the Hebrew parts of the Old Testament. Each entry consists of an ID number and the values of the *g_prs_utf8* and the *g_prs_translit* features.

The *texts* table has an entry for every possible pair of values of the *text_nocant_utf8* and *text_nopunct_translit* features in the Hebrew parts of the Old Testament. Each entry consists of an ID number and the values of the *text_nocant_utf8* and the *text_nopunct_translit* features.

The *lexsuf* table combines entries in the *lexemes* table with entries in the *suffixes* table. Similarly, the *lextext* table combines entries in the *lexemes* table with entries in the *texts* table. Thus, for example, the SQL statement

```
SELECT lex,suffix FROM lexemes
    JOIN lexsuf ON lexsuf.lexid=lexemes.id
    JOIN suffixes ON lexsuf.sufid=suffixes.id;
```

will list all possible combinations of the *lex* and *g_prs_utf8* features.

B.1 The Origin of the ETCBC4 Words Database

The Words Database is generated by the program *emdros_updater* which is available at https://github.com/EzerIT/ETCBC4BibleOL.

Appendix C

ETCBC4 Hints Database

The Hints Database (see Chapter 12) for the ETCBC4 Emdros database has the name `ETCBC4_hints.db`. Its structure is defined by this SQL statement:

```
CREATE TABLE hints (self integer primary key, hint text);
```

The *self* column contains a monad (see Chapter 6) which identifies the ambiguous word within the ETCBC4 database. The *hint* column contains the hint string in the form *"feature=value"* or *"feature=value, feature=value"*, where *feature* and *value* must be localized before they are displayed to the user.

C.1 The Origin of the ETCBC4 Hints Database

The Hints Database is generated by the program *emdros_updater* which is available at https://github.com/EzerIT/ETCBC4BibleOL. The data was collected by Dr Oliver Glanz of Andrews University.

Appendix D

Nestle1904 Details

The *nestle1904* database is in the public domain and derives from the 1904 version of Nestle's Greek New Testament text.

D.1 The *word* Object

The basic object type is the *word*. Each *word* corresponds to a single monad. The following sections give details about the features of the object.

Except where otherwise noted, all features of the *word* object are string features. They contain Unicode characters in UTF-8 encoding.

D.1.1 Features: surface, normalized, raw_normalized

The *surface* feature contains the actual text of the word.

The *normalized* feature is an attempt at a "normalized" form of the word. "Normalized" here means:

a) Punctuation has been removed.
b) Most accents due to throwback clitics have been eliminated.
c) Any final grave accent has been made acute when not eliminated by (b).

Note that process (b) is not perfect. It only normalizes words which have more than one accent. A consequence of this is that clitics such as μου will not get the accent removed even when the accent is present (e.g., due to a throwback clitic that follows it). Thus the *normalized* feature is not totally reliable.

The *raw_normalized* feature is the *normalized* feature with non-letter characters removed and all characters converted to lower case characters without accents.

D.1.2 Features: lemma and raw_lemma

The "lexeme" or "lemma" of a word is the version of the word found in a dictionary. For example, the English word "mice" has the lexeme "mouse" because in a dictionary, the word is found under the entry "mouse".

The *lemma* feature contains the lemma of the word. The *raw_lemma* feature is the lemma with non-letter characters removed and all characters converted to lower case characters without accents.

Note that the lemma may contain extra characters, for example:

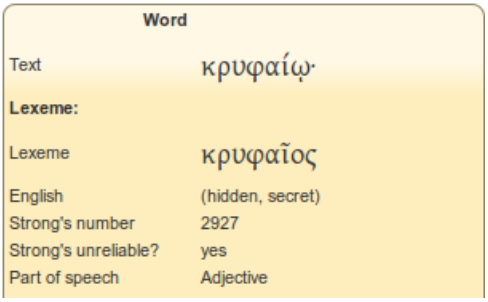
Lemma	Occurs in	Meaning
"βάτος (I)"	Luke 6 : 44	Thorn bush or bramble
"βάτος (II)"	Luke 16 : 6	"Bath," a liquid measure

In the *raw_lemma* feature, both of these are given as "βατος".

D.1.3 Features: strongs and strongs_unreliable

The *strongs* feature is an integer feature containing Strong's number for the lemma. The *strongs_unreliable* feature is a Boolean feature which is *true* if the indicated Strong's number is considered unreliable.

If a Strong's number is considered unreliable, Bible OL lists the corresponding gloss in parentheses. For example (from Matthew 6 : 18):



D.1.4 Feature: psp

The *psp* feature indicates the part of speech of the word. It is an enumeration features of type *psp_t* and can have these values:

- adjective
- adverb
- aramaic
- article
- cond
- conjunction
- correlative_or_interrogative_pronoun
- correlative_pronoun
- demonstrative_pronoun
- hebrew
- indefinite_pronoun
- interjection
- interrogative_pronoun
- letter_indeclinable
- noun
- noun_other_type_indeclinable
- numeral_indeclinable
- particle
- personal_pronoun
- possessive_pronoun
- preposition
- proper_noun_indeclinable
- reciprocal_pronoun
- reflexive_pronoun
- relative_pronoun
- verb
- NA (i.e., not applicable)

D.1.5 Features: person, number, gender, case, and possessor_number

The *person, number, gender,* and *case* features indicate the person, number, gender, and case of the word. They are enumeration features of type *person_t, number_t, gender_t,* and *case_t,* respectively.

For possessive pronouns the *number* feature indicates the number of the owned item, and the *possessor_number* feature indicates the number of the owning item. The *possessor_number* feature is an enumeration feature of type *number_t.*

The *person_t* enumeration has these values:

- first_person
- second_person
- third_person
- NA (i.e., not applicable)

The *number_t* enumeration has these values:

- singular
- plural
- NA (i.e., not applicable)

The *gender_t* enumeration has these values:

- masculine
- feminine
- neuter
- NA (i.e., not applicable)

The *case_t* enumeration has these values:

- nominative
- vocative
- genitive
- dative
- accusative
- NA (i.e., not applicable)

D.1.6 Features: tense, voice, and mood

The *tense, voice,* and *mood* features indicate the tense, voice, and mood of a verb. They are enumeration features of type *tense_t, voice_t,* and *mood_t,* respectively.

The *tense_t* enumeration has these values:

- present
- imperfect
- future
- second_future
- aorist
- second_aorist
- perfect
- second_perfect
- pluperfect
- second_pluperfect
- NA (i.e., not applicable)

The *voice_t* enumeration has these values:

- active
- middle

- passive
- middle_or_passive
- middle_deponent
- passive_deponent
- middle_or_passive_deponent
- impersonal_active
- NA (i.e., not applicable)

The *mood_t* enumeration has these values:

- indicative
- subjunctive
- optative
- imperative
- infinitive
- participle
- imperative_participle

D.1.7 Feature: suffix

The *suffix* feature indicates the meaning of a word suffix. It is an enumeration features of type *suffix_t* and can have these values:

- superlative
- comparative
- interrogative
- negative
- attic
- particle_attached
- crasis
- NA (i.e., not applicable)

D.1.8 Feature: ref

The *ref* feature indicates the Bible verse to which this word belongs. For example, all words in Luke 2 : 5 have the *ref* feature set to "Luke 2:5".

D.1.9 Features: form_tag and functional_tag

The *form_tag* and *functional_tag* are taken directly from the original CSV file on which this Emdros database is built. They are not used by Bible OL.

D.2 The *sentence* Object

The *sentence* object has no features. Its purpose is merely to group words.

D.3 The *clause1* and *clause2* Objects

The *clause1* and *clause2* objects have a single feature, *typ*.

Although Bible OL only provides three levels of the syntax trees (*sentence, clause1,* and *clause2*), the Syntax trees on which the database is based contain considerably more levels. Adding more levels to Bible OL would require a completely different way to present the syntax trees, and three levels are considered adequate for most cases.

D.3.1 Feature: typ

The *typ* feature indicates the function of the clause. It is an enumeration features of type *clause_type_t* and can have these values:

Value	Meaning
ADV	Adverbial function
CL	Clause
IO	Indirect object function
O	Object function
O2	Second object function
P	Predicate function
S	Subject function
V	Verbal function
VC	Verbal copula function

D.4 The Other Object Types

For information about the other object types in the nestle4 database, please consult the MQL code used for generating the database. The MQL code can be seen by running the *mqldump* program as described on page 27.

D.5 The Origin of the Data

The current Emdros database comes from three sources:

- A CSV file containing the text and grammar information, provided to me by Ulrik Sandborg-Petersen.
- A lexicon derived from Jeff Dodson's Public Domain lexicon of the Greek NT, provided to me by Ulrik Sandborg-Petersen.
- Syntax trees downloaded from `https://github.com/biblicalhumanities/greek-new-testament`.

The text and the lexicon are also available from `https://github.com/biblicalhumanities`.

The Emdros database has been created based on these sources using my program *nestle2mql*, which is currently not published or documented.

Index

Page numbers in italics are used to indicate the main references for an entry. A page number may appear both in italics and in upright letters if a term is used twice on a page.

.3et (file extension), 8, 49
3ET, 8

accent
    acute, 31
    circumflex, 31
    grave, 31
    Greek, 31
    monotonic, 31
    polytonic, 31
acute accent, *see* accent, acute
address, *see* email address
administrator, 13
AllowOverride, Apache directive, 14
alphabet
    native, *27*, 33, 114
    transcribed, *28*, 114
    transliterated, *28*, 33, 114
alphabet table, 83
Apache, 10, 11
    Configuration, 14
app ID, *see* Facebook, app ID
app secret, *see* Facebook, app secret
architecture, 20
ASCII, 28

base URL, 12
Bible OL, 9
Bible Online Learner, 9
bible_refs table, *85*, 112
bible_urls table, 85, 113
bol_user table, *see* user table
book, 28–30, 34, 66
Book Order File, *see* Database Book Order File
Bootstrap, 15, *19*, 24
browser, *see* web browser

C++, 10, 77
chapter, 28–30, 34, 66
"Check answer" button, 21
circumflex accent, *see* accent, circumflex
ckeditor, 15, 24
class table, 82
classexercise table, 84
clause, 6, 28, 29, 64
clause1, 30, 64, 130
clause2, 30, 64, 130
clause_atom, 28, 29
client, *20*, 59, 62, 95
client ID, *see* Google, client ID
client secret, *see* Google, client secret
clone, *11*, 15
CodeIgniter, 17, 24, *72*, 74
configuration JavaScript variable, *33*, 59, 95
Configuration TypeScript interface, 33
configure, 10
controller, 72–74
copyright, 5, 101
cron job, 80, 112
CSS, *18*, 25, 93

Database Book Order File, 32, 33, *47*
Database Description File, 32
Database Specification File, 32, *33*, 50, 59
Database Type Information File, 32, 33, *46*
database, user, *see* user database
db_localize table, *90*
Description File, *see* Database Description File
development, 14
dictionaries JavaScript variable, 60, 61, *62*, 95, 97
Dictionary PHP class, *62*, 62, 63, 75
Dictionary TypeScript class, 97
DictionaryIf TypeScript interface, *62*, 62, 97

display feature, *6*, 50, 53, 71, 88
DisplayMonadObject TypeScript class, 97
DisplayMultipleMonadObject TypeScript class, 97, 98
DisplaySingleMonadObject TypeScript class, 97
download Bible OL, 11
driver, MQL, *see* MQL driver
Dropbox, 11
dummy object, 99

editquiz (TypeScript program), *99*
Eep Talstra Center for Bible and Computer, 114
email address, 12, 73
email sender, 12, 73
Emdros, 10, 17, *22*, 27, 77
    enumeration, 37, 46, 47, 52, 56, 101, 103, 114
    feature, *22*, 36, 46, 50, 67, 102, 114
    ID_D, *23*, 66, 67, 70, 97, 98
    installation, 15
    object, *22*, 28, 30, 46, 102, 114, 127
emdros_updater, 124–126
emdrosobject, 101, *102*, 105
emdrostype, 101, *103*
enum2values, 47
enumeration, *see* Emdros, enumeration
`<enumfeature>`, 52
`<enumlistfeature>`, 52
ETCBC4, *27*, 27, *114*
EuroPLOT, 8
exercise, *5*, 60
exercise template, *see* quiz template
exercisedir table, 84
exerciseowner table, 85
exporting translations, *108*
ExtendedQuizFeatures PHP class, 70
ExtendedQuizFeatures TypeScript interface, 69, 70

Facebook
    app ID, 12, 78, 79
    app secret, 12, 79
    developers account, 12
    login, *see* login, Facebook
feature, *see* Emdros, feature
`<featurehandlers>`, 50
featuresetting, 36
“Finish” button, *see* “GRADE task” button
font, 24, 25, 75, 84
font table, 84
fork GitHub, 11, 14, 15

Git, 10, 11, 15
GitHub, 11, 15
Glanz, Oliver, 126
gmake, *see* make
Gnu make, *see* make
Google
    apps account, 12
    client ID, 12, 78, 79
    client secret, 12, 79
    login, *see* login, Google
Gottschalk, Judith, 111
“GRADE task” button, 61
“GRADE task” button, 21
Grammar Localization Structure, 90
grammar hierarchy, *6*, 64–66, 97
grammar information box, *6*, 7, 39, 40, 43–45, 59–61, 71, 83, 84, 96, 105, 106
grammar localization comments, *107*
Grammar Localization Structure, 33, 45, 50, 59, *101*
grammar selection box, *6*, 7, 39, 43–45, 53, 60, 61, 96–99, 105, 106
GrammarFeature, *42*, 101, 105
GrammarGroup, *44*, 102, 105, 106
GrammarMetaFeature, *43*, 101, 105
GrammarSubFeature, *43*, 102, 105, 106
grave accent, *see* accent, grave
Greek accent, *see* accent, Greek

half_verse, 28, 29
heb_urls table, 89
hierarchy, grammar, *see* grammar hierarchy
hints, 58
Hints Database, *58*, 126
hook, Git, 11
hosting, 10
.htaccess, 14
HTML, *18*, 95

icon, 26
ID_D, *see* Emdros, ID_D
importing translations, *108*
information box, *see* grammar information box
installation, 10
`<integerfeature>`, 51
internationalization, 100

JavaScript, 16, *18*, 20, 25, 95
jQuery, *18*, 25
jQuery UI, *18*, 24, 25
JSON, *18*, 32
    pretty file, *32*, 33
    ugly file, *32*, 33
json_pretty_print.php, 33
jstree, 15, 25, 60, 76

key/value pairs, 32
keyboard, virtual, *see* virtual keyboard
Kofoed, Jens Bruun, 8

l10n JavaScript variable, 59, 95, *101*
l10n_js JavaScript variable, 60, 95, *101*
language_comment table, *90*
language_LANG table, *90*
ldconfig, 11
Learning Journey, 111
lemma, *see* lexeme
Less, 16, 18, 25, *93*
lessc, 16, 93
    installation, 16
lexeme, *115*, 116, 127
lexicon localization, *108*
lexicon_Aramaic table, *91*
lexicon_Aramaic_LANG table, *91*
lexicon_greek table, *91*
lexicon_greek_LANG table, *92*
lexicon_Hebrew table, *91*
lexicon_Hebrew_LANG table, *91*
license, 5, 114, 127
Linux, 10, 14
localization, 89, 100
Localization TypeScript interface, 101
login
    Facebook, 12, 26, 73, 74, *77*, 82
    Google, 12, 26, 73, 74, *77*, 82
    local, 74, 77, 82
    OAuth2, *77*

Mac, 10, 14
make, 10, 93, 95
Makefile, 93, 95
maqaf, 29, 98
MatchedObject TypeScript interface, 62, 66, *67*
mayselect, 35, 36
md5, 13
md5sum, 14
migrations table, 89
model, 72, 73, 75
model-view-controller, *72*, 74
monad, *22*, 28, 30, 64, 70, 114, 126
MonadObject PHP class, 63, 75
MonadObject TypeScript interface, 62, *66*, 97
MonadPair PHP class, 63
MonadPair TypeScript interface, 62
MonadSet TypeScript interface, 62, 67, 97
monotonic accent, *see* accent, monotonic
MQL, 11, 17, *23*, 50, 54, 75, 76
    displaying in web page, 96
MQL driver, 12, 73, 76
    external, 76, *77*
    native, 76, *77*
mqldump, 27, 124, 131
multiple-choice, 36–38, 53, *56*, 71
MultipleMonadObject PHP class, 64
MultipleMonadObject TypeScript interface, 63, 64, 97
MVC, *see* model-view-controller
MySQL, 10, 11, 13

native alphabet, *see* alphabet, native
nestle1904, 27, *30*, 31, *127*
nestle2mql, 131
New Testament, 27, 127
nodejs, 14, 16
NORETRIEVE, 54
npm, 14

obj2feat, 46
objectSettings, 34, *35*, 36, 57
ol (TypeScript program), *95*, 95
ol.php, 12, 73, 76, 78, 79
Old Testament, 27, 48, 114
OlMatchedObject PHP class, 63, 66, 67
OlMonadSet PHP class, 63, 67
oxia, 31

passages, *6*, 50, 102, 107
    hierarchy, 34
password, 13
patriarch, *64*, 65, 99
perispomeni, 31
personal_font table, 84
PHP, 10, 14, *17*, 20, 72, 74
phrase, 6, 28, 29, 64
phrase_atom, 28, 29
PLOTLearner, 8
plug-in, 110
polytonic accent, *see* accent, polytonic
pretty JSON file, *see* JSON, pretty file
primary name, *32*, 33, 50
pseudofeature, *39*

`<qerefeature>`, 52
question, *6*, 6
question item, *6*, 6, 88, 89
question object, *see* quiz object
quick harvest, 77
quiz, *see* exercise
quiz object (a.k.a. sentence unit), *6*, 35, 50, 53, 54, 70
Quiz PHP class, *69*, 76

quiz template, *6*, 25, *49*, 60, 87
quizdata JavaScript variable, 60, 61, *69*, 95
QuizData TypeScript interface, 69
`<quizfeatures>`, *53*, 54
`<quizobjectselection>`, *53*, 54

`<rangeintegerfeature>`, 51
request feature, *6*, 38, 50, 53, 56, 71, 89
resource website, 34, 67, 74, 76, 85, *112*
RewriteBase, Apache directive, 14
RGraph, *19*, 25

salt, 12, 14, 73
Sandborg-Petersen, Ulrik, 8, 22, 131
"SAVE outcome" button, 21, 61
selection box, *see* grammar selection box
sender, *see* email sender
sentence, 6, 28, 30, 34, 64, 130
sentence unit, *see* quiz object
sentence_atom, 28
sentencegrammar, 39
`<sentenceselection>`, *50*, 54
server, *20*, 59, 62, 72, 74
SHEBANQ, 99, 113
"Show answer" button, 21
SingleMonadObject PHP class, 63
SingleMonadObject TypeScript interface, 63, 64, 66, 97
source code, 24
space (between words), 29, 97
Specification File, *see* Database Specification File
SQL, *17*, 73
SQLite3, 10, 14, 39, 56, 58
sta_displayfeature table, 88
sta_question table, 88
sta_quiz table, 70, *87*
sta_quiztemplate table, 87
sta_requestfeature table, 89
sta_universe table, 88
"Start quiz" button, 60
statistics, 46, 61, 74, 75, 86, 111
`<stringfeature>`, 51
subphrase, 28, 29
subset, 35, *45*
suffix, *29*, 30, 34, 115
system architecture, *see* architecture

template, *see* quiz template
tisch, *27*, 46
Tischendorf's Greek New Testament, 27
tonos, 31
tooltip, 59, 83
transcribed alphabet, *see* alphabet, transcribed
translations
  importing and exporting, *108*
transliterated alphabet, *see* alphabet, transliterated
tsc, 16, 95
  installation, 16
Type Information File, *see* Database Type Information File
typeinfo JavaScript variable, *46*, 60, 95
TypeInfo PHP class, 75
TypeInfo TypeScript interface, 46
TypeScript, 16, 18, 20, 25, *95*
  compiler, *see* tsc

Ubuntu, 14
ugly JSON file, *see* JSON, ugly file
Unicode, 27–29, 31, 127
universe, *see* passages
URL
  decoding in CodeIgniter, 72
  for displaying text, 59
  for running an exercise, 60
user database, 11, 12, 20, 39, 46, 70, 73, 74, 79, 80, *81*, 112, 113
user table, 13, 79, 80, *82*
userclass table, 83
userconfig table, 83
UTF-8, 27, 28, 127

varia, 31
verse, 28–30, 34, 66
view, 72, 73, 75
virtual keyboard, 26, 71
virtualkeyboard, 15, 26
"visual" feature, *27*, 34

web browser, 20, 59
web server, 10
Windows, 10, 14
Winther-Nielsen, Nicolai, 8
WIVU, *27*, 45, 46
word, 6, 28–30, 64, 114, 127
Words Database, *56*, 76, 125

XML, 49, 75

zocial, 15, 26