Internationalization - I18N

Support of multiple languages, also known as the standard *i18n* ('i' + 18 letters in 'nternationalizatio' + 'n'), is a Servoy feature that enables developing solutions that:

- Do not require additional forms or elements to display more than one default language.
- Can display multiple languages based on locale references.

This feature includes language keys for every Servoy menu item as well as all Servoy system information, warning and error dialogs. In addition, it also includes a complete set of locale message text for Servoy i18n system language keys in German, Dutch (Netherlands) and Italian.

Servoy provides i18n functions as part of the built-in Servoy Library, which means that one can set/get i18n message keys and values programmatically as part of a Servoy method (script).

This chapter gives a detailed view of the Servoy i18n feature and how to use it programmatically. See also the Internationalization chapter on the Developer User Guide.

In This Chapter

Solutions Using Multiple Languages

To enable a solution to support more than one language, the following actions need to be performed:

1. Assign or create an i18n table to store multiple language keys.
2. Create the i18n language keys that will be used or edit any of the language keys included in Servoy.

Note that only the i18n table set on the main solution is taken into account. All i18n messages of the solution, including its modules, will be stored into that table. It is not possible to use a different i18n table in a module than the one used in the main solution.

The ideal time to start with i18n is at the beginning of building a solution, so the developer can enter the i18n keys when building forms and methods. It is also possible to enter i18n after finishing the solution by using the Externalize Strings feature.

Definition

Inside Servoy Developer the message files are stored as property files in the workspace, under **resources** > **messages**. At runtime though, they are stored in the database. Thus, at some point before deploying to the server, the data in the workspace needs to be transferred to the database table. This can be done either at solution export, by checking the **Export i18n data** checkbox in the **Export Wizard**, or during design by right clicking the **I18N files** node under **Resources** in **Solution Explorer**, and choosing **Write to DB** option from the context menu.

On the server, the messages are loaded in the i18n table which was specified in the solution settings. The table needs to have the following columns:

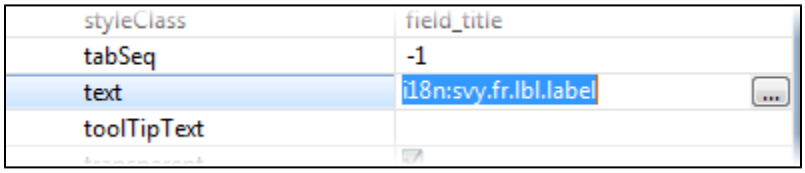
- tableName_id (primary key, integer)
- message_key (i18n key, string)
- message_language (i18n language, string)
- message_value (i18n value, string)

Usage

On Servoy elements, it is possible to use a text reference, which is a key linked to a messages file. Which is replaced at runtime with the localized text from the messages file.

Labels, Buttons, Tabpanels, FieldTitles

In design, i18n can be set for such items in their `text` or `titleText` property.



On the text property, select the **browse** button. This will open a dialog. The first tab is to enter text, but in the second tab the i18nkey can be selected. When selecting a key, it will replace the text with i18n: + the keyname.
An example of an element text reference is: `i18n:hello_world`, which is looked up in the messages file and produces the text in a locale like 'hello world' in English or 'hola mundo' in Spanish.

Dialogs, Calculations, Methods

To use i18n in scripting, use the function:

```
i18n.getMessage("i18n-key")
```

The i18n key has to be provided as a string.

Example This is an example of how to use the i18n function

```
var message = i18n.getMessage("servoy.general.clickOk");
```

The method will return the value of the language selected in the client. If the selected language has no entry, it will return the default value/reference text: 'Click OK to continue'.

It is also possible to use dynamic values. There can be an array provided with the values that have to be replaced. To use dynamic values, the i18n value should contain tags like {0}, {1}, ... , {n}. The tags will be replaced by the values provided in the same order.

```
var company_name = "Servoy";
var amount = 15
var type = "developers";
var message = i18n.getMessage("servoy.license.registered",[company_name, amount, type]);
```

For example, if the key `servoy.license.registered` has the value 'Registerd to {0} with {1} {2}' the outcome will be 'Registered to Servoy with 15 developers'.

When using i18n in a dialog, depending on the language of the user, the 'Yes' button can be 'Si', 'Ja' , 'Oui', 'Hai', etc.
To solve this, get the translation in a variable and use that to check which button is clicked.

Example This is an example of how to use i18n in dialogs

```
function question() {
	var yes = i18n.getI18NMessage("servoy.lbl.yes");
	var no = i18n.getI18NMessage("servoy.lbl.no");
	var cancel = i18n.getI18NMessage("servoy.lbl.cancel");
	var answer = plugins.dialogs.showQuestionDialog("i18n:servoy.lbl.title", "i18n:servoy.lbl.message", yes,
no, cancel);
	if(answer == yes) {
		application.output("yes is pressed");
		//execute code.
	}
}
```

Note that in dialogs it is not necessary to use the function `i18n.getI18Nmessage()` but only `i18n:key` is enough.

I18N Key Names

The developer is free to create and use their own i18n key names. Nevertheless, it is a good idea to have a naming convention for this.

In general, there are two types of i18n keys:

- short texts - which one could call labels
- long texts - like in dialogs, for example.

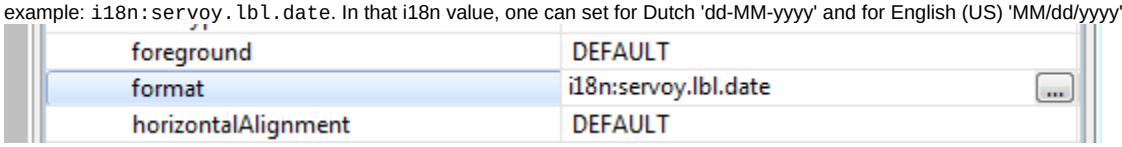
Example This is an example of how to use naming convention

Sample naming convention:
Labels : lbl.ok
Dialogs: dlg.ok

Or using the company name as prefix:
Labels: servoy.lbl.ok
Dialogs: servoy.dlg.ok

Date Format

In case of a date format, one can use for example: 'dd-MM-yyyy'. This would be a nice formatted date in the Netherlands, but it might not in other countries. That's why it is a good idea to use i18n instead of a hard coded format. Instead of setting the string in the format, do set an i18n key in the format. For example: `i18n:servoy.lbl.date`. In that i18n value, one can set for Dutch 'dd-MM-yyyy' and for English (US) 'MM/dd/yyyy'.



Overwriting i18n

An i18n message can be overwritten with the function `i18n.setI18NMessage()`.

Example When using i18n keys for date formats, let the users enter in the system how they prefer their date separator. Then when the user starts the system, use `i18n.setI18NMessage()` to replace the default separator with the user-defined separator. For example, in the i18n message the separator used is '/', but the user wants a '-' to separate the date. In the `onOpen` method the following code can be entered:

```
i18n.setI18NMessage('servoy.lbl.date', utils.stringReplace(i18n.getI18NMessage('servoy.lbl.date'),'/','-'))
```

Separators

Formatting defaults need to be specified. In the English(US) format, the dot ('.') is decimal separator (and comma is 1000 separator). Decimal values within code need to be specified in English format as well. This will automatically be translated to the end users format when they run the code.

Example In Dutch, for instance, the separator for decimals is a ','. In the `format` property of a field, the English format will be entered for a number (for example '#.00'). When a user with the locale 'Nederlands(Nederland)' uses the solution and enters '23' into the field, it will display '23,00'.

I18N Configuration

Problem Markers

Servoy provides support in Developer to enable problem markers on non-externalized strings found in scripting. By default, these type of problems is ignored, but this can be changed from **Preferences** page.

To enable the Externalized strings problem markers, go to **Window** > **Preferences** > **JavaScript** > **Errors/Warnings** > **Externalized strings** and set the problem severity level to `warning`, `error`, or `info`.

These problem markers can be suppressed at function level by using the `@SuppressWarnings` JavaDoc annotation with `nls` type, or at individual line level by using the `//$NON-NLS-<n>$` comment at the end of the line that needs to have warnings suppressed.

Servoy provides a Quick Fix option for the Externalized strings problem markers, with the options of adding the `@SuppressWarnings` annotation at function level, inserting the `//$NON-NLS-<n>$` tag at the end of the individual code line, or opening the **Externalize strings** wizard.

Save Action

Servoy offers a clean up option for automatically removing unnecessary Non-nls comments from code on saving a JavaScript file.

To enable this option, go to: **Window** > **Preferences** > **JavaScript** > **Editor** > **Externalize Strings** and check the **Remove unused $NON-NLS$ tags on save** option.

Localized Formatting

Servoy supports locale formats on numbers, integers, and dates.

There are four levels of setting the locale format:

1. Java Virtual Machine (JVM) level - setting the locale `language`, `country` and (optional) `variant` Java arguments. This will determine the formats used by the Servoy Clients.
   Since multiple countries may speak the same language, but the number/date formats are different from one country to another, the `country` argument is needed to ensure precision to the formats used by the application.
   Example This is an example of setting the locales for Germany, with specific for Macintosh platforms.

   ```
   -Duser.language=de -Duser.country=DE -Duser.variant=MAC
   ```

   A number format used by Servoy will therefore be: 123.456,789
   See more information on Oracle's page Internationalization: Understanding Locale in the Java Platform.
2. Servoy Application Server level - setting the Locale Settings on the **Servoy Server Home** page; the formats set will be applied to all fields that do not have a format set at table or field level.
   2.1. The Smart Client's locale formats can be also set via **Edit** > **Preferences** > **Locale**; the formats set in this way will be applied on top of the Admin Page settings.
3. table level - setting the **Default format** under **Details** tab on a column, in the Table Editor; the format set will be applied to all fields having the column as data provider, that do not have a format set on field level.
4. field level - setting the `format` property of a field in **Properties** view.

Timezones

In JavaScript (and Java), dates are always timestamps with the number of milliseconds which have passed since January 1st 1970 in UTC. Only when a date is displayed or converted to a string, the conversion to a certain timezone is done.

Example
If one calls `application.getTimeStamp()` at 8pm in Colombo, Sri Lanka and at 8pm in LA, California USA, two different timestamp values will be received. On the other hand, if two developers, one located in LA and one in Colombo would execute the code at exactly the same time, they would get exactly the same timestamp value, regardless of the timezone they are in.

The Servoy Application Server can be forced to run in a certain timezone, by including the following JVM argument in command line:

```
-Duser.timezone=US/Eastern
```

A complete table with the supported values of the timezone property can be found on IBM's WebSphere Application Server page.

Servoy has 2 modes for working with Timezones. Switching between these two modes can be done by toggling the `servoy.use.client.timezone` property on the Servoy Application Server's **Servoy Server Home** page, under **Locale Settings**.

Property set to true:

A datetime entered will be presented exactly the same in each client, regardless of the timezone of the client

Example: Any client, regardless of which timezone they are in, sees a datetime entered as 8pm as 8pm. Servoy makes corrections under the hood to achieve this.

Property set to false:

Servoy performs no calculation and the dates are automatically converted by Java to match the user's timezone.

Example: A client in UTC + 5 enters 10pm. The server runs in UTC, so stores 5pm. And a client in UTC - 2 sees 3pm. Since databases do not store a timezone with a date, everything will be mixed up when the database is started in another timezone.

The first scenario (having client timezone property set to true) is the preferred way of operating, because this gives the most consistent outcome.