# Internationalization



## VIVO Language Support

Multiple language support can mean many things. When a VIVO site supports a language other than English, that support includes:

- Text that is displayed in the VIVO pages.
    - For example, menus, selections, prompts, tool-tips and plain text.
- Terms in the Ontology, which are frequently displayed as links or section headings.
    - Labels and descriptions of properties and classes
- Text in the data model.
    - For example, if a book title is available in both French and English, a French-speaking user sees the French title. If a title is available only in English, it is displayed, without regard to the user's preference in languages.

Languages can be selected in a variety of ways, depending on the installation parameters:

- A VIVO installer can configure VIVO to use one of the supported languages.
- Different users may see different languages, depending on the settings in their web browser.
- Different users may select a language from a list of available languages.

Language support in VIVO is being implemented in phases:

- Phase 1 includes read-only support of public pages:
    - Pages that are visible to users who are not logged in.
    - Also includes support of some administrative pages.
    - This is currently available.
- Phase 2 will also provide read-write support of profile pages:
    - Users will be able to edit language-specific data in profile pages.
- Phase 3 will support administrative pages
    - Creating user accounts, manipulating RDF data and other administrative functions.
- Phase 4 will support "back-end" pages.
    - Used to edit the ontology, or to do low-level editing on individual entities.

VIVO language files are available for English, Spanish, Brazilian Portuguese, and German. If you need support for another language, please inquire of the VIVO mailing lists, to see if another group is already developing the files you need.

## Adding a language to your VIVO site

### Adding language files to VIVO

VIVO is distributed with English as the only supported language. VIVO also includes a set of "pseudo-language" files, as a demonstration of how language support is implemented.

Additional language files are available in the Git repositories at https://github.com/vivo-project/Vitro-languages and https://github.com/vivo-project/VIVO-languages.

If the repository contains files for the language you want, in the VIVO release that you are using, you can just download those files and install them.

## Translating VIVO into your language

First, contact the VIVO development team: we would love to talk to you. We will tell you if anyone else is working on your language, and we will be happy to help with any questions you may have.

When you are ready to go ahead, you must determine the "locale" of your translation. Then you prepare translations of twenty-one files, as shown below.

## The locale

Your locale is an internationally recognized code that specifies the language you choose, and the region where it is spoken. For example, the locale string `fr_CA` is used for French as spoken in Canada, and `es_MX` is used for Spanish as spoken in Mexico. Recognized codes for languages and regions can be found by a simple Google search. Here is a list of locales that are recognized by the Java programming language. You may also use this definitive list of languages and regions, maintained by the Internet Assigned Numbers Authority.

The locale code will appear in the name of each file that you create. In the files that contain RDF data, the locale code will also appear at the end of each line.

When the locale code appears in file names, it contains an underscore (`en_US`). When it appears inside RDF data files, it contains a hyphen (`en-US`).

## The language files

You can get the Spanish or the English files from the VIVO and Vitro language repositories, to use as a template for your own files.

The example that follow assume that you are creating files for the Estonian language, as spoken in Estonia, with the locale `et_EE`.

### Text strings (.properties)

These files contain about 1500 words and phrases that appear in the VIVO web pages. The page templates contain more than just text – they contain programming logic and display specifiers.

These words and phrases have been removed from the page templates, so no programming knowledge is required to translate them.

There is one file for phrases used in Vitro, the core around which VIVO is built, and one file for phrases that are specific to VIVO. In the example, these two files are both called `all_et_EE.properties`.

**Example file names**

```
[Vitro]/webapp/languages/et_EE/i18n/all_et_EE.properties
[VIVO]/languages/et_EE/themes/wilma/i18n/all_et_EE.properties
```

**Example content**

```
minimum_image_dimensions = Minimaalne pildi mõõdud: {0} x {1} pikslit
cropping_caption = Profiilifoto näeb alloleval pildil.
```

### Freemarker Templates (.ftl)

Almost all of the text in the Freemarker templates is supplied by the text strings in the properties files. However, some Freemarker templates are essentially all text, and it seemed simpler to create a translation of the entire template. These include the `help` and `about` pages, the `Terms of Use` page, and the emails that are sent to new VIVO users.

**Example file names**

```
[Vitro]/webapp/languages/et_EE/templates/freemarker/search-help_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/termsOfUse_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-acctCreatedEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-acctCreatedExternalOnlyEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-confirmEmailChangedEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-firstTimeExternalEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-passwordCreatedEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-passwordResetCompleteEmail_et_EE.ftl
[Vitro]/webapp/languages/et_EE/templates/freemarker/userAccounts-passwordResetPendingEmail_et_EE.ftl
[VIVO]/languages/et_EE/templates/freemarker/aboutMapOfScience_et_EE.ftl
[VIVO]/languages/et_EE/templates/freemarker/aboutQrCodes_et_EE.ftl
[VIVO]/languages/et_EE/templates/freemarker/mapOfScienceTooltips_et_EE.ftl
```

**Example content**

```
<section id="terms" role="region">
    <h2>kasutustingimused</h2>

    <h3>Hoiatused</h3>

    <p>
        See ${termsOfUse.siteName} veebisait sisaldab materjali; teksti informatsiooni
        avaldamine tsitaadid, viited ja pildid ikka teie poolt ${termsOfUse.siteHost}
        ja erinevate kolmandatele isikutele, nii üksikisikute ja organisatsioonide,
        äri-ja muidu. Sel määral copyrightable Siin esitatud infot VIVO veebilehel ja
        kättesaadavaks Resource Description Framework (RDF) andmed alates VIVO at
        ${termsOfUse.siteHost} on mõeldud avalikuks kasutamiseks ja vaba levitamise
        tingimuste kohaselt
        <a href="http://creativecommons.org/licenses/by/3.0/"
                target="_blank" title="creative commons">
            Creative Commons CC-BY 3.0
        </a>
        litsentsi, mis võimaldab teil kopeerida, levitada, kuvada ja muuta derivaadid
        seda teavet teile anda laenu ${termsOfUse.siteHost}.
    </p>
</section>
```

## RDF data (.n3)

Data in the RDF models includes labels for the properties and classes, labels for property groups and class groups, labels for menu pages and more.

**Example file names**

```
[VIVO]/languages/et_EE/rdf/applicationMetadata/firsttime/classgroups_labels_et_EE.n3
[VIVO]/languages/et_EE/rdf/applicationMetadata/firsttime/propertygroups_labels_et_EE.n3
[VIVO]/languages/et_EE/rdf/display/everytime/PropertyConfig_et_EE.n3
[VIVO]/languages/et_EE/rdf/display/firsttime/aboutPage_et_EE.n3
[VIVO]/languages/et_EE/rdf/display/firsttime/menu_et_EE.n3
[VIVO]/languages/et_EE/rdf/tbox/firsttime/initialTBoxAnnotations_et_EE.n3
```

**Example content**

```
<http://vivoweb.org/ontology#vitroClassGrouppeople>
    <http://www.w3.org/2000/01/rdf-schema#label> "inimesed"@et-EE .
<http://vivoweb.org/ontology#vitroClassGrouppublications>
    <http://www.w3.org/2000/01/rdf-schema#label> "teadus"@et-EE .
<http://vivoweb.org/ontology#vitroClassGrouporganizations>
    <http://www.w3.org/2000/01/rdf-schema#label> "organisatsioonid"@et-EE .
<http://vivoweb.org/ontology#vitroClassGroupactivities>
    <http://www.w3.org/2000/01/rdf-schema#label> "tegevused"@et-EE .
```

### The selection image (.png, .jpeg, .gif)

If you allow the user to select a preferred language, you must supply an image for the user to click on. Typically, this image is of the flag of the country where the language is spoken.

**Example file names**

```
[VIVO]/languages/et_EE/themes/wilma/i18n/images/select_locale_et_EE.gif
```

**Example content**

## How can I contribute my language files to the VIVO community?

If you are planning to create a translation of VIVO, you should coordinate with the VIVO developers. When your files are ready, you can make them available to the development team in any way you choose. Note that the VIVO project will release your files under the Apache 2 License. They will require a Contributor Agreement stating that you agree to those terms.

# Adding language support to your local modifications

If you make changes to the VIVO code or templates, you may want to add language support to your changes. This is only necessary if your site supports multiple languages, or if you plan to contribute your code to the VIVO community.

## Language in the data model

The usual form of language support in RDF is to include multiple labels for a single individual, each with a language specifier.

In fact, any set of triples in the data model are considered to be equivalent if they differ only in that the objects are strings with different language specifiers. If language filtering is enabled, VIVO will display the value that matches the user's preferred locale. If no value exactly matches the locale, the closest match is displayed.

Consider these triples in the data:

```
<http://abc.edu/individual/subject1> <http://abc.edu/individual/property1> "coloring" .
<http://abc.edu/individual/subject1> <http://abc.edu/individual/property1> "colouring"@en-UK .
<http://abc.edu/individual/subject1> <http://abc.edu/individual/property1> "colorear"@es .
```

VIVO would display these values as follows:

| User's preferred locale | displayed text |
|---|---|
| en_UK | colouring |
| en_CA | colouring |
| es_MX | colorear |

| | |
|---|---|
| `fr_FR` | coloring |

VIVO has limited language support for editing values in the GUI. It is possible to edit language-specific labels for individuals. Language-specific values for other properties must be ingested into VIVO.

## Language support in VIVO pages

This section deals with the framework of the VIVO pages: the page titles, the prompts, the tool tips, the error messages; everything that doesn't come from the data model. These pieces of text are not stored in RDF, so we need a different mechanism for managing them.

The mechanism we use is based on the Java language's built-in framework for Internationalization. You can find more information in the Java tutorials for resource bundles and properties files.

"Internationalization" is frequently abbreviated as "I18n", because the word is so long that there are 18 letters between the first "I" and the last "n".

In the I18n framework, displayed text strings are not embedded in the Java classes or in the Freemarker template. Instead, each piece of text is assigned a "key" and the code will ask the framework to provide the text string that is associated with that code. The framework has access to sets of properties files, one set for each supported language, and it will use the appropriate set to get the correct strings.

For example, suppose that we have:

- The text that will appear in an HTML link, used to cancel the current operation, with the key `cancel_link`.
- The title of a page used to upload an image, with the key `upload_image_page_title`.
- The text of a prompt message, telling users how big an image must be, with the key `minimum_image_dimensions`.

The default properties file might show the English language versions of these properties, like this:

**Excerpt from all.properties**

```
cancel_link = Cancel
upload_image_page_title = Upload image
minimum_image_dimensions = Minimum image dimensions: {0} x {1} pixels
```

Notice that the actual image dimensions are not part of the text string. Instead, placeholders are used to show where the dimensions will appear when they are supplied. This allows us to specify the language-dependent parts of a message in the properties file, while waiting to specify the language-independent parts at run time.

A Spanish language properties file might show the Spanish versions of these properties in a similar manner:

**Excerpt from all_es.properties**

```
cancel_link = Cancelar
upload_image_page_title = Subir foto
minimum_image_dimensions = Dimensiones mínimas de imagen: {0} x {1} pixels
```

To use these strings in Java code, start with the I18n class, and the key to the string. Supply values as needed to replace any placeholders in the message.

**Using I18n strings from Java code**

```
protected String getTitle(String siteName, VitroRequest vreq) {
    return I18n.text(vreq, "upload_image_page_title");
}

private String getPrompt(HttpServletRequest req, int width, int height) {
    return I18n.text(req, "minimum_image_dimensions", width, height);
}
```

Similarly, using text strings in a Freemarker template begins with the `i18n()` method.

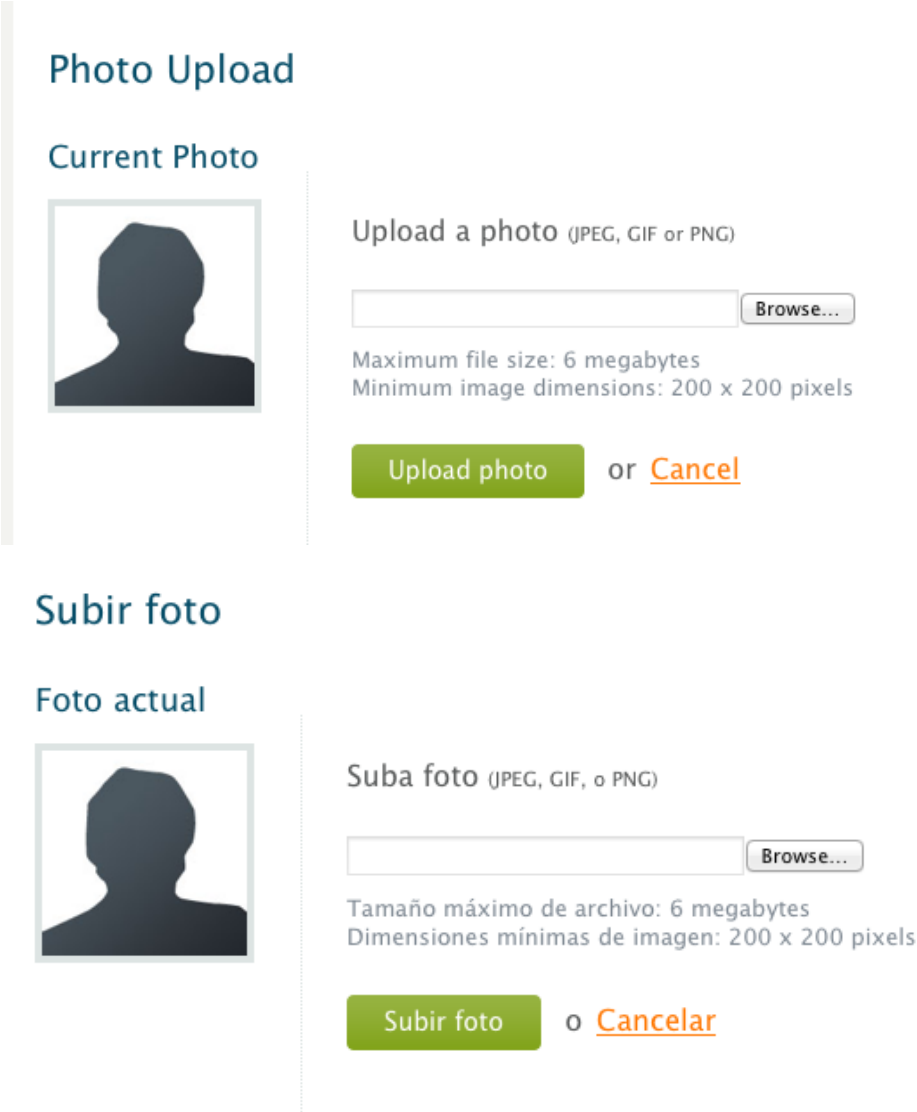
**Using I18n strings in a Freemarker template**

```
<#assign text_strings = i18n() >

<a href="../cancel" >
    ${text_strings.cancel_link}
</a>

<p class="note">
    ${text_strings.minimum_image_dimensions(width, height)}
</p>
```

Here is the appearance of the page in question, in English and in Spanish:




## Structure of the properties files

The properties files that hold text strings are based on the Java I18n framework for resource bundles. Here is a tutorial on resource bundles.

Most text strings will be simple, as shown previously. However, the syntax for expressing text strings is very powerful, and can become complex. As an example, take this text string that handles both singular and plural:

**A complex text string**

```
deleted_accounts = Deleted {0} {0, choice, 0#accounts |1#account |1<accounts}.
```

The text strings are processed by the Java I18n framework for message formats. Here is a tutorial on message formats. Full details can be found in the description of the MessageFormat class.

### Local extension: application vs. theme

The Java I18n framework expects all properties files to be in one location. In VIVO, this has been extended to look in two locations for text strings. First, it looks for properties files in the current theme directory. Then, it looks in the main application area. This means that you don't need to include all of the basic text strings in your theme. But you can still add or override strings in your theme.

If your VIVO theme is named "frodo", then your text strings (using the default bundle name) would be in

- *webapp*/themes/frodo/i18n/all.properties
- *webapp*/i18n/all.properties

If you specify a complex locale for VIVO, this search pattern becomes longer. For example, if your user has chosen Canadian French as his language /country combination, then these files (if they exist) will be searched for text strings:

- *webapp*/themes/frodo/i18n/all_fr_CA.properties
- *webapp*/i18n/all_fr_CA.properties
- *webapp*/themes/frodo/i18n/all_fr.properties
- *webapp*/i18n/all_fr.properties
- *webapp*/themes/frodo/i18n/all.properties
- *webapp*/i18n/all.properties

When VIVO finds a text string in one of these files, it uses that value, and will not search the remaining files.

## Language in Freemarker page templates

Here is some example code from `page-home.ftl`

**Excerpt from page-home.ftl**

```
<section id="search-home" role="region">
    <h3>${i18n().intro_searchvivo} <span class="search-filter-selected">filteredSearch</span></h3>
    <fieldset>
        <legend>${i18n().search_form}</legend>
        <form id="search-homepage" action="${urls.search}" name="search-home" role="search" method="post" >
            <div id="search-home-field">
                <input type="text" name="querytext" class="search-homepage" value="" autocapitalize="off" />
                <input type="submit" value="${i18n().search_button}" class="search" />
                <input type="hidden" name="classgroup"  value="" autocapitalize="off" />
            </div>
            <a class="filter-search filter-default" href="#" title="${i18n().intro_filtersearch}">
                <span class="displace">${i18n().intro_filtersearch}</span>
            </a>
            <ul id="filter-search-nav">
                <li><a class="active" href="">${i18n().all_capitalized}</a></li>
                <@lh.allClassGroupNames vClassGroups! />
            </ul>
        </form>
    </fieldset>
</section> <!-- #search-home -->
```

This code lays out all of the formatting and markup, but the actual strings of text are retrieved from the property files, depending on the current language and locale. Here are the English-language strings used by this code:

**English properties used in the example**

```
intro_searchvivo = Search VIVO
search_form = Search form
search_button = Search
intro_filtersearch = Filter search
all_capitalized = All
```

### Language-specific templates

Most Freemaker templates are constructed like the one above; the text is merged with the markup at runtime. In most cases, this results in lower maintenance efforts, since the markup can be re-structured without affecting the text that is displayed.

In some cases, however, the template is predominantly made up of text, with very little markup. In these cases, it is simpler to rewrite the entire template in the chosen language.

The Freemarker framework has anticipated this. When a template is requested, Freemarker will first look for a language-specific version of the template that matches the current locale. So, if the current locale is `es_MX`, and a request is made for `termsOfUse.ftl`, Freemarker will look for these template files:

| **Search order for `termsOfUse.ftl`**<br><br>**Current locale is `es_MX`** |
|---|
| `termsOfUse_es_MX.ftl` |
| `termsOfUse_es.ftl` |
| `termsOfUse.ftl` |

## Language in Java code

Java code has access to the same language properties that Freemarker accesses. Here is an example of using a language-specific string in Java code:

**Excerpt from UserAccountsAddPageStrategy.java**

```
  FreemarkerEmailMessage email = FreemarkerEmailFactory.createNewMessage(vreq);
  email.addRecipient(TO, page.getAddedAccount().getEmailAddress());
  email.setSubject(i18n.text("account_created_subject", getSiteName()));
```

The properties files contain this line:

**English language properties used in the example**

```
 account_created_subject = Your {0} account has been created.
```

Note how the name of the VIVO site is passed as a parameter to the text message.

## Language in JSPs

Up through VIVO release 1.7, no attempt has been made to add language support to JSPs.

## Language in JavaScript files

There is no convenient way to access the `i18n` framework from JavaScript files. One workaround is to assign values to JavaScript variables in the Freemarker template, and then access those values from the JavaScript.

For example, the template can contain this:

**Excerpt from page-home.ftl**

```
    <script>
        var i18nStrings = {
            countriesAndRegions: '${i18n().countries_and_regions}',
            statesString: '${i18n().map_states_string}',
    </script>
```

And the script can contain this:

**Excerpt from homePageMaps.js**

```
        if ( area == "global" ) {
            text = " " + i18nStrings.countriesAndRegions;
        }
        else if ( area == "country" ) {
            text = " " + i18nStrings.statesString;
        }
```

# Tools you can use

## i18nChecker

This is a set of Ruby scripts that are distributed with VIVO, in the `utilities/languageSupport/i18nChecker` directory. Use them to scan your language properties files and your freemarker templates. The scripts look for common errors in the files.

### Scanning language properties files:

- Warn if a specialized file has no default version.
- Warn about duplicate keys, keys with empty values.
- Warn about keys that do not appear in the default version.
- If the "complete" flag is set,
    - Warn if the default version is not found.
    - Warn about missing keys, compared to the default version.

### Scanning Freemarker templates:

- Warn about visible text that contains other than blank space or Freemarker expressions.
- Visible text is:
    - Anything that is not inside a tag and not between <script> tags
    - title="" attributes on any tags
    - alert="" attributes on <img> tags
    - alt=""  attributes on <img> tags
    - value="" attributes on <input> tags with submit attributes