Internationalization

Children Pages

Summary of this Page

VIVO Language Support

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.
- Text from terms in the Ontology, which are frequently displayed as links or section headings. Text includes labels and annotations of properties and classes.
- Text values stored in the data. 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 administrator 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.

VIVO language files are available for English (U.S. and Canadian), Spanish, Brazilian Portuguese, French (Canadian) and German. If you need support for another language, please inquire of the VIVO mailing lists, to see if another group has the files you need.

Adding an existing language to your VIVO site

In this step by step guide we will use the German language files as an example. Be sure to use the theme 'wilma' or 'tenderfoot' for this to work without issues.

- Edit the `vivo_home_dir/config/runtime.properties` file in your VIVO home directory:
    - uncomment/add `RDFService.languageFilter = true`
    - uncomment/add `languages.selectableLocales = en_US, de_DE`
- Restart the tomcat
- You should now be able to select your installed language (in this case German) in the header of your VIVO site

For more details, see Enabling Interface Languages in VIVO as an Administrator.

Building VIVO and Vitro language repositories from source (for developers)

- Clone the VIVO and Vitro repositories to your local machine.
- Build VIVO from the VIVO project folder using `mvn install -o -s installer/my-settings.xml` (Note the -o flag, this forces Maven to use the language projects from your local repository instead of downloading from a remote repository)
- Edit the `vivo_home_dir/config/runtime.properties` file in your VIVO home directory:
    - uncomment/add `RDFService.languageFilter = true`

    - uncomment/add `languages.selectableLocales = en_US, de_DE`
- Restart the tomcat
- You should now be able to select your installed language (in this case German) in the header of your VIVO site

Creating new language files for your language

First, contact the VIVO development team. We would love to talk to you. We will be happy to help with any questions you may have and introduce you to others who may be working on the same language as you are.

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 the terms in the agreement.

Translating VIVO into your language involves determining a locale, and preparing files as discussed 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 US English (home/src/main/resources/rdf/i18n/en_US) files from the VIVO and Vitro among the vivo-project repositories (https://github.com/vivo-project), to use as a template for your own files.

The process simply consists of making a directory inside VIVO and Vitro directories home/src/main/resources/rdf/i18n/, for instance et_EE (for Estonian), and copying there the complete file hierarchy from home/src/main/resources/rdf/i18n/en_US. In all copied ttl, n3, nt files, the @en-US language tags should be replaced with @ee-EE, and associated labels should be translated from English to Estonian.

In the process of initializing the files for a new language. You will encounter the following types of file:

User interface labels

These files contain about 1500 words and phrases that appear in the VIVO/Vitro web pages.

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

They appear at different level in the application (the <locale> might be for instance en_US) :

- in Vitro - home/src/main/resources/rdf/i18n/<locale>/interface-i18n/firsttime/vitro_UiLabel.ttl
- in VIVO - home/src/main/resources/rdf/i18n/<locale>/interface-i18n/firsttime/vivo_UiLabel.ttl
    - There are also files for themes:
        - home/src/main/resources/rdf/i18n/<locale>/interface-i18n/firsttime/vivo_UiLabel_wilma.ttl and
        - home/src/main/resources/rdf/i18n/<locale>/interface-i18n/firsttime/vivo_UiLabel_tenderfoot.ttl

The structure of those files is defined in the Vitro UI label vocabulary (in Vitro - home/src/main/resources/rdf/tbox/firsttime/UILabelsVocabulary.ttl).

The application will look for an entry starting with the activated theme (like tenderfoot or wilma), then VIVO and lastly Vitro. Of course, the selected locale (UI language) will be taken into account.

NOTE: VIVO/Vitro 1.13 was based on property files located in VIVO-languages and Vitro-languages repositories. Although, those repositories have been archived, and property files have been replaced with ttl files and moved to VIVO and Vitro home directory in VIVO/Vitro 1.14.0 release, due to back compatibility VIVO/Vitro 1.14 also supports using property files. It means, instead of previously listed files, customers can add vivo_all_<locale>.property and vitro_all_<locale>.property into the webapp/src/main/webapp/i18n directory.

RDF data (.n3, .nt)

Data in the RDF models include labels for the properties and classes, labels for property groups and class groups, labels for menu pages and more. Here is the list of directories where one will have to create required rdf files:

- [VIVO]/home/src/main/resources/rdf/i18n/<locale>/applicationMetadata/firsttime
- [VIVO]/home/src/main/resources/rdf/i18n/<locale>/display/firsttime
- [VIVO]/home/src/main/resources/rdf/i18n/<locale>/tbox/firsttime
- [Vitro]/home/src/main/resources/rdf/i18n/<locale>/display/firsttime
- [Vitro]/home/src/main/resources/rdf/i18n/<locale>/tbox/firsttime

In each case, the delivered file in English has a corresponding file with the same name but in a different directory structure (locale tag is different). for instance:

File names (Estonian)

```
[VIVO]/home/src/main/resources/rdf/i18n/et_EE/applicationMetadata/firsttime/classgroups_labels.n3
```

In each file, labels specify text to be used by VIVO. Each label should be translated and affixed with the appropriate locale tag. See below:

Some classgroups_labels (Estonian)

```
<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 .
```

How VIVO supports languages

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

Language support in VIVO pages

VIVO uses the Java language's built-in framework for Internationalization based on triplets preserved in the graph base.

"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 key. 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_photo`.
- 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 ttl files, like this:

Excerpt from /home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vitro_UiLabel.ttl

```
prop-data:cancel_link.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Cancel"@en-US ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "cancel_link" .

prop-data:upload_photo.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Upload a photo"@en-US ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "upload_photo" .

prop-data:minimum_image_dimensions.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Minimum image dimensions: {0} x {1} pixels"@en-US ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "minimum_image_dimensions" .
```

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 /home/src/main/resources/rdf/i18n/es/interface-i18n/firsttime/vitro_UiLabel.ttl

```
prop-data:cancel_link.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Cancelar"@es ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "cancel_link" .

prop-data:upload_photo.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Suba foto"@es ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "upload_photo" .

prop-data:minimum_image_dimensions.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Dimensiones mínimas de imagen: {0} x {1} pixels"@es ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "minimum_image_dimensions" .
```

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.

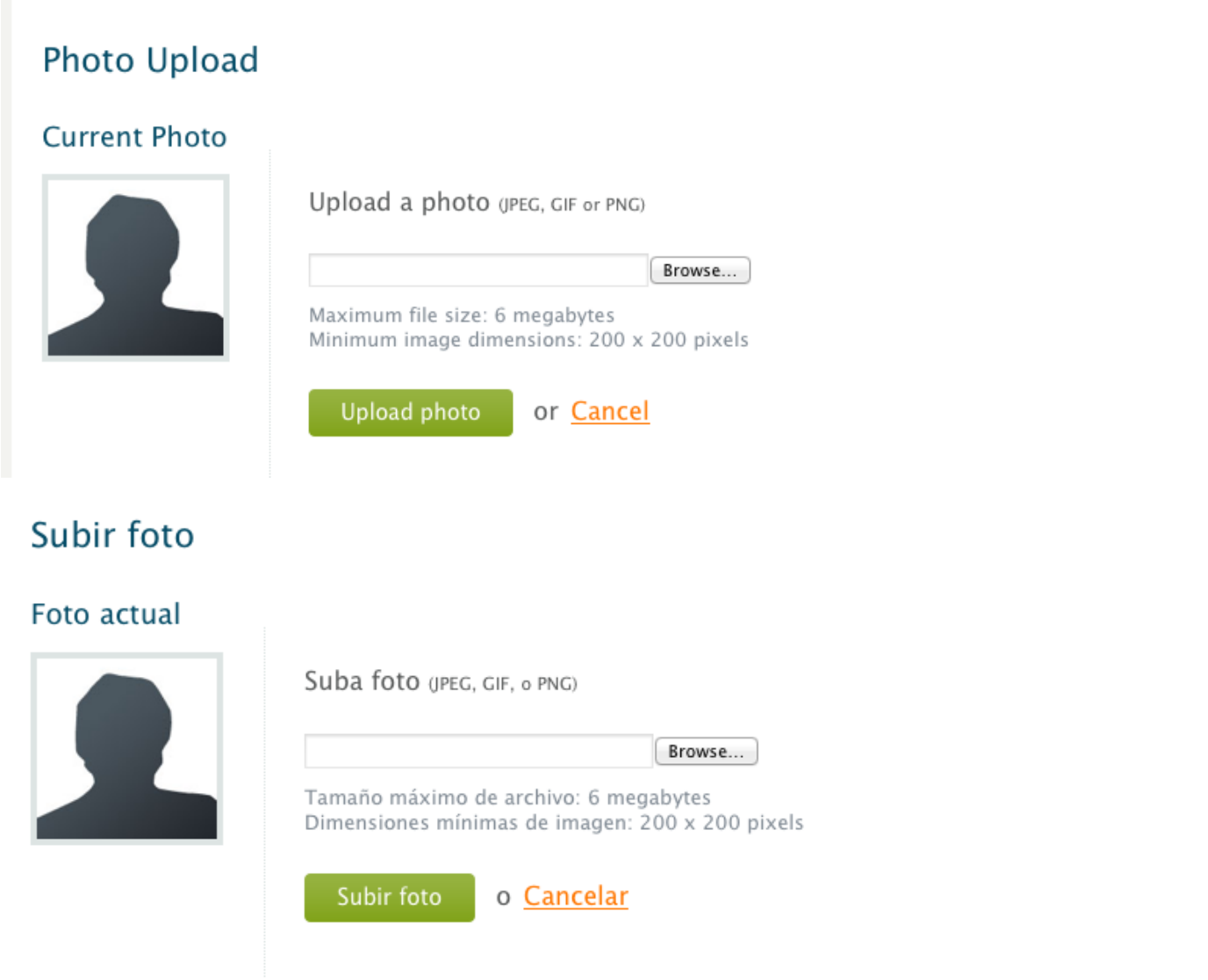
Using I18n strings in a Freemarker template

```
<#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

```
prop-data:deleted_accounts.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Deleted {0} {0, choice, 0#accounts|1#account|1<accounts}."@en-US ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "deleted_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 ui labels translations to be in one location. In VIVO, this has been extended to look in three locations for text strings. First, it looks for ui label translation files in the current theme (for instance, in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel_wilma.ttl). Then, it looks in the main VIVO UI labels file (in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel.ttl), and at the end in Vitro UI labels file (in Vitro - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vitro_UiLabel_wilma.ttl). 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 UI text strings would be in

- in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel_frodo.ttl
- in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel.ttl
- in Vitro - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vitro_UiLabel.ttl

NOTE: Please use properly hasApp and hasTheme data properties in ttl files, as well as language tags.

If you specify more than one locale for VIVO, this search pattern becomes longer. For example, if your user has chosen UQAM Canadian French as his language/country/private-use subtags combination (fr_CA_x_uqam), then these files (if they exist) will be searched for text strings:

- in VIVO - home/src/main/resources/rdf/i18n/fr_CA_x_uqam/interface-i18n/firsttime/vivo_UiLabel_frodo.ttl
- in VIVO - home/src/main/resources/rdf/i18n/fr_CA_x_uqam/interface-i18n/firsttime/vivo_UiLabel.ttl
- in Vitro - home/src/main/resources/rdf/i18n/fr_CA_x_uqam/interface-i18n/firsttime/vitro_UiLabel.ttl
- in VIVO - home/src/main/resources/rdf/i18n/fr_CA/interface-i18n/firsttime/vivo_UiLabel_frodo.ttl
- in VIVO - home/src/main/resources/rdf/i18n/fr_CA/interface-i18n/firsttime/vivo_UiLabel.ttl
- in Vitro - home/src/main/resources/rdf/i18n/fr_CA/interface-i18n/firsttime/vitro_UiLabel.ttl
- in VIVO - home/src/main/resources/rdf/i18n/fr/interface-i18n/firsttime/vivo_UiLabel_frodo.ttl
- in VIVO - home/src/main/resources/rdf/i18n/fr/interface-i18n/firsttime/vivo_UiLabel.ttl
- in Vitro - home/src/main/resources/rdf/i18n/fr/interface-i18n/firsttime/vitro_UiLabel.ttl
- in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel_frodo.ttl
- in VIVO - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vivo_UiLabel.ttl
- in Vitro - home/src/main/resources/rdf/i18n/en_US/interface-i18n/firsttime/vitro_UiLabel.ttl

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 ttl files, depending on the current language and locale.

Language-specific templates

Language-specific templates have been completely removed starting from the VIVO/Vitro 1.14.0 release.

All Freemaker templates are constructed like the one above; the text is merged with the markup at runtime.

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 ttl files contain these lines:

English language properties used in the example

```
prop-data:account_created_subject.Vitro
    rdf:type         owl:NamedIndividual ;
    rdf:type         prop:PropertyKey ;
    rdfs:label       "Your {0} account has been created."@en-US ;
    prop:hasApp      "Vitro" ;
    prop:hasKey      "account_created_subject" .
```

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

Language in JSPs

Up through VIVO release 1.14, no attempt has been made to add language support to JSPs.

Language in JavaScript files

To access string properties in JavaScript called from a template, assign the properties to 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;
        }
```