Rigi Solution editor

Version 0.7 (21 April 2020)

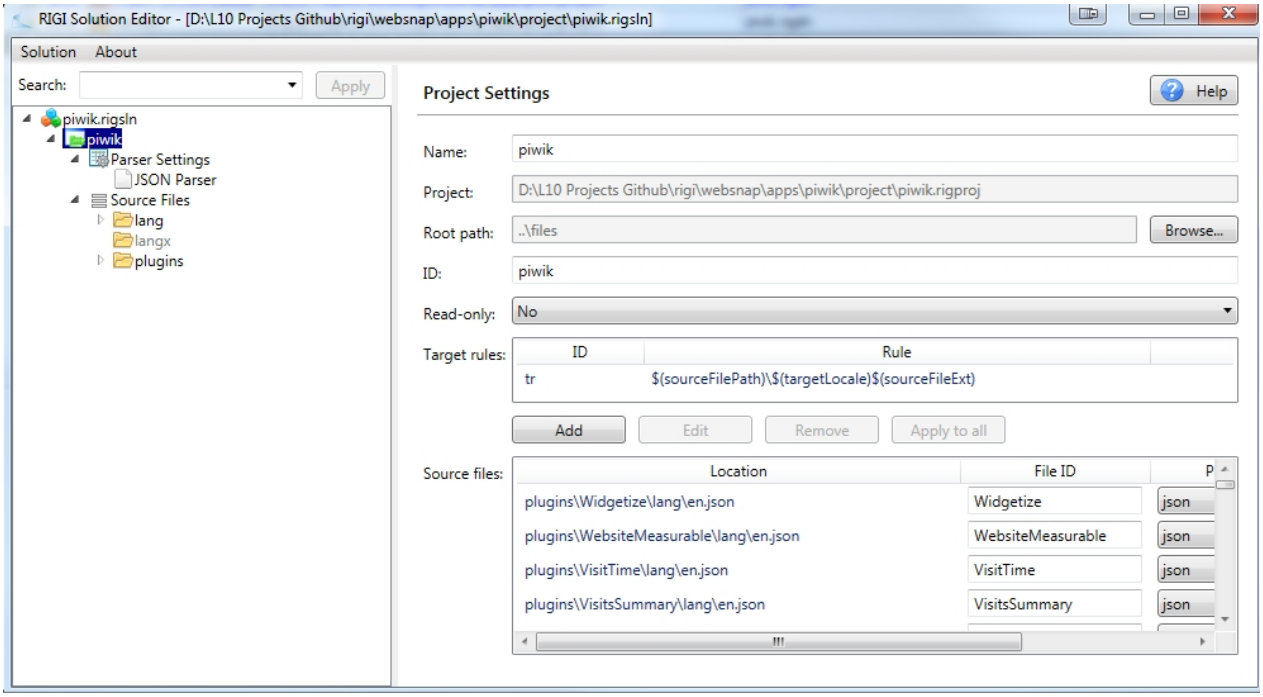
Rigi.io - Localization Platform

Table of contents

What is the Solution Editor? 3
Demo application 4
Create a solution 6
Create an empty solution 7
Advanced solution settings 8
Enter solution settings 9
Add a project 11
Define a project 12
Enter project settings 13
Enter target rules 15
Add source file 18
Mass adding source files 19
Parser settings 21
Delete source file 22
File formats 24
Solution file (.rigsln) 25
Project file (.rigproj) 27
Advanced 29
Mapping of variables in target rules 30
Screenshot size fixation 32
Parser SDK 33

What is the Solution Editor?

The Solution Editor is a Windows tool that helps the user to create a Rigi solution.



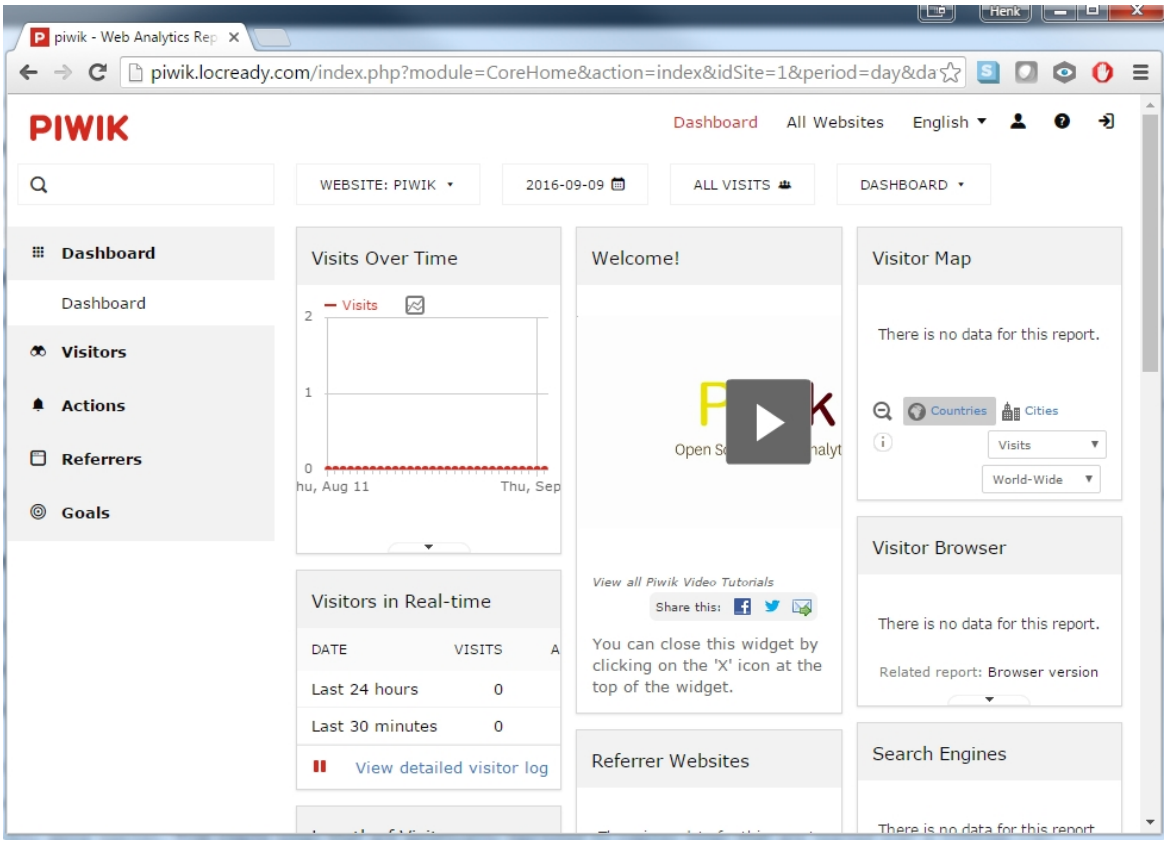
A Rigi solution contains one or more Rigi projects. Each Rigi project contains source files that must be localized into various languages. Per resource file the parser and target rule are defined.

Connectors to localization tools, such as the Rigi addin for SDL Passolo, use the solution definition to create projects and generate target files.

Demo application

We will demonstrate how the Solution Editor works by setting up a project for our Piwik demo application.

Piwik is an open source dynamic web based application. The visible content in the browser is constructed by server and client side technology.



All translatable texts for this application are defined in 49 resource files. All files are in the json file format. Each string has a unique string identifier and may optionally contain variables (e.g. %s).

The English source file `.\files\lang\en.json` has the following contents:

```
{
    "General": {
        "AbandonedCarts": "Abandoned Carts",
        "AboutPiwikX": "About Piwik %s",
        "Action": "Action",
... }
}
```

The goal is to localize this file in various languages. The German file shall have file name `.\files\lang\de.json` and the following contents:

```
{
"General": {
"AbandonedCarts": "Verlassene Warenkörbe",
```

```
"AboutPiwikX": "Über Piwik %s",
"Action": "Aktion",
... }
}
```

Create a solution

Use the Solution Editor to define the required attributes to localize the resources of an application.

This is done in a number of steps.

1. Create an empty solution.
2. Enter solution settings.
3. Add a project.

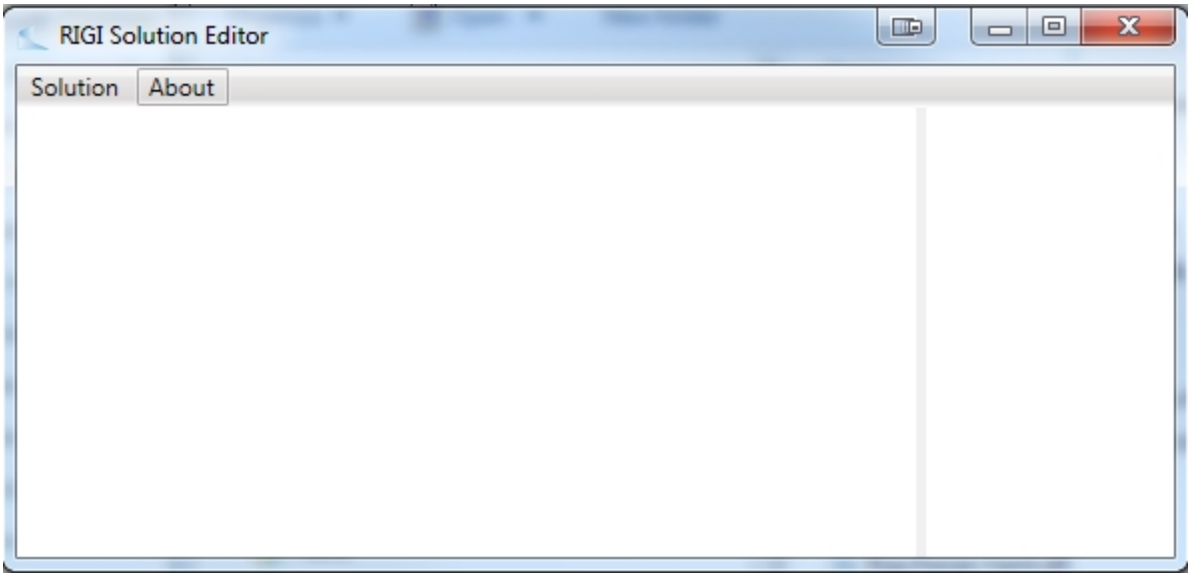
The Solution Editor can be started in the following ways:

- Stand alone (run the .exe)
- From localization tool SDL Passolo: `Tools > Rigi > Create Solution...`

Create an empty solution

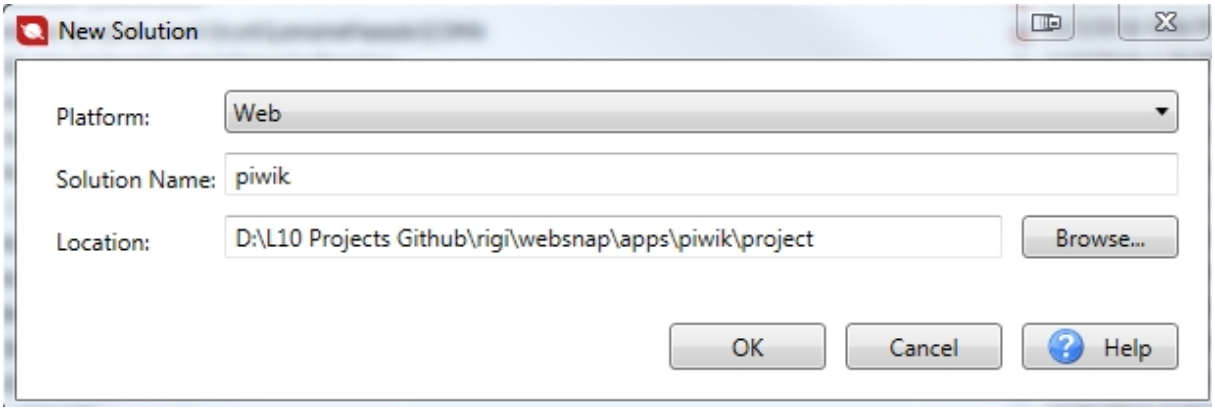
A Rigi project has a solution at root-level.

1. Start the Solution Editor.



2. Select Solution > New

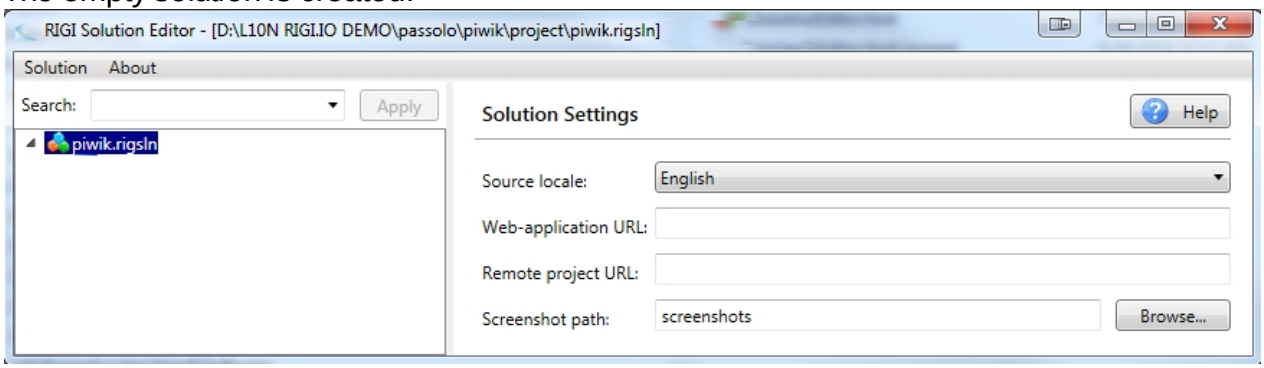
   A New Solution dialog appears.



Attribute	Description
Platform	The application type (e.g. Web and WPF). Each application type has specific settings. For example, the Web-application specifies a URL to the web-application and the WPF-application a path to the executable. You can change these settings at a later moment in time via the Advanced solution settings.
Solution name	Name of the solution.
Location	Location where the solution shall be stored.

3. Click OK.

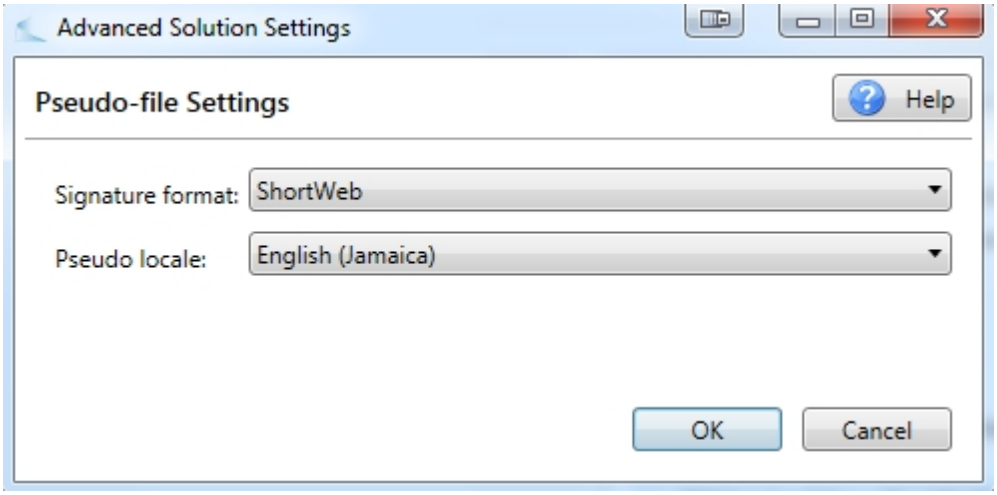
The empty solution is created.



Advanced solution settings

The application type (e.g. Web and WPF) determine the default signature format and pseudo locale.

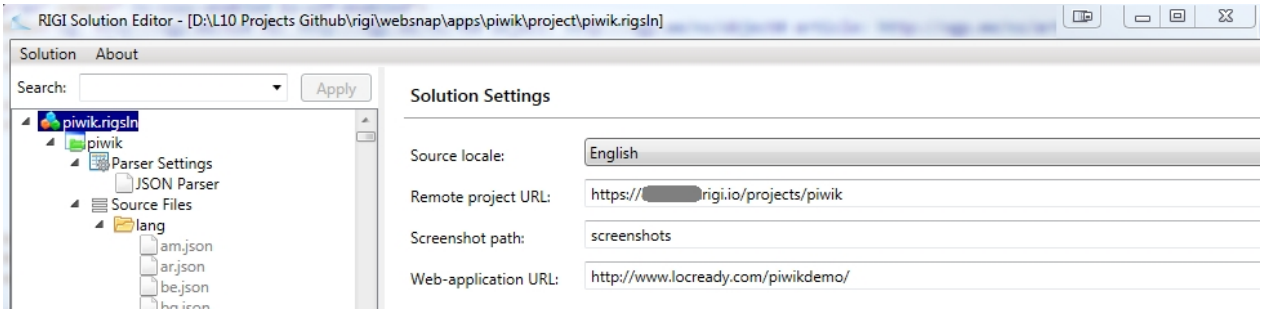
1. Select Solutions > Advanced Settings... to see and override the settings.



- The signature format indicates which characters Rigi uses to encode the identifiers.
- The Pseudo locale indicates the locale that is used for localization purposes. This locale must be loaded when Rigi screenshot files are genererated using Websnap and also when the web application is localized in live mode.

Enter solution settings

Define the solution settings.



1. Enter the solution settings that apply to all platforms.

Attribute	Description
Source locale	The source language of the resource files.
Remote project URL	The URL of the project on the Rigi server that hosts the screenshots, e.g. `xyz.rigi.io/projects/<projectcode>`. • Keep this field empty if your screenshots are not hosted on rigi.io. • The server assigns a *<projectcode>* to each project (e.g. 68gf28a519789dfb56123c4). You can find the project URL in the project settings on the Rigi server.
Screenshot path	The folder where the screenshots will be stored. This is a relative path with respect to the location of the .rigsln file.

Enter the platform specific settings:

Web:

Attribute	Description
Web-application URL	The URL of the staging application. The staging application has the ability to load the pseudo language, so that Rigi can determine which strings are shown on the UI. Keep this field empty if the URL is unknown for the moment.

WPF:

Attribute	Description
Executable path	File path and name of the WPF executable. Websnap will start this application.

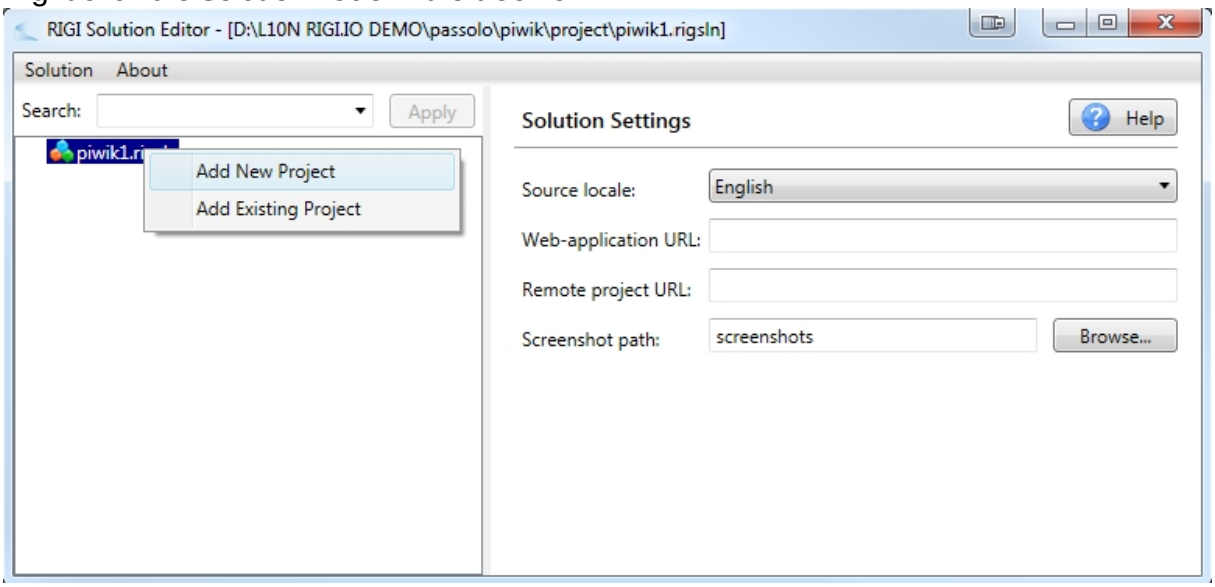
2. Select Solution > Save

Note: For projects that use local screenshots, there is an advanced setting to let Websnap fixate the width and/or height of the generated screenshots, see Screenshot resizing.

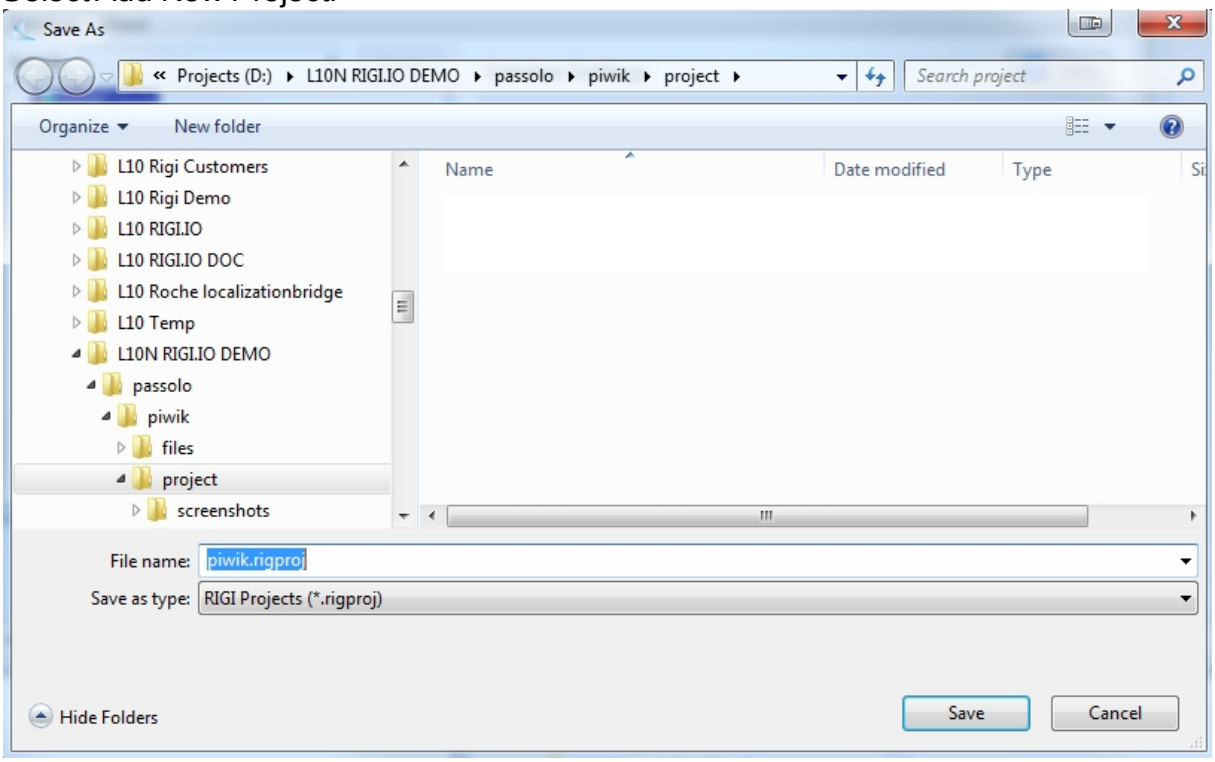
Add a project

A project contains information about the resource files that must be localized.

1. Right click the solution node in the treeview.



2. Select Add New Project.



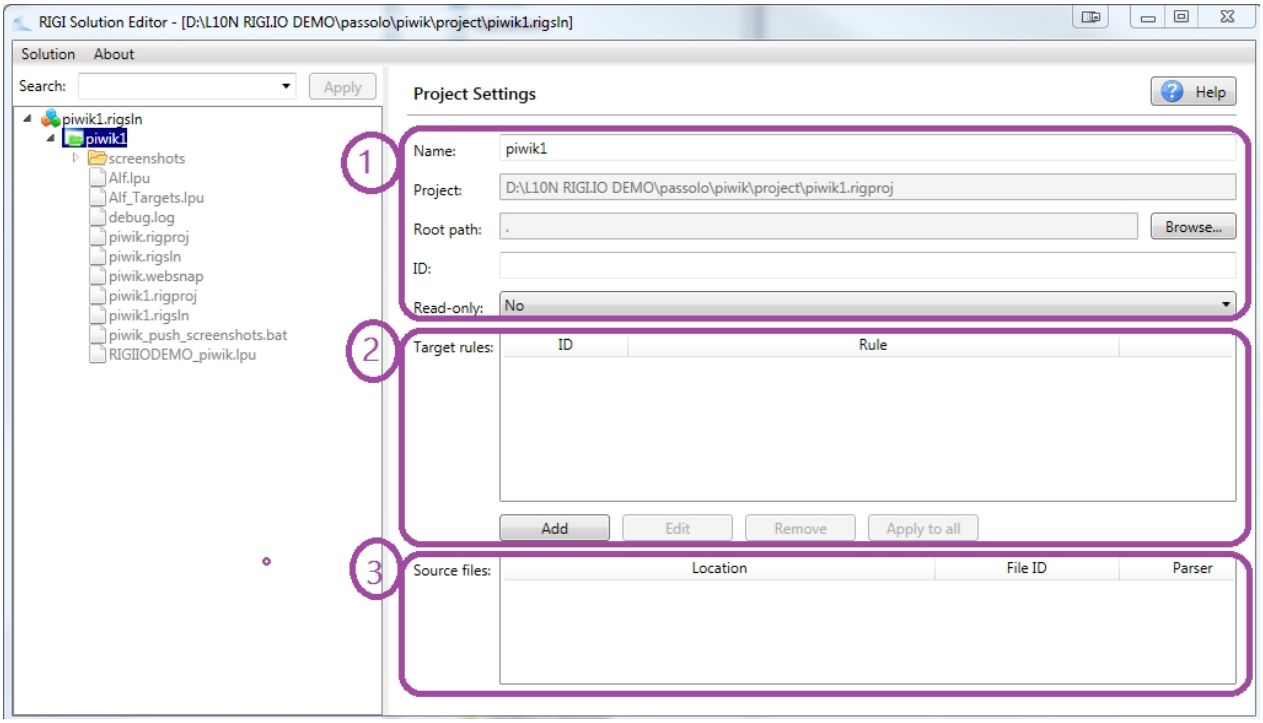
3. Enter the file name.
4. Click Save.

The empty project is created.

Define a project

This section explains how to define a project that is part of the solution.

The empty project that was created as part of the solution has quite an overwhelming layout.



Note: the screenshots in show the Source Files directly under the project. The latest version of the Solution Editor shows two nodes: Parser Settings and Source Files.

Execute the following steps:

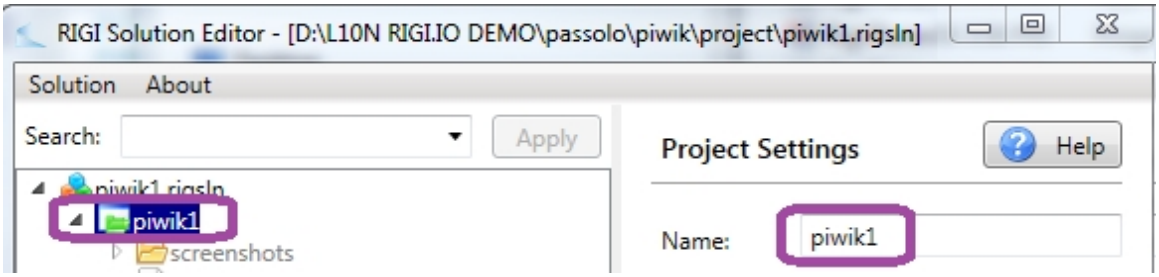
1. Enter project settings.
2. Enter target rules.
3. Select source files.
4. Define parser settings.

Enter project settings

Define the project attributes.

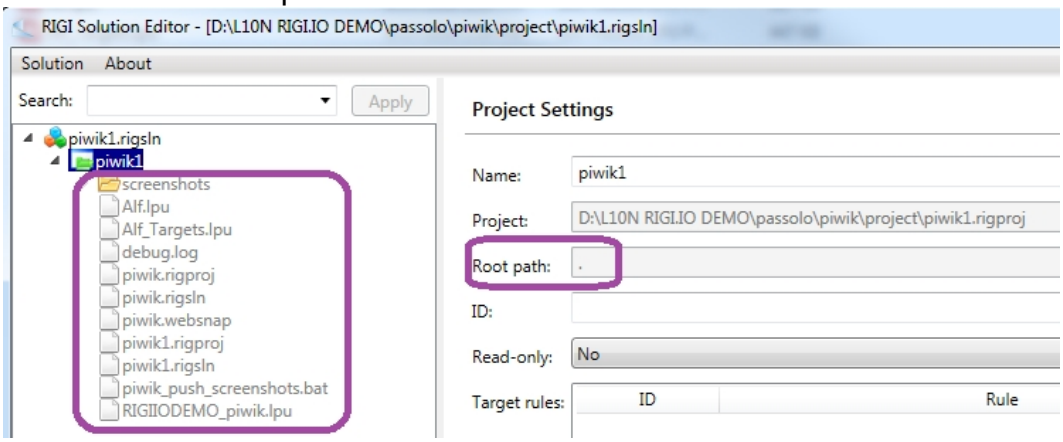
1. Enter the project settings.

Name: Enter a project name. This name is only used for display purposes. The entered project name is shown in the tree view.

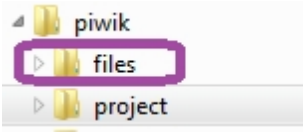


Project: This is the location of the project file (.rigproj). The absolute path is shown in the UI, but under the hood Rigi stores relative paths to the location of the solution file (.rigsln).

Root path: This is the root folder of where the files are located that must be localized. The initial value is the location of the project file. The tree view shows the files that are located in that path.

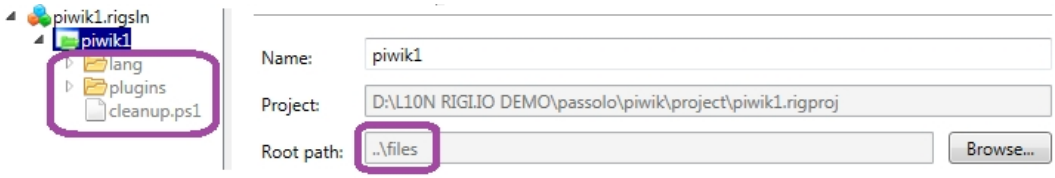


In this example, the source files or this particular project are not located in `piwik\project`, but in folder `piwik\files`.



Click the Browse-button to select that folder.

The treeview now shows the subfolders and files in that folder. The root path has a relative location to the solution.



ID: It is recommended to define a **project ID** for this project. Rigi concatenates three identifiers (separated with a dot) to determine a unique ID within the solution:

- Project ID - the identifier that is described here.
- File ID - each file in a project will be assigned with a unique File ID.
- String ID - each file contains localizable strings. Each string has a unique ID.

If the solution only contains one project, then you could decided to leave it empty. If a solution contains multiple projects, there is a risk for collisions if files in both projects have the same File ID and String ID.

Read-only: Define if the strings in a project are read-only (default = no). Rigi provides a WYSIWYG translation environment for translators and reviewers. Rigi substitutes translations in Rigi screenshot with the actual translation.

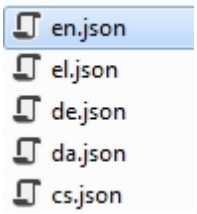
Some software projects may have different teams that work on various projects, where the projects have different localization cycles. In those cases, it is possible to import the translations from existing projects as a reference for the translator. It means that translators can see the translations, but not change them.

2. Select Solution > Save

Enter target rules

A Rigi project contains source files with translatable texts. The objective of Rigi is to generate localized target files that contain the translations. The location and name of those files are project dependent.

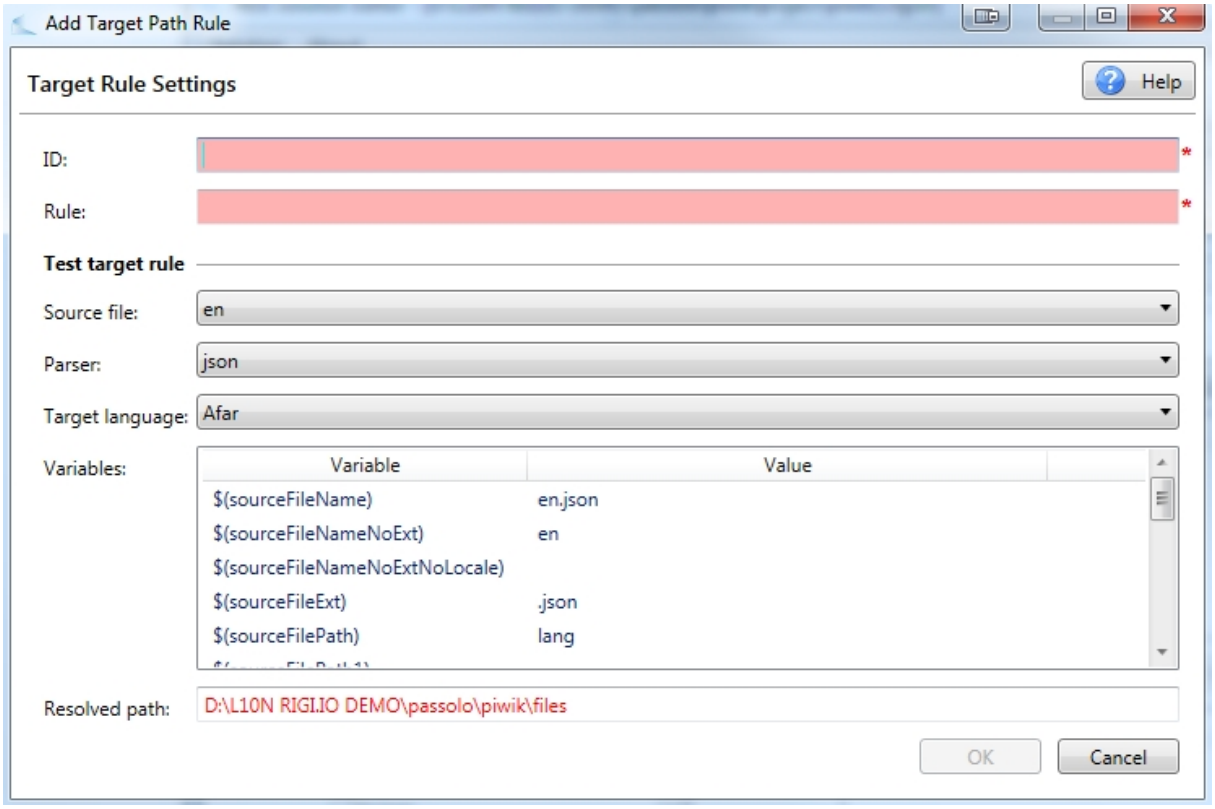
Localized Java properties files, for example, are usually located in the same folder as the source file and have the locale as extension. For example, the German target of `string.properties` gets name `strings.de.properties`. In the Piwik sample project, the Json files have name `<locale>.json`. The source is `en.json` and German target `de.json`. This requires that each resource file is located in a separate folder.



Each target rule has a unique ID. The rule itself contains variables `$(xyz)`, that represent for for example, the locale, source path and file extension.

1. It is recommended to add at least one source file to the project, see Select source files. This is required to test the resolved path of the target rule and see the variables.

2. Click Add



This dialog consists of three parts:

- Data entry: the goal of this dialog is to enter the **ID** and the **Rule**.
- Test target rule: select the source file ID, Parser and Target language to see what

the **resolved path** is.
- Variables: an overview of variables. The table shows the variable name and its value (derived from the selected **source file**).

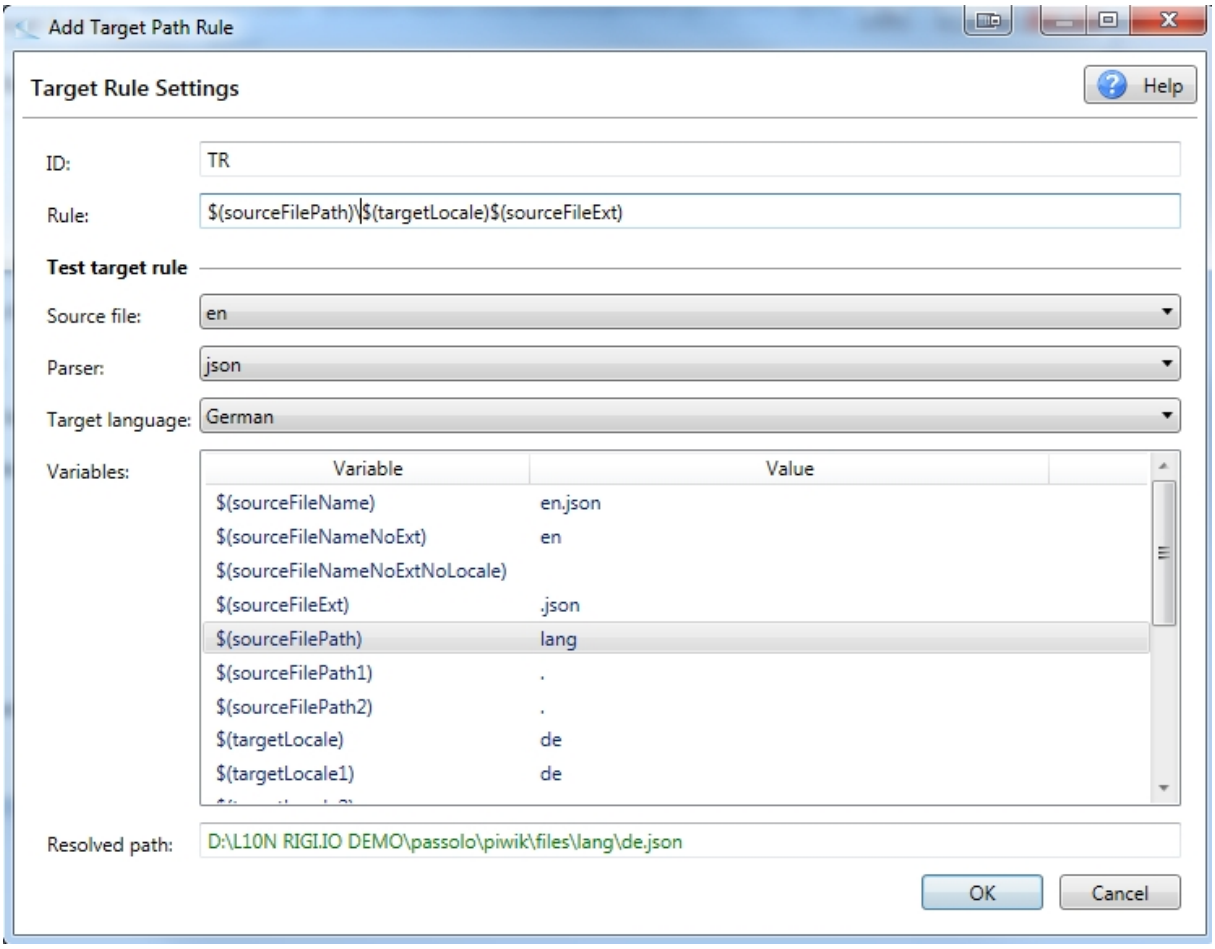
3. Enter the ID, for example "TR".

4. Enter the rule.

   Double-click a variable in the Variables-field to insert it in the Rule-field. Please note that the Variable-table supports quickly capturing the required target rule and it also makes sure that all characters are written properly in the path. Try to take as much advantage of the variables as possible when creating your target rule.

   The Resolved path presents the name and location of the target file with the current Rule. It acts as a check functionality when (old) target files are available at the target folder. *Resolved path* checks if there is an existing file at the path given in the *Rule*. As an additional aid, the text is colored red if the target file does not exist, otherwise green.

   In the example below, the German target file for `en.json` is located in the same folder (`$(sourceFilePath)`) as the source file and will get name (`$(targetLocale)$(sourceFileExt)`). This rule is then resolved to `lang\de.json`. That file already exists and therefore it is colored green.



This way it is possible to configure any mapping.

The project files implement a variable mapping mechanism. For example, `$(targetLocale)` maps to `de`. It can be overridden to map to any other value (e.g. `Berlin`). That mapping is not supported by the user interface of the Solution Editor, but can be achieved by editing the project file in an editor, see

5. Click OK

Note: it is possible to override the values of variables for specific locales. For example, `$(sourceFileExt)` can be overrridden with `".ItalianJson"` if the locale is `it-IT`. See Mapping of variables in target rules.

Add source file

A project consists of files. The source file is located in a folder and has a filename.

Each file needs to be assigned with the following attributes:

- File ID. This is a unique identifier for this file within this project.
- Parser. Select the Rigi parser that must be used for this file. Rigi comes with several parsers. It is possible to implement your own parsers using the *Rigi Parser SDK*.
- Target rule.

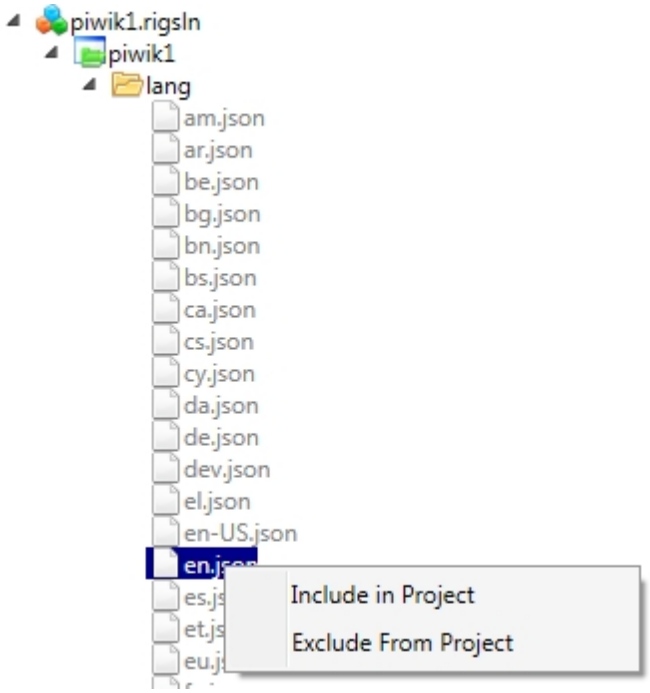
All source files in the Piwik demo application have the same name: en.json.

The following steps explain how to add one source file at a time.

1. In the tree view, select the file that must be added.

2. Right-click to see the context menu.

3. Select Include in project.



The file is added to the source files overview. The File ID gets a unique name (name of the file plus an optional number), a default Parser is assigned (based on the extension) plus a target rule.

4. You can change the File ID. File IDs are usually short names. Make sure that each File ID is unique within a project.

5. Make sure that the correct Parser and Target rule are selected.

   Under the hood, the file is added to the .rigproj file:

```
<?xml version="1.0" encoding="utf-8"?>
<project name="piwik1" relbasepath="..\files">
  <parsersettings />
  <targetrules>
    <targetrule id="TR">$(sourceFilePath)\$(targetLocale)$(sourceFileExt)</targetrule>
  </targetrules>
  <mappings />
  <resources>
    <file relpath="lang\en.json" guid="en" parser="json" targetrule="TR" />
  </resources>
</project>
```

You can add additional files by repeating the previous steps.

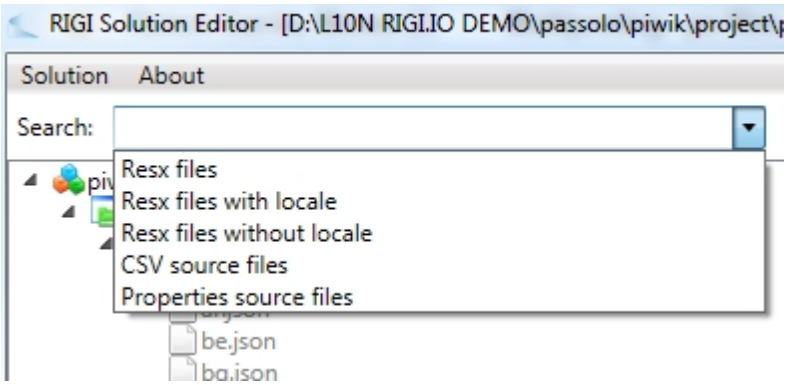
> *Tip: you can select multiple files at once using the Search/Apply function. This is explained in section Mass adding source files.*

Mass adding source files

Projects may have many files. Adding file-by-file to the project is time consuming. The Solution Editor supports multi-selection of files that can be added at once.

All source files in the Piwik demo project have filename `en.json`.

1. The Search dropdown box provides some prepared search functions.



   If you would select Resx-files, then after the selection, the Search box will show `\.resx$.`

   This is a Regular Expression that selects all files that end with `.resx`*. The $ indicates the end-of-string.*

2. Enter the regular expression in the Search field.

   In the case of demo application Piwik, we want to select all files that have name

`en.json`.

Note that the regular expression needs to escape the dot.

3. Click Apply

   All files that have name en.json are now selected.

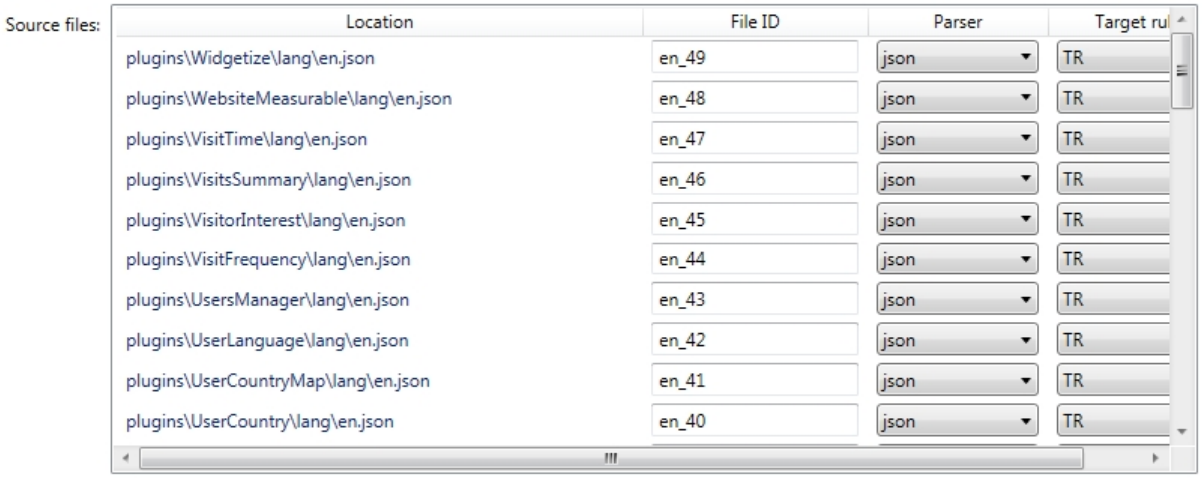
4. Right-click one of the selected files to see the context menu.

5. Select Include in project.

   All files are added to the project. An overview is shown:

6. Change the Target rule and/or Parser for all selected files if needed.

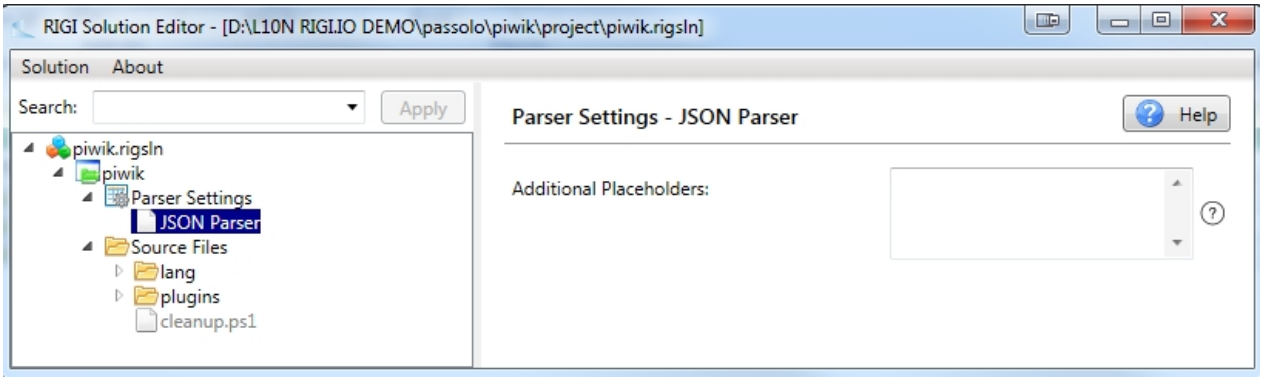
   The files are added to the project:



Parser settings

Use parser settings to configure parsers.

The resx-parser, for example, recognizes {0}, {1}, ... as placeholders for variables. It is possible to define additional placeholder patterns for any parser. For example, define the regular expression `%.+?%` to recognize variables like %var%. You can define as many additional placeholders as you like (one per line).



Hover the mouse over the setting to get additional explanation.

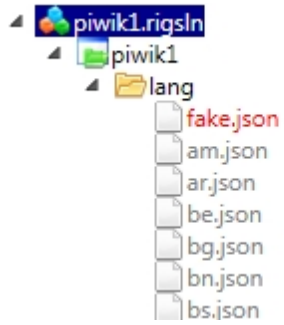
The JSON-parser does not have additional settings on top of the Additional Placeholders.

The Properties parser quite some settings. In the example below, an additional placeholder is defined to recognize a pattern like `{samplevar}` as variable.

Delete source file

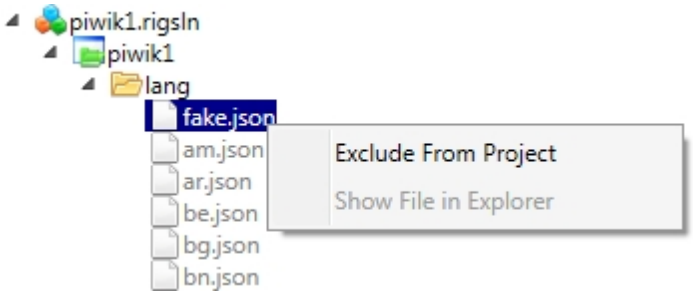
Engineers make changes to projects. It can happen that they removed or renamed a source file. In that case, the file is marked.

As an example, we defined a reference to file `fake.json` in the Piwik .rigproj file. Project files that do not exist on disk are marked in the Solution Editor:



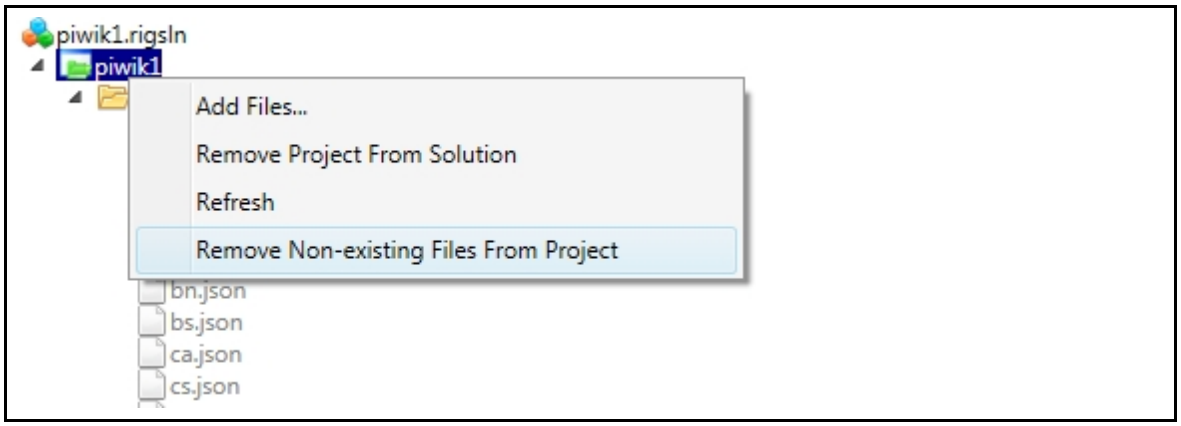
To remove that file from the project, execute the following steps:

1. In the tree view, select the file that must be removed.
2. Right-click to see the context menu.
3. Select Exclude from project.



The file is removed from the source files overview.

> *Tip: You can remove all unused files from a project at once.*
>
> *Use the context menu at project level:*


piwik1.rigsln
piwik1
Add Files...
Remove Project From Solution
Refresh
Remove Non-existing Files From Project
bn.json
bs.json
ca.json
cs.json

File formats

The solution and project are XML files (extensions .rigsln and .rigproj) that can be opened in an editor like Notepad++ and edited. Any modifications to these files need to be made within the Solution Editor to avoid the risk of missing key characters.

The solution contains one or more projects. The formats are described in the following sections:

- Solution file format (.rigsln).
- Project file format (.rigproj).

Solution file (.rigsln)

The solution file (extension .rigsln) for the demo application in this manual has the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<solution>
  <sourcelocale>en</sourcelocale>
  <pseudolocale>lv</pseudolocale>
  <signatureformat>ShortWeb</signatureformat>
  <screenshotpath>screenshots</screenshotpath>
  <url>http://www.locready.com/piwikdemo/</url>
  <remoteProjectUrl>https://xtmdemo.rigi.io/projects/piwi
k</remoteProjectUrl>
  <applicationtype>Web</applicationtype>
  <screenshotsizing>Dynamic</screenshotsizing>
  <projects>
    <refproject relpath="piwik.rigproj"
readonly="false" />
  </projects>
</solution>
```

Element	Description
solution	Root element, no attributes
sourcelocale	Language Culture Name of the source files (e.g. en-GB for English - United Kingdom and de-DE for German - Germany)
pseudolocale	Language Culture Name of the target files that contain the Rigi signatures. Use a locale that is not used as target language in your application. You can change the pseudo locale at a later stage during the project.
signatureformat	Rigi supports various formats to encode signatures to the pseudo locale file. • `ShortWeb` for web based applications. It uses a specific set of Unicode characters. • `ShortSilverlight` for Microsoft Silverlight applications. It uses a specific set of Unicode characters. • `Wpf` for Microsoft .Net WPF applications. It uses a specific set of Unicode characters. • `AlphaNumeric`. Some applications do not support the full range of Unicode characters. In that case alphanumeric characters can be used.
screenshotpath	Relative path (w.r.t. to the location of the solution file) where local screenshots are stored. This is typically a setting for developers, because translators will get the screenshots from Rigi server.
url	• For web applications: The URL of the live application. Keep this empty if there is no live application. • For WPF applications: The relative path to the executable.
remoteProjectUrl	• The URL of the project on the Rigi server that hosts the screenshots, e.g. `xyz.rigi.io/projects/projectname`.

	Keep this field empty if your screenshots are not hosted on rigi.io.
applicationtype	The type of application: Web, WPF, Silverlight, Android, iOS.
screenshotsizing	See Screenshot size fixation.
projects	The `projects` element contains `refproject` elements. That element has the following attributes: • `relpath`. The relative path (w.r.t. the solution file) to the project file (.rigproj). • `readonly`. If the project is read-only, then the translation management tool can display the translations, but not modify them.

Project file (.rigproj)

The solution for the demo application in this manual has one project file (extension .rigproj). That file has the following content:

```
<?xml version="1.0" encoding="utf-8"?>
<project name="piwik" guid="piwik" relbasepath="..
\files">
  <parsersettings />
  <targetrules>
    <targetrule id="tr">$(sourceFilePath)
\$(targetLocale)$(sourceFileExt)</targetrule>
  </targetrules>
  <mappings />
  <resources>
    <file relpath="lang\en.json" guid="Core"
parser="json" targetrule="tr" />
    <file relpath="plugins\en.json" guid="Actions"
parser="json" targetrule="tr" />
    ...
    <file relpath="Measurable\en.json" guid="meas"
parser="json" targetrule="tr" />
    <file relpath="Widgets\en.json" guid="widgets"
parser="json" targetrule="tr" />
  </resources>
</project>
```

Element	Description
project	Root element. That element has the following attributes: • `name`: name of the project. • `guid`: unique identifier of this project within this solution. It can be left empty. In that case, the files over all projects in the solution must have a unique ID. • `relbasepath`: path (w.r.t. the location of this project file) that is the root folder of the `resources`.
parsersettings	See Parser settings.
targetrules	The `targetrules` element contains `targetrule` elements. The `targetrule` element specifies the target rule, see also Enter target rules. The `targetrule` element has the following attributes: • `id.` The unique ID of the target rule within this project.
mappings	See Mapping of variables in target rules.
resources	The `resources` element contains `file` elements. The `file` element has the following attributes: • `relpath`. The relative path (w.r.t. the `project/@relbasepath`) of the file. • `guid`. Unique identifier of the file within the project. • `parser`. The ID of the parser. • `targetrule`. The ID of the target rule (specified in the

`targetrules`-section).

Advanced

> *This is only for experts. Editing the files in an editor may corrupt the structure. Always make a backup before you start editing.*

The Solution Editor is a tool that helps the user to define a Rigi solution and it projects. It does not support all editing features in the user interface (yet).

Under the hood, the editor creates XML files. The main solution file has extension .rigsln. It contains references to one or more project files (having extension .rigproj). You can edit these files in a text editor like Notepad++.

The following tweaks are possible:

- Map variables in a target rule to other values.
- Fixate the size of generated screenshots for the screenshots that Websnap generates.

Mapping of variables in target rules

A target rule in a project specifies variables that are resolved when the target file location and name is needed.

For example, variable `$(targetLocaleUnderscore)` is resolved as `it_IT` for locale `Italian (Italy)`.

There may be exceptions in some cases when the developers do not follow the standard rules.

Definition

Use the `mappings`-element in the .rigproj file to override the default behaviour.

- The `mappings`-element is the root element. It has `mapping` element with the following characteristics.
    - Attributes:
        - `id`. The ID of the mapping rule.
        - `var`. This specifies the variable to which the mapping applies.
    - The map-element defines map-elements. Each map-element has the following attributes:
        - `iso`. The ISO language/locale. For example, de-DE, it-IT, en-US, en-UK, en, de, nl.
        - `value`. The replacement for that specific locale.
        - `remove`. Defines the number of characters that must be removed before and after the replacement. Examples:
            - -1: removes 1 character before the match.
            - -2: removes 2 characters before the match
            - 1: removes 1 character before the match.
            - 2: removes 1 character before the match.
            - -1,2: removes 1 character before and 2 after the match.
- `Targetrule`-elements have an optional `mapping`-attribute. That `mapping`-attribute contains a comma-separated list of the `mapping` ids.

Example

This is best explained by means of an example. Assume that we have a project with 20 languages. For two variables, there are exceptions for it-IT and en-US.

- Exception 1: variable `$(targetLocaleUnderscore)`.
    - for Italian (Italy) it must be resolved as `ita_ITA`.
    - for English (United States), nothing shall be written and the preceeding underscore must be removed. This is typically the case when the source files must be overwritten.
- Exception 2: variable `$(sourceFileNameNoExt)`. Normally this is the name of the source file without the extension. E.g. "messages" if the filename is "messages.properties".
    - for Italian (Italy) it must be resolved as `"ItalianMessages"` instead of "messages"

The project file for this example is:

```
<?xml version="1.0" encoding="utf-8"?>
<project name="Test" relbasepath=".">
```

```
  <parsersettings />
  <targetrules>
    <targetrule id="TR"
mapping="map1,map2">targets\$(sourceFileNameNoExt)
_$(targetLocaleUnderscore).properties</targetrule>
  </targetrules>
  <mappings>
    <mapping id="map1" var="targetLocaleUnderscore">
      <map iso="it-IT" value="ita_ITA" />
      <map iso="en-US" value="" remove="-1" />
    </mapping>
    <mapping id="map2" var="sourceFileNameNoExt">
      <map iso="it-IT" value="ItalianMessages" />
    </mapping>
  </mappings>
  <resources>
    <file relpath="source\messages.properties" guid="MSG"
parser="properties" targetrule="TR" />
  </resources>
</project>
```

For example, the resolved variables for the following ISO codes are:

- it-IT: targets\ItalianMessages_ita_ITA.properties
- en-US: targets\messages.properties
- de-DE: targets\messages.de_DE.properties

Screenshot size fixation

It is possible to let Websnap generate previews with fixed dimensions. This can be configured via the `screenshotsizing` element in the solution. There is no possibility to set this via the Solution Editor.

Websnap tags the generated previews with meta data that contains the dimensions of the actual viewport (browser dimensions). The default use case is that when a preview is retrieved from Rigi server, it will be shown in an iframe that has the correct dimensions. The user can override those and resize the preview.

In specific use cases such as using screenshots that are stored locally on disk, the screenshots must get fixed dimensions at the moment Websnap creates them.

The `screenshotsizing` element in the .rigsln file defines if and how websnap should fixate the dimensions of the preview.

screenshotsizing	Description
Dynamic	(default). Websnap will not insert specific HTML codes to fixate the width and/or height.
FixedWidth	The generated screenshot will have a fixed width.
FixedHeight	The generated screenshot will have a fixed height.
FixedWidthHeight	The generated screenshot will have a fixed width and height.

Parser SDK

Rigi is shipped with general parsers, including resx, properties and json.

It is possible to implement custom parsers using the Rigi Parser SDK. For example, Rigi specifies an XML base parser where only a few methods need to be implemented.

In a parser it is possible to define:

- Logic how to read key-value pairs from a file and generate its target file.
- Placeholders. Using regular expressions, it is possible to specify for example that %s is a variable.
- Parser settings. Using these settings, it is possible to adapt the behaviour of the parser.
- Target path variables. It is possible to extend the list of variables that is shown in target rules editor with parser-specific rules.
- ... see the Rigi Parser SDK for all options.

Contact us for more informaton.