Building Multi-language Reports in Power BI

Power BI provides Internationalization and localization features which make it possible to build multi-language reports. For example, you can design a Power BI report that renders in English for some users while rendering in Spanish, French, German or Dutch for other users. If a company or organization has the requirement of building Power BI reports that support multiple languages, it's no longer necessary to clone and maintain a separate PBIX project file for each language. Instead, they can increase reuse and lower report maintenance by designing and implementing multi-language reports.

This document is a work in progress and should be considered a draft and not a final version

Contents

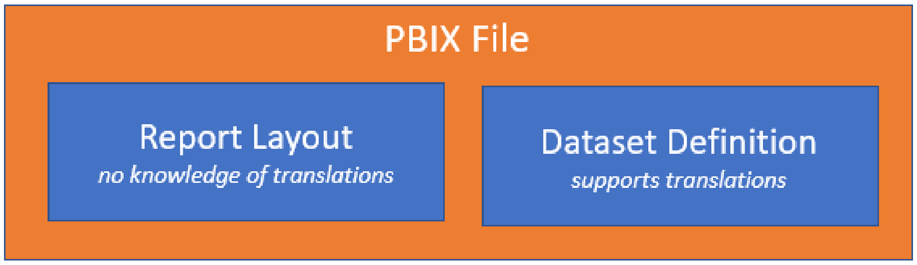
Overview of Multi-language Report Design

Power BI provides the features required to design and implement multi-language reports. However, the path to success is not overly intuitive. The purpose of this article is to explain how to use the Power BI features for Internationalization and localization from the ground up and to provide the guidance for building reports that support multiple languages.

The primary feature in Power BI used to build multi-language reports is known as **translations**. Power BI inherited this feature from its predecessor, Analysis Services, which introduced translations to add localization support for the data model associated with a tabular database or a multidimensional database. In Power BI, translations support has been integrated at the dataset level.

A translation represents the property for an object in a data model that's been translated for a specific language. Consider a simple example. If your data model contains a table with an English name of **Products**, you can add translations for the **Caption** property of this table object to provide alternative names for when the report is rendered in a different language. The object types in a Power BI dataset that support translations include tables, columns, measures and hierarchies. In addition to the **Caption** property which tracks an object's display name, dataset objects also support adding translations for two other properties which are **Description** and **DisplayFolder**.

Keep in mind that the Power BI support for translations only applies to metadata inside a dataset. The Power BI does not support adding translations for any text values that are stored as part of the report layout. For example, if you add to a textbox or a button a Power BI report and then after that you type in a literal text value, that text value cannot be localized. Therefore, you must avoid using textboxes and buttons with literal text when designing multi-language reports. As a second example, page tabs in a Power BI report are also problematic because their display names cannot be localized. Therefore, you must design multi-language reports so that page tabs are never displayed to the user.



A little later in this article you learn about the nitty-gritty details of building multi-language reports in Power BI Desktop. At this point, however, it's possible to make a high-level observation. For someone experienced with report building in Power BI Desktop, the challenge of learning how to build multi-language reports isn't as much about learning ***what to do*** but more about learning ***what not to do***. There are lots of popular Power BI report design techniques that cannot be localized and therefore cannot be used when building multi-language reports.

Metadata Translations versus Content Translations

Dataset translations are used to localize the metadata for dataset object properties such as the names of tables, columns and measures. Since these translations are created and maintained as part of the metadata for a dataset, they are also known as **metadata translations**. While metadata translation help to localize the names of tables and columns, they don't offer any assistance when it comes to localizing text values in the data itself. If your dataset has a **Products** table, how do you localize the text-based product names that exist in the individual rows of the **Products** table?

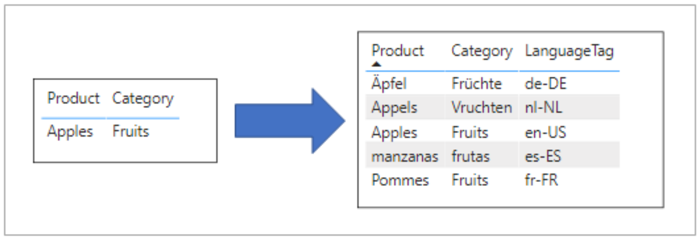
While adding metadata translations to your dataset is an essential first step, it doesn't usually provide a complete solution by itself. A complete solution often requires not only localizing the names of tables and columns, but also localizing the text-based content stored in table rows such as product names, product categories and country names. The key point is that the use of dataset translations to localize metadata must usually be complimented by a data model design which supports **content translation**.

Long before Microsoft introduced Power BI, software developers around the world have been building multi-language applications that support content translation. After two decades of designing and refining various database designs, several common design patterns have emerged as industry best practices to support content translation. Some of these design patterns involve adding a new table column for each language while other design patterns involve adding a new table row for each language. Each approach has benefits and drawbacks when compared to the other.

Currently, there is a limitation with DAX and the VertiPaq engine which makes it impractical to implement a content translation scheme based on adding a column for each language. The specific limitation is that calculated columns are evaluated at load time and do not yet support dynamic evaluation. While Microsoft has plans to update DAX and the VertiPaq engine to support dynamic calculated columns, there is currently no timeline for when this feature with be available in Preview or when it will reach GA.

Until dynamic column support is added, it doesn't make sense to implement a content translation scheme based on adding a column for each language. Currently, you would have to use measures instead of columns but that is very limiting because measures do not have row context. For example, measures cannot be used to supply values for the axes in a bar chart or line chart. Furthermore, measures cannot be used as legends or to filter data using slicers.

Currently, the best design pattern for content translation in a Power BI solution is row replication. Consider a simple example of using the row replication to implement content translation in a Power BI dataset. Let's say the **Products** table contains two text columns named **Product** and **Category** and you'd like your report to support five different languages including English, Spanish, French, German and Dutch. For each product in the **Products** table, you need to generate 5 records where each record contains the product name and product category translated to a specific language. Whenever the report is loaded, a row filter is applied to **LanguageTag** column so that users only see the rows for one of the supported languages at a time.

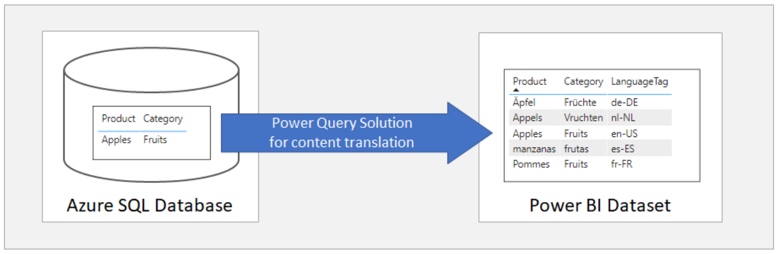


Product	Category
Apples	Fruits

Product	Category	LanguageTag
Äpfel	Früchte	de-DE
Appels	Vruchten	nl-NL
Apples	Fruits	en-US
manzanas	frutas	es-ES
Pommes	Fruits	fr-FR

When using the row replication design pattern in Power BI, you must somehow generate the extra rows which add the translated content into any table with text-based columns requiring translation. Many multi-language applications add support for content translation at the database level and use some type of ETL process to generate the extra rows with the translations. You can certainly use a content translation design supported by the underlying database when building multi-language reports in Power BI.

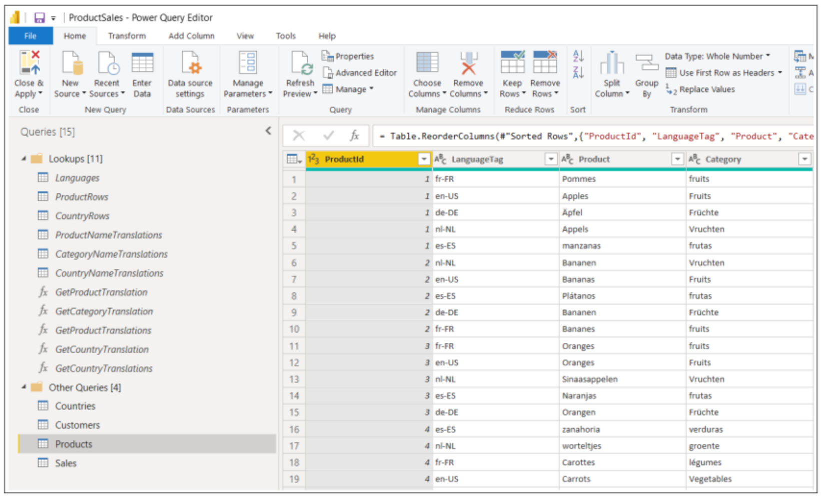
The developer sample named **ProductSales.pbix** that accompanies this article provides a content translations solution based on Power Query with an import-mode dataset. The solution is implemented using logic written in M which uses translation lookup tables to generate the extra rows when importing data from an Azure SQL database. This type of design allows you to create multi-language reports without having to make any changes to the underlying database. Instead, you can package all the ETL logic you need for content translations as query logic inside a PBIX template file.



Product	Category
Apples	Fruits

Product	Category	LanguageTag
Äpfel	Früchte	de-DE
Appels	Vruchten	nl-NL
Apples	Fruits	en-US
manzanas	frutas	es-ES
Pommes	Fruits	fr-FR

While implementing a content translation strategy with Power Query isn't always the right choice, it's great for scenarios where you don't have either the authority or the time it takes to add content translations at the database level. If you do decide to use this strategy, you'll find the writing Power Query logic in the M programming language provides developers with a very elegant way to replicate rows with translated content during a dataset refresh operation. You can review the entire Power Query solution in the **ProductSales.pbix** developer sample.



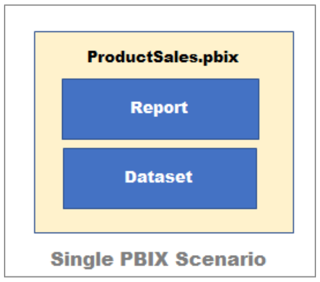
```
= Table.ReorderColumns(#"Sorted Rows",{"ProductId", "LanguageTag", "Product", "Cate
```

	ProductId	LanguageTag	Product	Category
1	1	fr-FR	Pommes	fruits
2	1	en-US	Apples	Fruits
3	1	de-DE	Äpfel	Früchte
4	1	nl-NL	Appels	Vruchten
5	1	es-ES	manzanas	frutas
6	2	nl-NL	Bananen	Vruchten
7	2	en-US	Bananas	Fruits
8	2	es-ES	Plátanos	frutas
9	2	de-DE	Bananen	Früchte
10	2	fr-FR	Bananes	fruits
11	3	fr-FR	Oranges	fruits
12	3	en-US	Oranges	Fruits
13	3	nl-NL	Sinaasappelen	Vruchten
14	3	es-ES	Naranjas	frutas
15	3	de-DE	Orangen	Früchte
16	4	es-ES	zanahoria	verduras
17	4	nl-NL	worteltjes	groente
18	4	fr-FR	Carottes	légumes
19	4	en-US	Carrots	Vegetables

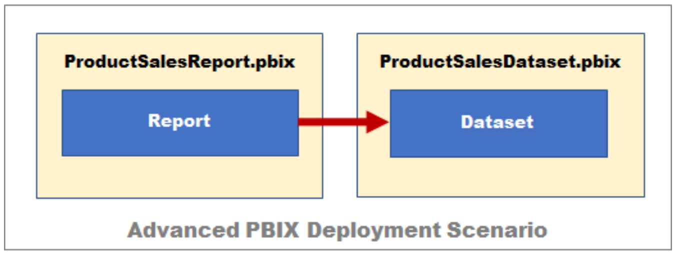
Multi-language Report Development Process

Now that you understand essential concepts of building multi-language reports, it's time to move forward and discuss how to structure the PBIX development process. First, you must decide whether to package and distribute your datasets and reports using a single PBIX project file or by using multiple PBIX project files. After that, you must define a workflow to take PBIX project files through the stages of the localization development process.

When you start a new project, you must decide how to package and distribute your dataset and report definitions. For simpler scenarios, you might decide to create a single PBIX project which contains both the dataset and the report.



For larger and more complicated scenarios it often make sense to separate datasets and reports into their own separate PBIX project files. This provides more flexibility in versioning because it makes it possible to push out updates to datasets and reports independently of one another. For example, you can maintain the dataset in one PBIX project file and publish that dataset to the Power BI Service. After that you can create one or more report template PBIX project files that connect to that dataset using Live Connect.



From the perspective of adding multi-language support, it really doesn't matter that much whether you decide to build your solution using a single PBIX project file or using multiple PBIX project files. The techniques used to add multi-language support to Power BI reports are the same. There are specific tasks you need to perform at the dataset level and other tasks and disciplines you must use when building reports. These task and disciplines don't really change if you move from a single PBIX file distribution strategy to a multiple PBIX file distribution strategy.

The PBIX development process for a multi-language report can be broken down into a few distinct phases. Each of these phases will be examined in detail in this article.

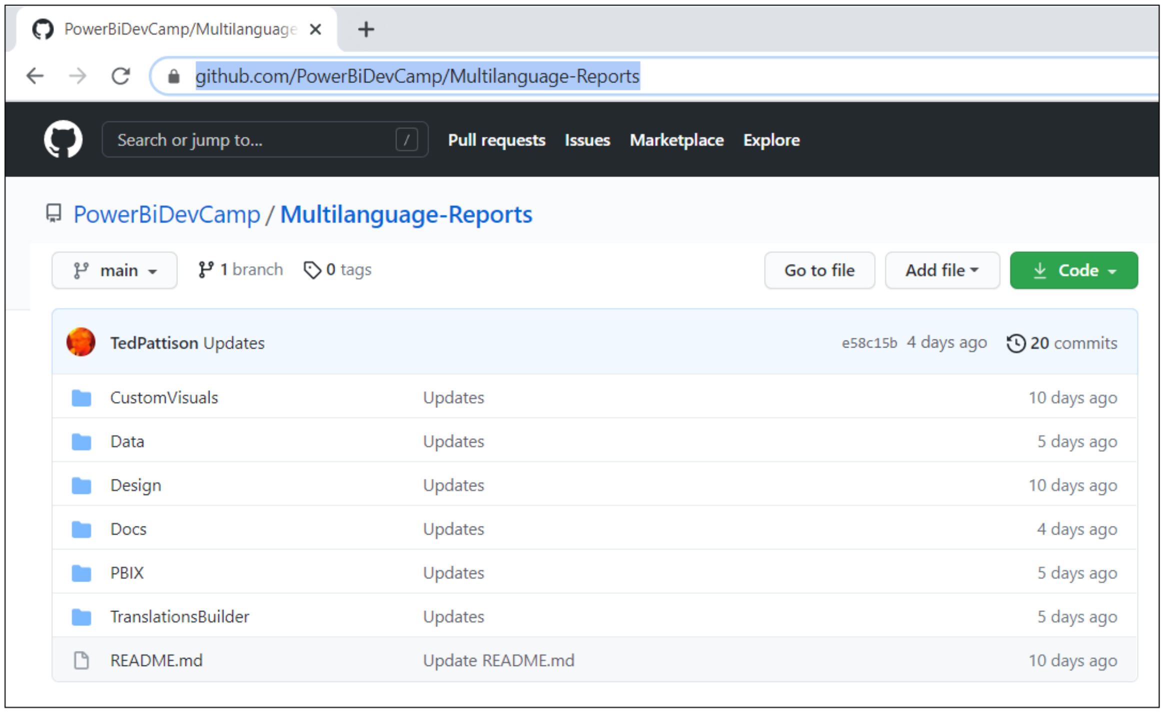
1. Prepare the PBIX project file for localization
2. Extend the PBIX project file with metadata translations
3. Design and implement a content translation strategy

The ProductSales.pbix Developer Sample

This article is accompanied by a developer sample based on a PBIX project file named **ProductSales.pbix**. This PBIX project file has been created to demonstrate building a multi-language report for Power BI which supports English, Spanish, French, German and Dutch. The PBIX project files and all the supporting code and translation files used to build it are available to view online or to download from a GitHub repository at the following URL:

https://github.com/PowerBiDevCamp/Multilanguage-Reports

This GitHub repository contain several different versions of the **ProductSales.pbix** so you can see what the PBIX project looks like at various stages of the development process. The GitHub repository also contains a few other development projects including a Power BI custom visual named **LocalizedLabel** and a C# console application named **TranslationsBuilder** which is used as an external tool with Power BI Desktop to automate the process of adding and updating dataset translations.


PowerBiDevCamp/Multilanguage
github.com/PowerBiDevCamp/Multilanguage-Reports
Search or jump to...
Pull requests
Issues
Marketplace
Explore
PowerBiDevCamp / Multilanguage-Reports
main
1 branch
0 tags
Go to file
Add file
Code

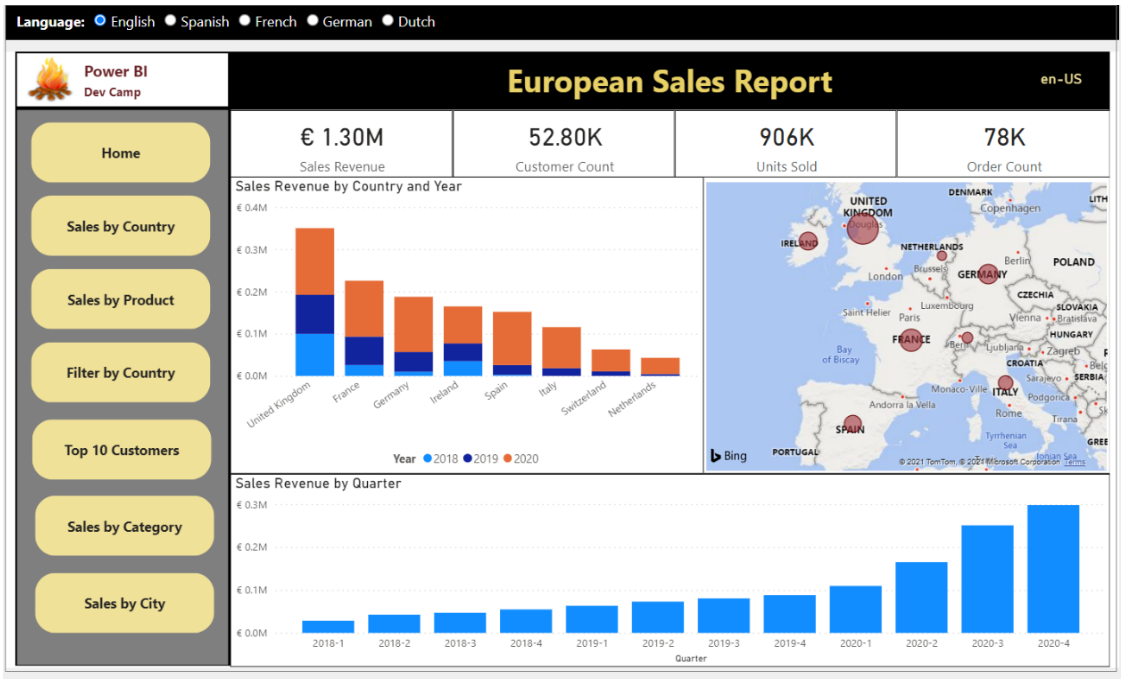
TedPattison Updates   e58c15b 4 days ago   20 commits

CustomVisuals	Updates	10 days ago
Data	Updates	5 days ago
Design	Updates	10 days ago
Docs	Updates	4 days ago
PBIX	Updates	5 days ago
TranslationsBuilder	Updates	5 days ago
README.md	Update README.md	10 days ago

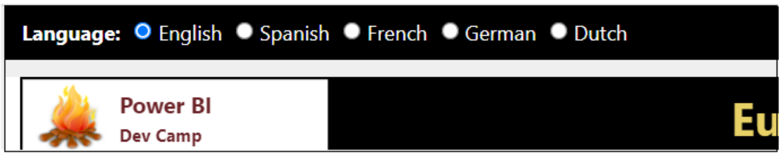
In addition to the project files available in the GitHub repository, there is also a live version which you can look at to see the completed multi-language report in action.

https://multilanguagereportdemo.azurewebsites.net

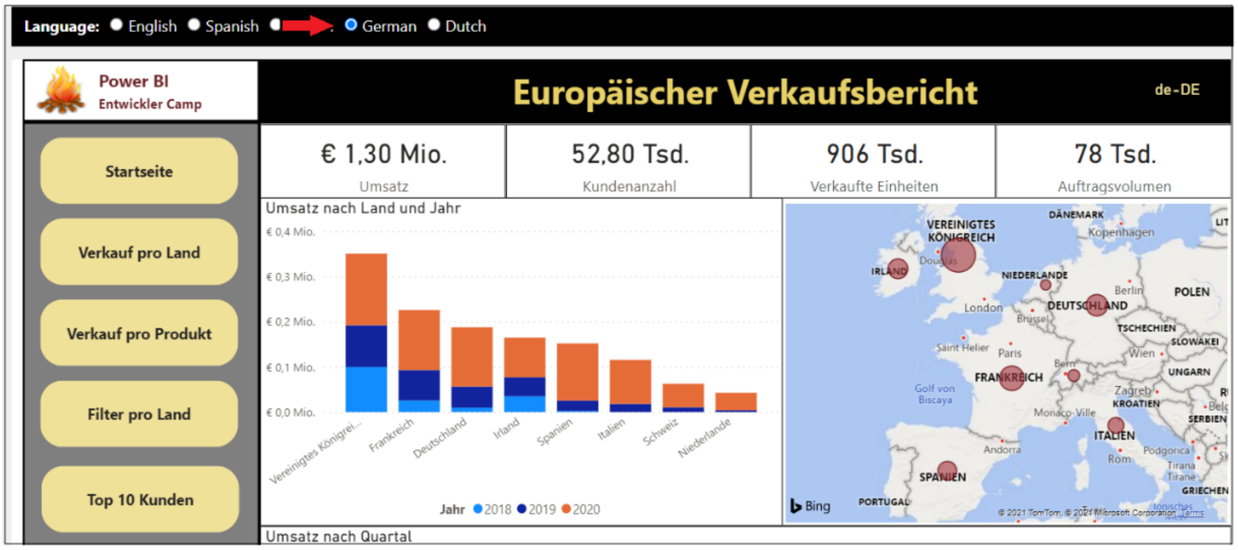
If you navigate to this URL in a browser, you will see the completed solution for **ProductSales.pbix** displayed using it's default language of English. There are buttons in the left-hand navigation with captions such as **Home**, **Sales by Country** and **Sales by Product** which make it possible for the user to navigate from page to page.



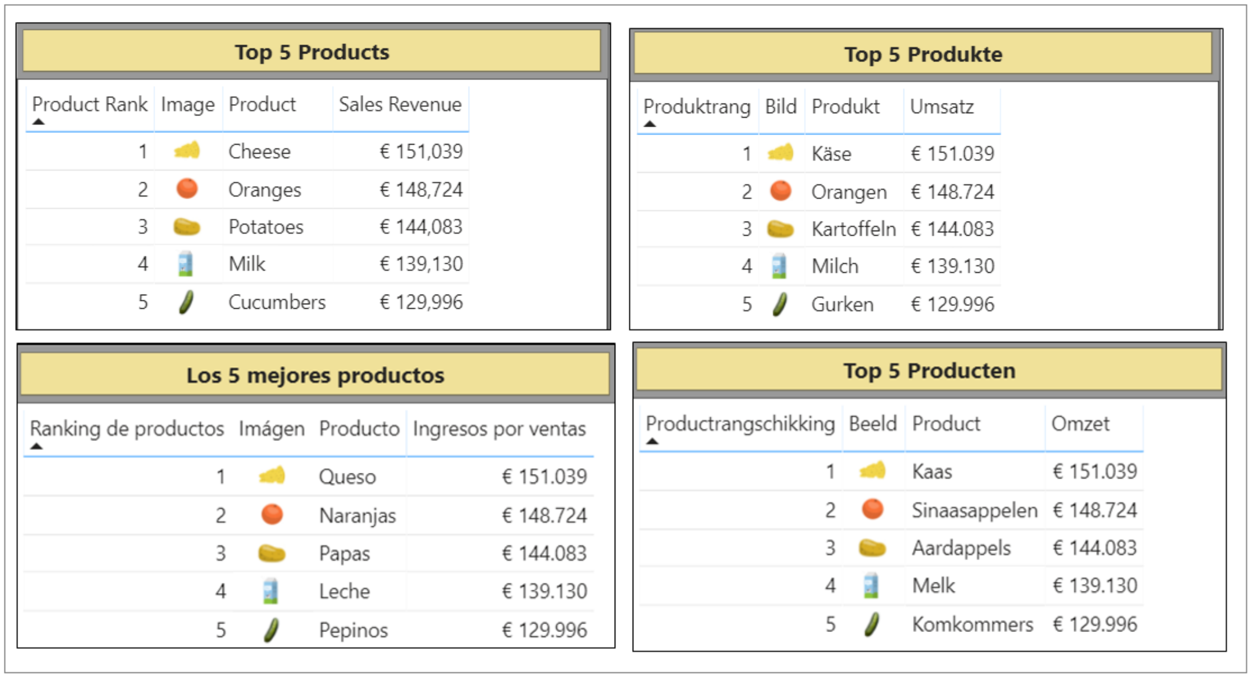
At the top of the page above the embedded report, you will see a set of radio buttons that allow you to reload the report using a different language.



You can click any of these radio buttons to reload the report in a specific langauge. For example, click on the radio button with the caption of **German**. When you do that, there is JavaScript behind this page that responds by reloading the report using the language of German intead of English. You can see that all the button captions in the left-hand navigation amd text-based values in the visuals on the page now display their German translations instead of English.



If you experiment loading the report with each of the supported languages, you can see that the reports implements both metadata translation and content translations. The names of the columns and measures change as you switch between languages as well as the text-based values for the product names from the rows of the **Products** table.



Top 5 Products

Product Rank	Image	Product	Sales Revenue
1		Cheese	€ 151,039
2		Oranges	€ 148,724
3		Potatoes	€ 144,083
4		Milk	€ 139,130
5		Cucumbers	€ 129,996

Top 5 Produkte

Produktrang	Bild	Produkt	Umsatz
1		Käse	€ 151.039
2		Orangen	€ 148.724
3		Kartoffeln	€ 144.083
4		Milch	€ 139.130
5		Gurken	€ 129.996

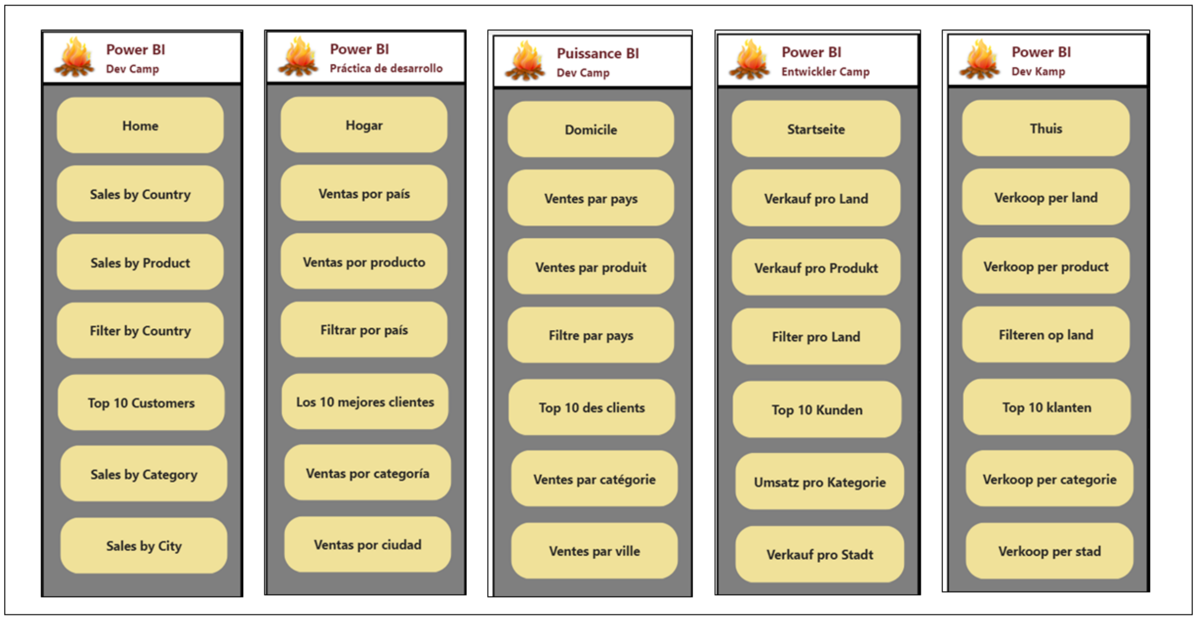
Los 5 mejores productos

Ranking de productos	Imágen	Producto	Ingresos por ventas
1		Queso	€ 151.039
2		Naranjas	€ 148.724
3		Papas	€ 144.083
4		Leche	€ 139.130
5		Pepinos	€ 129.996

Top 5 Producten

Productrangschikking	Beeld	Product	Omzet
1		Kaas	€ 151.039
2		Sinaasappelen	€ 148.724
3		Aardappels	€ 144.083
4		Melk	€ 139.130
5		Komkommers	€ 129.996

In addition to column names, measure names and the row-based content, reports are often designed with labels for report elements such as headings and captions for buttons. In the previous example, there's a heading **Top 5 Products** which has been localized along with the other translations. Also, there are captions for each button in the left navigation that have been translated as well. Here is a screenshot which shows the left navigation for all 5 languages side by side.



Power BI Dev Camp	Power BI Práctica de desarrollo	Puissance BI Dev Camp	Power BI Entwickler Camp	Power BI Dev Kamp
Home	Hogar	Domicile	Startseite	Thuis
Sales by Country	Ventas por país	Ventes par pays	Verkauf pro Land	Verkoop per land
Sales by Product	Ventas por producto	Ventes par produit	Verkauf pro Produkt	Verkoop per product
Filter by Country	Filtrar por país	Filtre par pays	Filter pro Land	Filteren op land
Top 10 Customers	Los 10 mejores clientes	Top 10 des clients	Top 10 Kunden	Top 10 klanten
Sales by Category	Ventas por categoría	Ventes par catégorie	Umsatz pro Kategorie	Verkoop per categorie
Sales by City	Ventas por ciudad	Ventes par ville	Verkauf pro Stadt	Verkoop per stad

As you remember from earlier, the Power BI report designer does not support any type of localization or translation on its own. Therefore, you have to use a special trick to localize labels in a Power BI report. In the next section, you will see how this is done by adding a localized label table into the data model.

Prepare the PBIX Project File for Localization

So far you've learned the essential concepts and background information you'll need to build multi-language reports. Now, it's time to move ahead and discuss the actual development process. We'll start by discussing a few general topics associated with software localization. After that, we'll move on to topics that are specific to Power BI and designing reports in Power BI Desktop.

When it comes to localizing software, there are some universal principals to keep in mind. The first is to plan for localization from the start of any project. It's significantly harder to add localization to an existing dataset or report that was initially built without any regard for internationalization or localization. This is especially true with Power BI reports because there are so many popular design techniques that do not support localization. You might find that much of the work for adding localization support to existing Power BI reports involves moving backward and undoing the things that do not support localization before you can move forward with design techniques that support localization.

Another important concept in localization is to plan for growth. A label that's 400 pixels wide when displayed in English could require a much greater width when translated into another language. If you optimize the width of your labels for text in English, you might find that translations in other languages introduce unexpected line breaks or get cut off which, in turn, creates a compromised user experience.

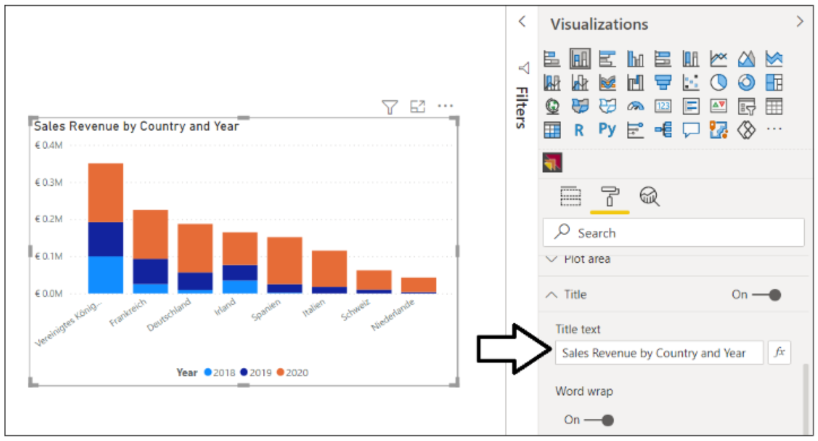
Adding a healthy degree of padding to localized labels is the norm when developing localized software and it's essential that you test your report with each language you plan to support. In essence, you need to ensure your report layout looks the way you expect with any language you have chosen to support.

Avoid report design techniques that do not support localization

If you have experience with Power BI Desktop, it's critical that you learn which report design techniques to avoid when you begin building multi-language reports. Let's begin with the obvious things which cause problems due to a lack of localization support.

- Using textboxes or buttons with literal text
- Adding literal text for the title of a visual
- Displaying page tabs to the user

The key point here is that any literal text that gets added to the report layout cannot be localized. Consider the case where you add a column chart to your report. By default, a Cartesian visual such as a column chart is assigned a dynamic **Title** property which is parsed together using the names of the columns and measures that have been added into the data roles such of **Axis**, **Legend** and **Values**.



There is good news here. The default **Title** property for a Cartesian visual is dynamically parsed together in a fashion that supports localization. As long as you have supplied dataset translations for the columns and measures in the underlying data model (e.g. **Sales Revenue**, **Country** and **Year**), the **Title** property of the visual will use the translations for whatever language is used to load the report. The following table shows how the default **Title** property of this visual is updated from each language supported by the **ProductSales.pbix** developer sample.

Language	Visual Title
English	Sales Revenue by Country and Year
Spanish	Ingresos por ventas por país y año
French	Chiffre d'affaires par pays et année
German	Umsatz nach Land und Jahr
Dutch	Omzet per land en jaar

Even if you don't like that default visual **Title** property, you must resist replacing it with a literal text value. Any literal text you type into the **Title** property of the visual will be added to the report layout and, therefore, cannot be localized. Therefore, you should either leave the visual **Title** property with its default value or hide the title so it is not displayed.

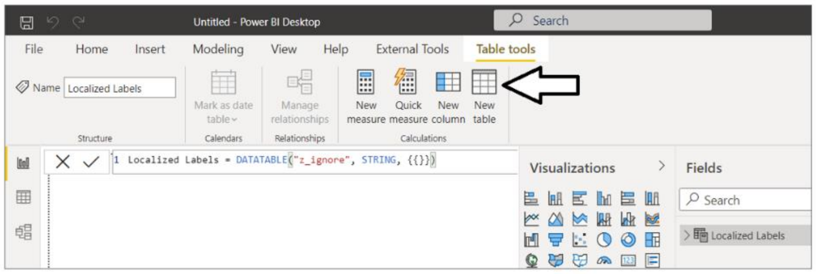
You should also understand that Power BI Apps do not support localization. That's because Power BI Apps are built with a navigation scheme which uses the text captions for page tabs which cannot be localized. Therefore, you must plan to deploy multi-language reports using something other than a Power BI App.

Create the Localized Labels table

When designing reports, it's a common practice to use text labels for report elements such as titles, headings and button captions. You've learned that any text value stored in a report layout cannot be localized. If you want to localize the text labels displayed on a Power BI report, those labels must be defined inside the dataset and not inside the report layout. This leads to the creative design technique of creating a specialized table in the data model for localized labels.

The idea behind the **Localized Labels** table is pretty simple. You can localize the name of any measure inside a dataset. When you need a text label for a report title, you can add a new measure to the **Localized Labels** table and name the measure using the English label for the report title such as **European Sales Report**. Since the label is a measure name, you can add translations to supply a text version of the label for each language.

There are several different techniques that can be used in Power BI Desktop to add a new table into a data model. For example, you can click on the **New table** button on the Table tools tab and then add a DAX expressions with the DATATABLE function to create a new table named **Localized Labels**.

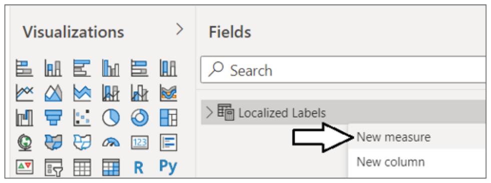


```
Localized Labels = DATATABLE("z_ignore", STRING, {{}})
```

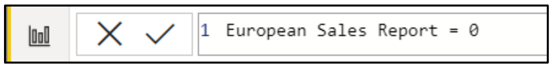
Below is the full DAX expressions that creates the **Localized Label** table. The DATATABLE function requires that you create a least one column. Therefore the table is created with a single column named **z_ignore** which can be hidden from report view.

```
Localized Labels = DATATABLE("z_ignore", STRING, {{}})
```

Once you have created the **Localized Labels** table, you can begin to add new measure just as you would in any other Power BI Desktop project.

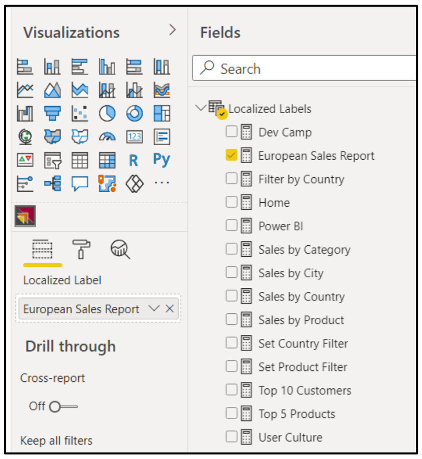


When creating a measure for a localized label, you can add the label text as the measure name and then set the DAX expression for this measure to a static value of 0. This value of 0 has no significance and is only added because each measure must be created with a DAX expression.



```
European Sales Report = 0
```

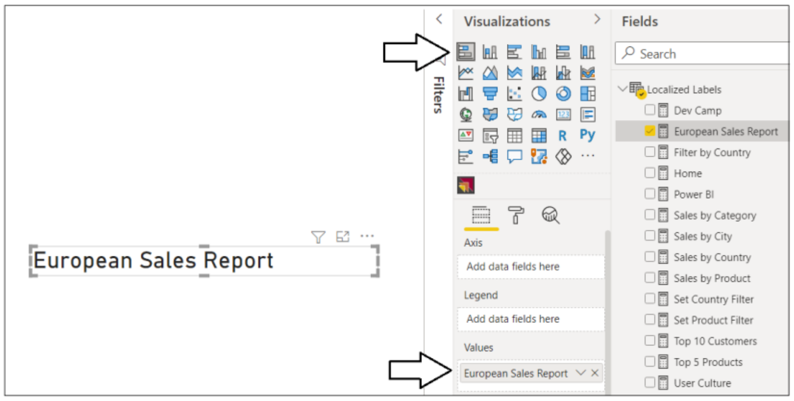
The **ProductSales.pbix** developer sample use this technique to add localized labels for all the titles, headings and button captions used throughout the report.



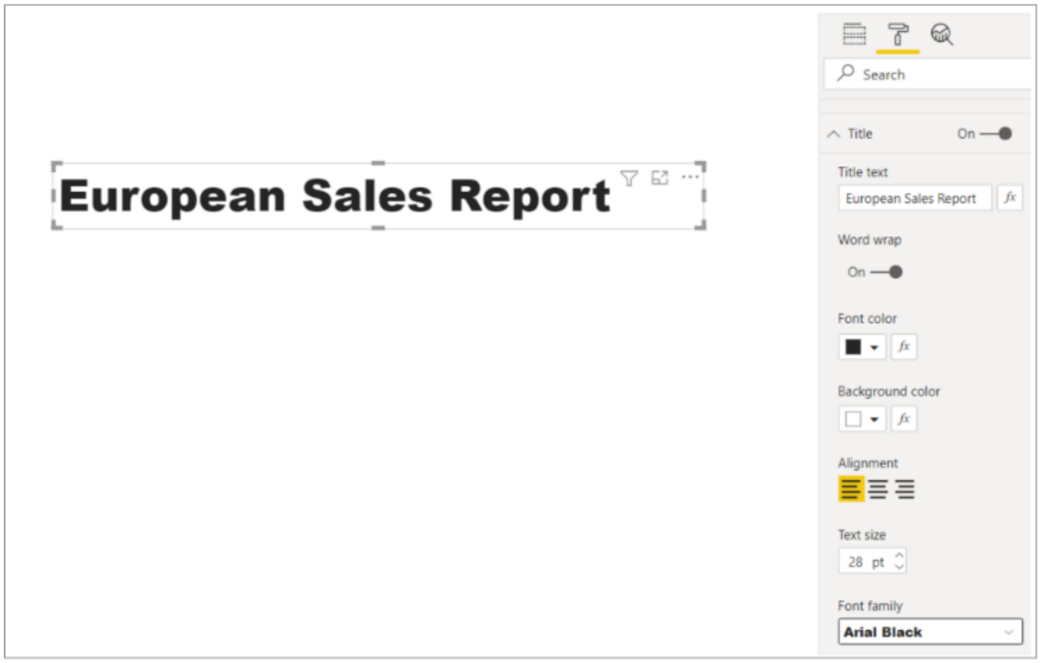
Now that you've seen how to create the **Localized Labels** table, it's time to learn how to surface the measure name for a localized label on a Power BI report.

Display localized labels using Power BI core visuals

The technique used to display measure names from the **Localized Labels** table in a Power BI report is neither straightforward nor intuitive. You can start by adding a Cartesian visual such as the Stacked Barchart visual to a page and then adding the measure for the desired localized label into the **Values** data role. Once you have added the Stacked Barchart visual, you can adjust the visual's height so only the visual **Title** property can be seen by the user.



After you have added the visual to a report page to display a localized label, you can adjust the font formatting for the label in the **Title** section of the **Format** pane as shown in the following screenshot.

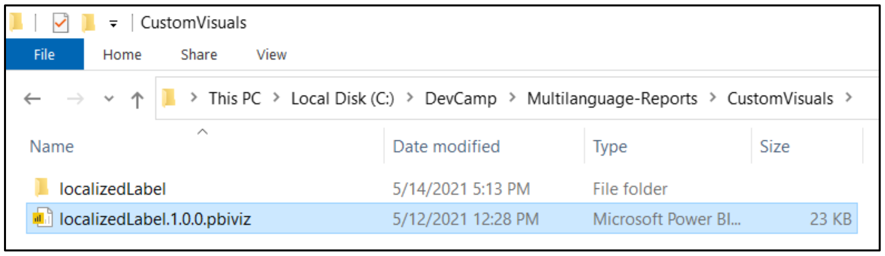


In summary, you can use the Power BI core visuals such as the Stacked Barchart to display localized labels for titles, headings and button captions. However, most report authors find that the design experience is pretty limited when formatting a localized label using any of the Power BI core visuals. For example, you cannot configure padding for the label or center its text vertically. In the next section, we will discuss creating a custom visual to provide a much improved design experience for display localized labels on a Power BI report.

Display localized labels using the LocalizedLabel custom visual

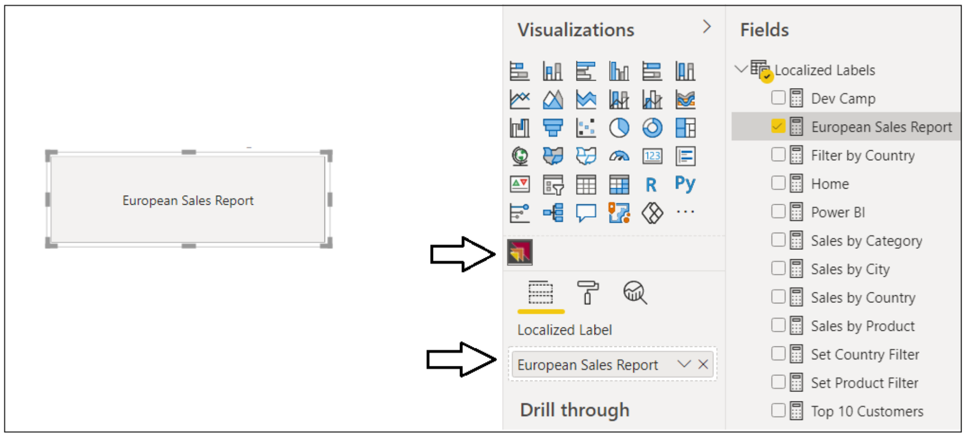
The last section discussed using a Stacked Barchart visual to display a localized label. While this technique can be used reliably to create multi-language reports, it is clear that none of the Power BI core visuals were not designed to support this scenario. The reason this technique works is really a more of a coincidence than a planned outcome. Furthermore, there is extra overhead due to the visual implementation being designed to do far more then just display a title. This provides a motivation to develop a custom visual that is explicitly designed to support the scenario of displaying a localized label.

This GitHub repository mentioned earlier contains a CustomVisuals folder. Inside the **CustomVisuals** folder, there is a child folder with a custom visual development project named **localizedLabel**. If you have experience with custom visual development, you can open the **localizedLabel** project in Visual Studio Code to see how this custom visual is implemented. The **CustomVisuals** folder also contains a custom visual distribution file for the **localizedLabel** project named localizedLabe.1.0.0.pbiviz. You can import this custom visual distribution file directly into a Power BI Desktop project to begin using this custom visual to display localized labels.

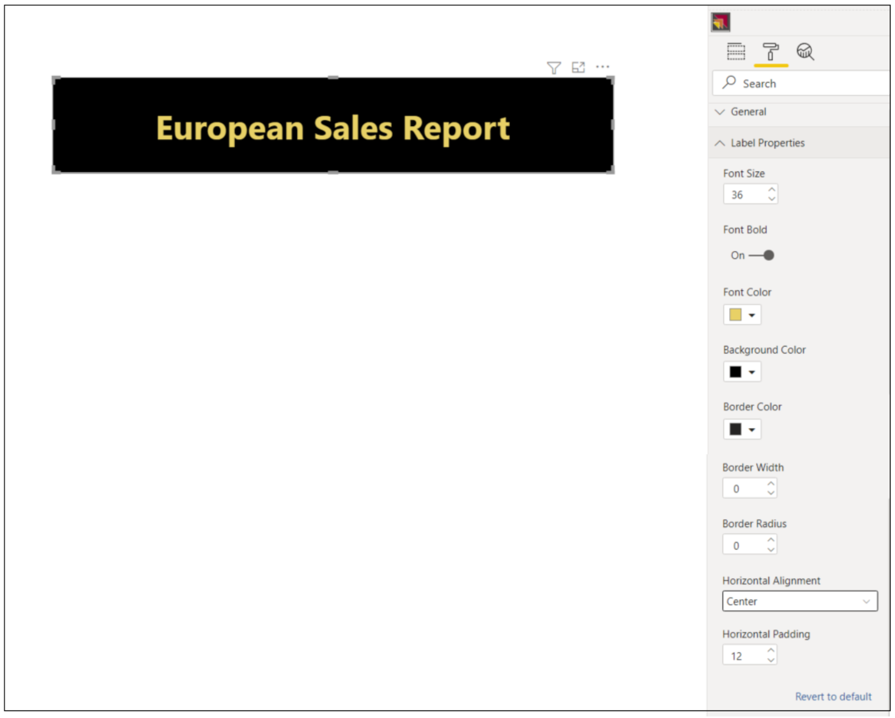

CustomVisuals
File
Home
Share
View
This PC > Local Disk (C:) > DevCamp > Multilanguage-Reports > CustomVisuals

Name	Date modified	Type	Size
localizedLabel	5/14/2021 5:13 PM	File folder	
localizedLabel.1.0.0.pbiviz	5/12/2021 12:28 PM	Microsoft Power BI...	23 KB

Once you have imported the custom visual distribution file named localizedLabe.1.0.0.pbiviz into a Power BI Desktop project, you should be able to begin using it. Once you have added an instance of the Localized Label visual to a report, you can add one of the measures in the **Localized Labels** table into the **Localized Label** data role as shown in the following screenshot.



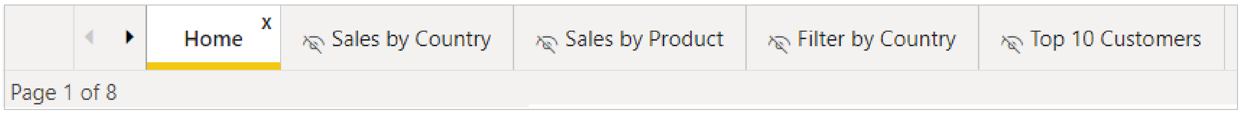
After you have added configured the **Localized Label** data role, you can configure the font and background formatting of the visual in the **Label Properties** section of the Format Pane.



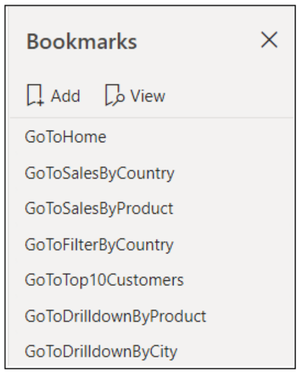
Font Size: 36
Font Bold: On
Font Color
Background Color
Border Color
Border Width: 0
Border Radius: 0
Horizontal Alignment: Center
Horizontal Padding: 12
Revert to default

Add support for page navigation

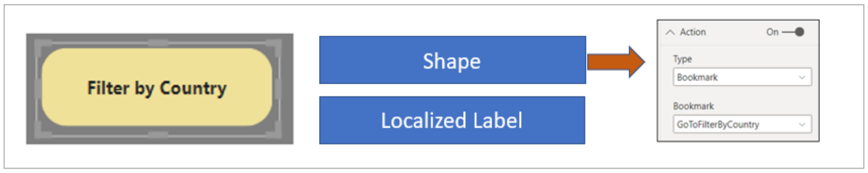
Given that you cannot display page tabs to the user in a multi-language report, you must provide an alternative way for users to navigate from page to page. This can be accomplished by creating shapes which act as buttons. When the user clicks on a shape, the share will apply a bookmark to navigate to another page. To use this technique, you can begin by hiding every page in a report except for the first page which will act as the report's landing page.



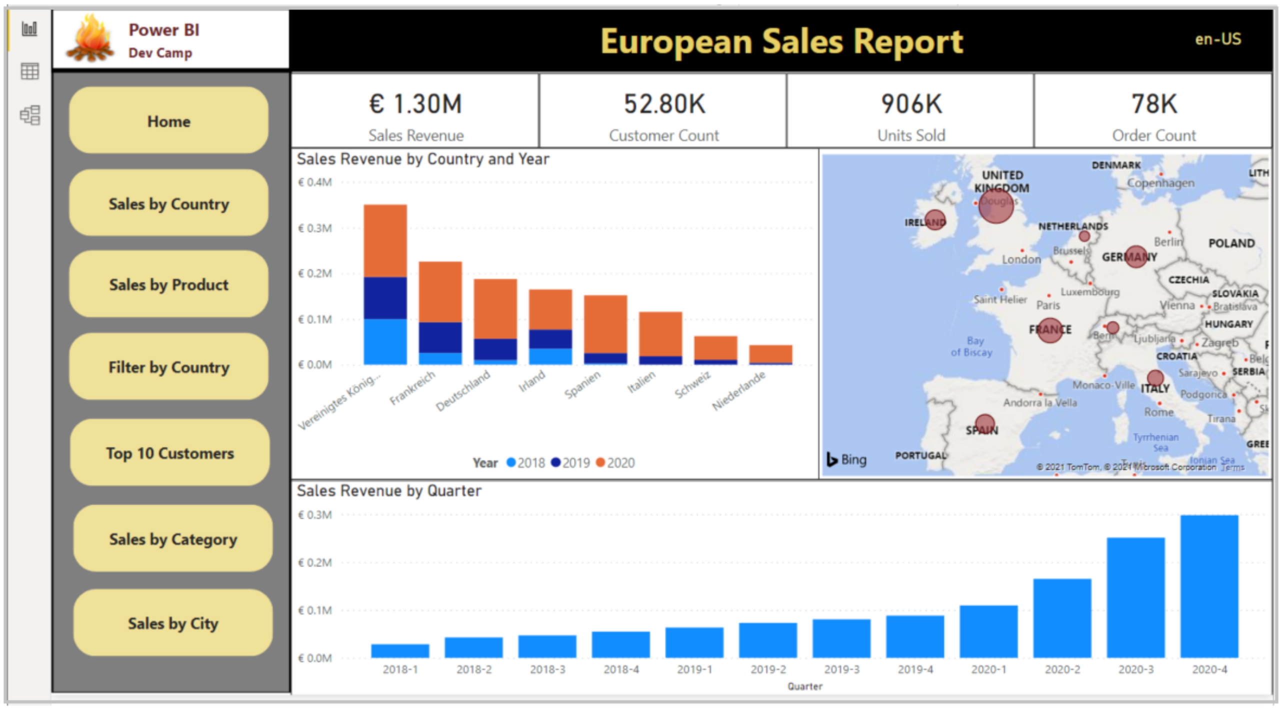
Next, you should create a set of bookmarks. Each bookmark should be created to navigate to specific page in the report. For example, the **ProductSales.pbix** developer samples defines the set of navigation bookmarks shown in the following screenshot.



Remember, that you cannot add a button with literal text to a multi-language report. Instead, you must be a bit more creative and use a technique that supports localizing button captions. This can be accomplished by using a shape which is overlaid on top of a localized label visual. The shape should be configured as completely transparent without a border or any background color. The shape should also be configured with an action to trigger a bookmark. The idea is that the user only sees the localized label with the button caption. However, when the user clicks on the localized label visual, there is a transparent shape on top that triggers the action to apply a bookmark and navigate to another page.



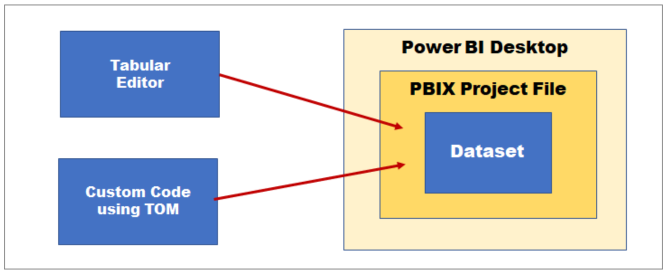
The **ProductSales.pbix** developer sample uses this technique to build a left navigation menu with a set of buttons which allows users to navigate between the pages in the report. The key point is that these button captions can be localized and will be displayed using metadata translations.



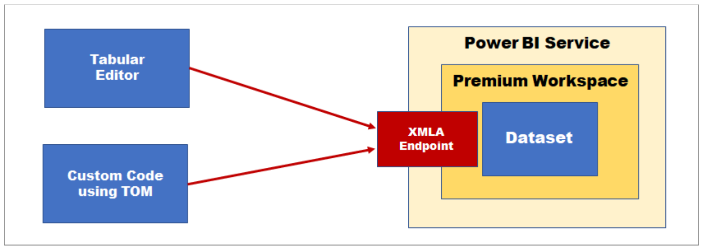
Extend the PBIX project file with metadata translations

The previous section discussed how to prepare a PBIX project file for localization. In this section we'll discuss how to add dataset translations for each language you need to support. You can accomplish this task by hand using an external tool named the **Tabular Editor**. Alternatively, you can automate the task using an Power BI dataset API known as the **Tabular Object Model (TOM)**. With either approach you can work directly on a PBIX project file that is open in Power BI Desktop.

While Power BI Desktop doesn't offer direct support for adding and managing dataset translations, it is still often used in the development process. Once you've opened a PBIX project file in Power BI Desktop, the dataset in the PBIX project is loaded into memory and becomes accessible to both the Tabular Editor and to custom code you write using the TOM.



In many scenarios, it makes sense to add translations directly to a dataset loaded into Power BI Desktop as you are building and preparing to ship a PBIX project file for the first time. Note that after a dataset has been deployed to the Power BI Service, it's still possible to access that datasets and manage translations using the exact same techniques. The screenshot below demonstrates how a connection is established to a workspace in the Power BI Service through the XMLA endpoint.



Note that access to a Power BI dataset running in the Power BI Service with the Tabular Editor or the Tabular Object Model (TOM) requires routing through the XMLA endpoint. It also requires that the dataset is in workspace in a dedicated capacity. In other words, you must ensure your workspace has a diamond when viewed in the Power BI Service. This requirement can be met with either Power BI Premium, Power BI Premium Per User or one of the A SKUs for the Power BI Embedded Service in Microsoft Azure.

Install the Tabular Editor

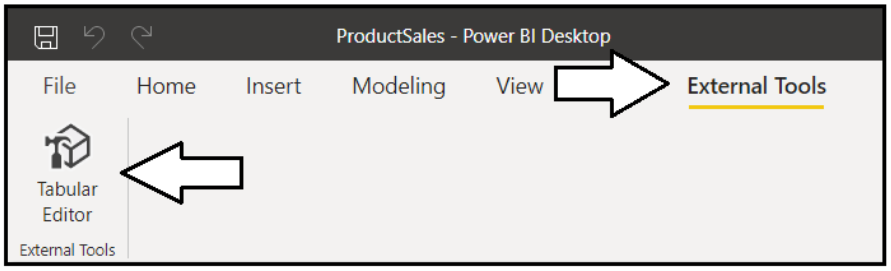
If you plan to work with dataset translations, you should become familiar working with the Tabular Editor. Even if you eventually plan to write your own TOM code to manage translations, working with the Tabular Editor is a great way to learn how translations are structured within a dataset. You can download the open source version of Tabular Editor (version 2) for free from the web page at following URL:

https://github.com/otykier/TabularEditor

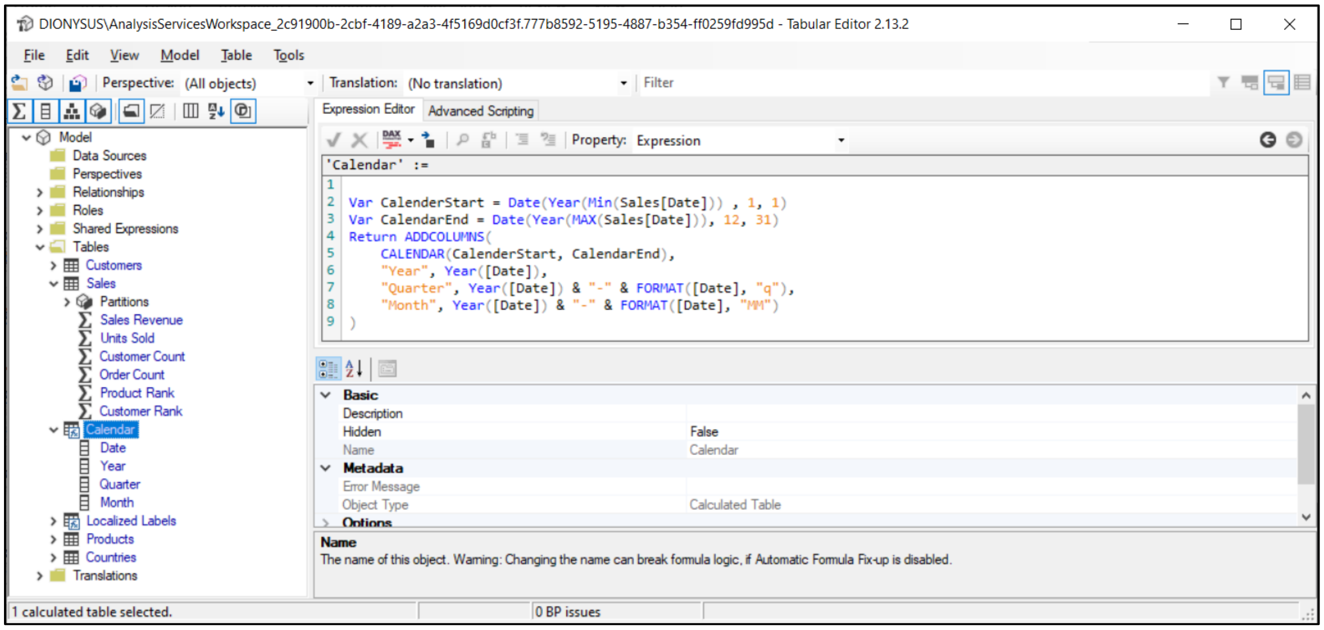
Providing first-class support for managing dataset translations is just one of the areas where the Tabular Editor shines. In addition to its support for working with translations, Tabular Editor provides a comprehensive toolset for advanced data modeling that go far beyond the data modeling support in Power BI Desktop. Tabular Editor is currently recognized by industry experts as the premiere tool for building and optimizing large-scale Power BI datasets.

The Tabular Editor was created and continues to be maintained by a very talented developer in the Power BI community named Daniel Otykier. While the open source version of Tabular Editor is available for free, Daniel has recently introduced Tabular Editor version 3 which is based on a paid licensing model. While the free version of Tabular Editor provides all the support you need to add and manage dataset translations, you must purchase Tabular Editor version 3 to take advantage of the newer advanced data modeling and dataset management features that are not available in the free version. You can get more information from the Tabular Editor website at https://tabulareditor.com.

Once you have installed Tabular Editor, Power BI Desktop will display a launch button for it on the **External Tools** tab in the ribbon. Clicking on the **Tabular Editor** button in Power BI Desktop will launch Tabular Editor and automatically open the data model for the current PBIX project file.



The Tabular Editor provides a user experience for viewing and modifying dataset objects such as tables, columns and measures. If you expand the **Tables** node in the left navigation, you can select a dataset object and then view its properties in a property sheet on the right. The following screenshot demonstrates selecting a calculated table named **Calendar** in the **ProductSales.pbix** developer sample. Once you have selected an object in the left navigation, you can view or modify any of its properties including the DAX expression used to create the calculated table.



```
'Calendar' :=

Var CalenderStart = Date(Year(Min(Sales[Date])) , 1, 1)
Var CalendarEnd = Date(Year(MAX(Sales[Date])), 12, 31)
Return ADDCOLUMNS(
    CALENDAR(CalenderStart, CalendarEnd),
    "Year", Year([Date]),
    "Quarter", Year([Date]) & "-" & FORMAT([Date], "q"),
    "Month", Year([Date]) & "-" & FORMAT([Date], "MM")
)
```

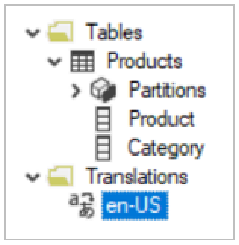
Description	
Hidden	False
Name	Calendar
Error Message	
Object Type	Calculated Table

Name
The name of this object. Warning: Changing the name can break formula logic, if Automatic Formula Fix-up is disabled.

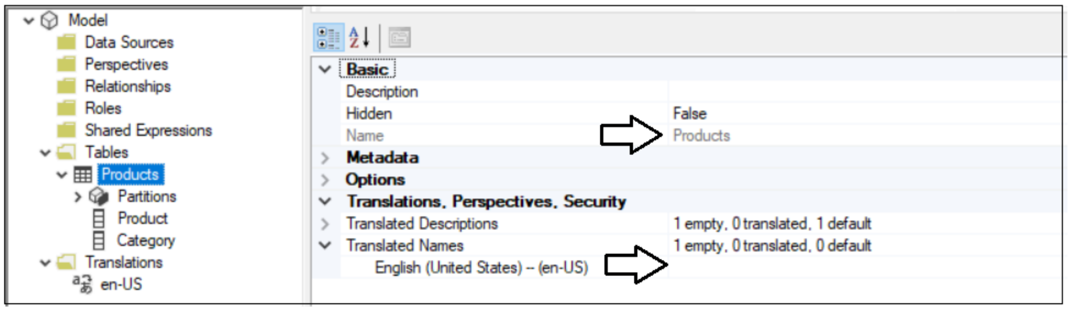
Add dataset translations using Tabular Editor

Behind the scenes, the Tabular Editor uses the Tabular Object Model (TOM) to create and manage dataset object and to access dataset object properties. Therefore, it's helpful to have a basic understanding of how TOM works. For each object, you can add translations using three different properties that are supported at the dataset object level. These properties are **Caption**, **Description** and **DisplayFolder**. The **Caption** property is used to add translations for the names of dataset objects. You can think of the **Caption** property of a dataset object as the display name which is seen by users.

Underneath the **Tables** node in the Tabular Editor, there is another top-level node named **Translations**. This node contains all the **Culture** objects which have been added to the current PBIX project. By default, every new PBIX project contains a single **Culture** object based on a default language and a default locale. The **ProductSales.pbix** developer sample has default **Culture** object based on **English (en)** as its language and the United **States (US)** as its locale. That's why this default culture is tagged with an identifier of **en-US**.

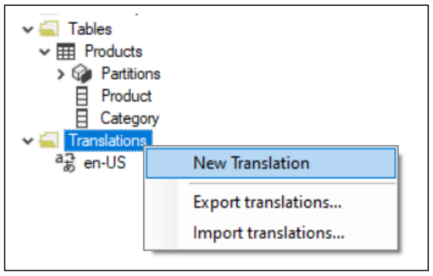


If you examine the property sheet in the Tabular Editor for a dataset object, you will see there is a **Translated Names** section which tracks the value of the **Caption** property. By default, all the translations for the default **Culture** object will have blank values. If you want to add translations manually, you can copy the **Name** property for a dataset object and then paste that text value into the default Culture in the **Translated Names** section.

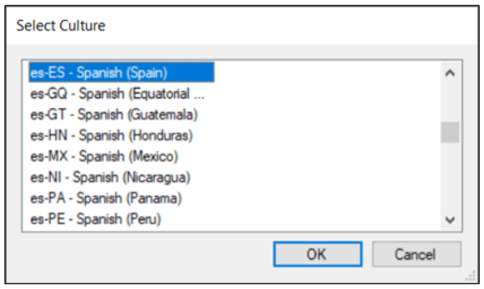


Basic	
Description	
Hidden	False
Name	Products
Metadata	
Options	
Translations, Perspectives, Security	
Translated Descriptions	1 empty, 0 translated, 1 default
Translated Names	1 empty, 0 translated, 0 default
English (United States) – (en-US)	

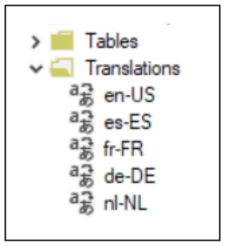
To add support for secondary languages, you must add one or more new **Culture** objects to the data model. This can be accomplished by right-clicking the top-level **Translations** node in Tabular Editor and then selecting the **New Translation** menu command.



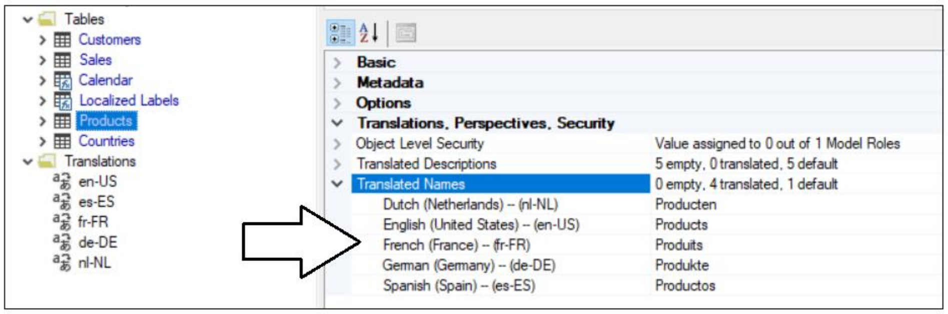
When you invoke the **New Translation** command, you'll be prompted with the **Select Culture** dialog which makes it possible to find and add a new **Culture** object for a specific language and locale.



The data model for the **ProductSales.pbix** developer sample has been extended with four secondary **Culture** objects to add translation support for in Spanish, French, German and Dutch.

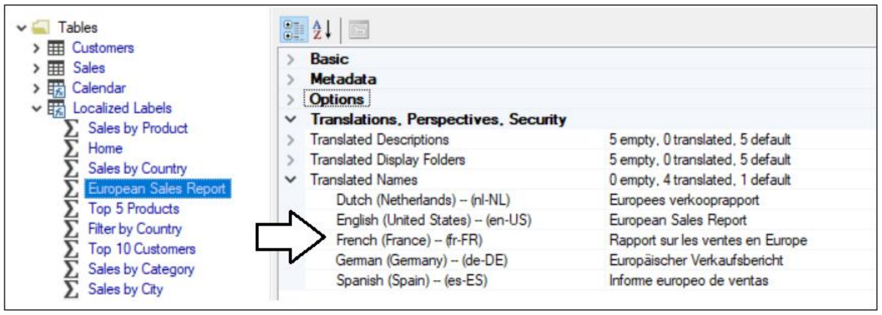


Once you have added **Culture** objects for the secondary languages, the **Translated Names** section of the property sheet for each database object will provide the ability to add translations for each language. The following screenshot shows how the dataset translations have been added to supply translated names for the **Products** table.



Object Level Security	Value assigned to 0 out of 1 Model Roles
Translated Descriptions	5 empty, 0 translated, 5 default
Translated Names	0 empty, 4 translated, 1 default
Dutch (Netherlands) -- (nl-NL)	Producten
English (United States) -- (en-US)	Products
French (France) -- (fr-FR)	Produits
German (Germany) -- (de-DE)	Produkte
Spanish (Spain) -- (es-ES)	Productos

The **ProductSales.pbix** developer sample also contains translations for all the measures in the **Localized Labels** table.

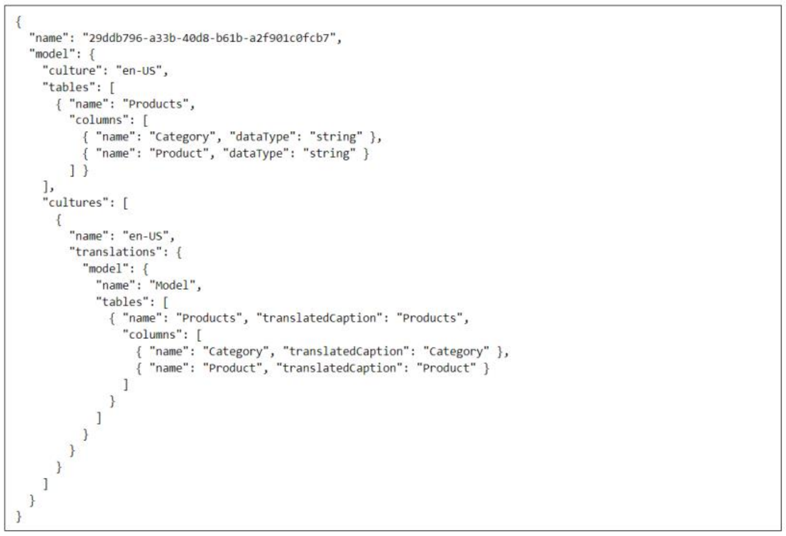


Translated Descriptions	5 empty, 0 translated, 5 default
Translated Display Folders	5 empty, 0 translated, 5 default
Translated Names	0 empty, 4 translated, 1 default
Dutch (Netherlands) -- (nl-NL)	Europees verkooprapport
English (United States) -- (en-US)	European Sales Report
French (France) -- (fr-FR)	Rapport sur les ventes en Europe
German (Germany) -- (de-DE)	Europäischer Verkaufsbericht
Spanish (Spain) -- (es-ES)	Informe europeo de ventas

The Tabular Editor provides a **Save As** command which can be used to save a data model definition in a JSON file format. By convention, a JSON file with a tabular dataset definition is created with a ***.bim** extension. You can build up your understanding of how translations work by saving a dataset definition as a ***.bim** file and then inspecting the JSON inside. Let's begin by examining the JSON for a simple a dataset definition which contains a table with two columns.

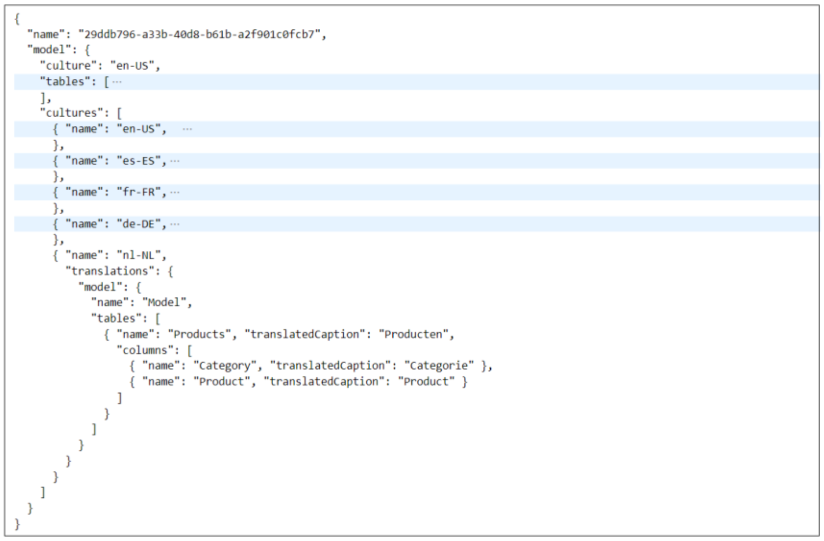
```
{
  "name": "29ddb796-a33b-40d8-b61b-a2f901c0fcb7",
  "model": {
    "culture": "en-US",
    "tables": [
      { "name": "Products",
        "columns": [
          { "name": "Category", "dataType": "string" },
          { "name": "Product", "dataType": "string" }
        ] }
    ],
    "cultures": [
      { "name": "en-US" }
    ]
  }
}
```

As you can see from the previous screenshot, this new PBIX project file has been created with a default Culture of en-US. However, the default Culture object is created without any translations inside. You must populate the default Culture object with translations yourself either by hand or by writing code to automate the process. Once you have populated the default Culture object with translations, you can see these translations are tracked on an object-by-object basis in the **tables** collection using the **translatedCaption** property.



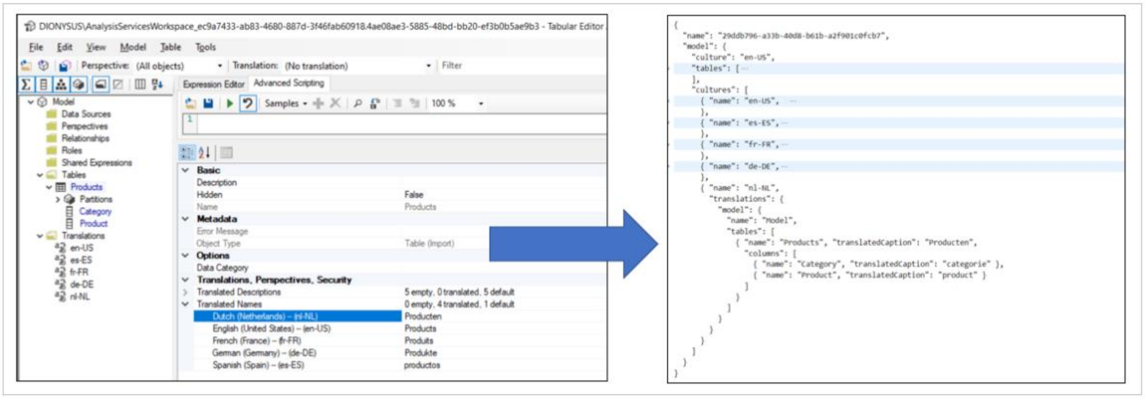
```
{
  "name": "29ddb796-a33b-40d8-b61b-a2f901c0fcb7",
  "model": {
    "culture": "en-US",
    "tables": [
      { "name": "Products",
        "columns": [
          { "name": "Category", "dataType": "string" },
          { "name": "Product", "dataType": "string" }
        ] }
    ],
    "cultures": [
      {
        "name": "en-US",
        "translations": {
          "model": {
            "name": "Model",
            "tables": [
              { "name": "Products", "translatedCaption": "Products",
                "columns": [
                  { "name": "Category", "translatedCaption": "Category" },
                  { "name": "Product", "translatedCaption": "Product" }
                ]
              }
            ]
          }
        }
      }
    ]
  }
}
```

As you begin to add translations for secondary languages, they're tracked in a separate Culture object in a similar fashion



```
{
  "name": "29ddb796-a33b-40d8-b61b-a2f901c0fcb7",
  "model": {
    "culture": "en-US",
    "tables": [⋯
    ],
    "cultures": [
      { "name": "en-US",  ⋯
      },
      { "name": "es-ES",⋯
      },
      { "name": "fr-FR",⋯
      },
      { "name": "de-DE",⋯
      },
      { "name": "nl-NL",
        "translations": {
          "model": {
            "name": "Model",
            "tables": [
              { "name": "Products", "translatedCaption": "Producten",
                "columns": [
                  { "name": "Category", "translatedCaption": "Categorie" },
                  { "name": "Product", "translatedCaption": "Product" }
                ]
              }
            ]
          }
        }
      }
    ]
  }
}
```

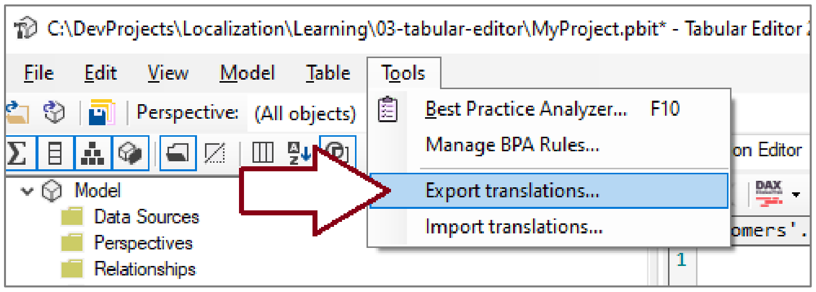
When you use the Tabular Editor to add translations by hand, it's just populating Culture objects behind the scenes.



Basic	
Description	
Hidden	False
Name	Products
Metadata	
Error Message	
Object Type	Table (Import)
Options	
Data Category	
Translations, Perspectives, Security	
Translated Descriptions	5 empty, 0 translated, 5 default
Translated Names	0 empty, 4 translated, 1 default
Dutch (Netherlands) - (nl-NL)	Producten
English (United States) - (en-US)	Products
French (France) - (fr-FR)	Produits
German (Germany) - (de-DE)	Produkte
Spanish (Spain) - (es-ES)	productos

```
{
  "name": "29ddb796-a33b-40d8-b61b-a2f901c0fcb7",
  "model": {
    "culture": "en-US",
    "tables": [⋯
    ],
    "cultures": [
      { "name": "en-US",  ⋯
      },
      { "name": "es-ES",⋯
      },
      { "name": "fr-FR",⋯
      },
      { "name": "de-DE",⋯
      },
      { "name": "nl-NL",
        "translations": {
          "model": {
            "name": "Model",
            "tables": [
              { "name": "Products", "translatedCaption": "Producten",
                "columns": [
                  { "name": "Category", "translatedCaption": "categorie" },
                  { "name": "Product", "translatedCaption": "product" }
                ]
              }
            ]
          }
        }
      }
    ]
  }
}
```

Tabular Editor supports exporting and importing translations using a JSON file format. This can be a useful feature when you need to integrate human translators into the development process. Imagine a scenario where you have just finished adding the tables, columns and measures to a data model in Power BI Desktop. You can begin by populating translations for default Culture and, after that, you can export the translations as a JSON file.



The JSON file with exported translations has the exact same JSON layout for Culture object as a **.bim** file. The difference is that the JSON file with exported translations omits any metadata from the data model definition that does not involve translations. The exported JSON file can then be extended with other Culture objects containing the translations for secondary languages. Once the JSON file has been extended, it can then be imported back into the original data model to extend it with the translations for secondary languages.

Add dataset translations using custom code

So far, you've seen that the Tabular Editor provides a way to add and manage translations by hand. While adding translations by hand with the Tabular Editor is a great way to learn about translations, it can become more tedious as the number of database objects requiring translations increases.

If the number of tables and fields in a data model is small, you can add translations by hand without any problems. But what happens when you're working with a large data model that contains 30 tables, 500 columns and 250 measures? It can take you 3-4 hours of tedious copy-and-paste operations just to populate the translations for the default Culture object. This provides a real motivation for learning how to write custom code with TOM that can automate the process.

Another important consideration when building multi-language reports involves the human aspect of translating text values from one language to another. While it's possible to initially generate the first round of translations using the Microsoft Translation service, you will eventually need to integrate carbon-based life forms (i.e. people) into the development process who can act as translators to generate quality translations. Furthermore, you cannot expect that people who work as language translators will be able to use an advanced data modeling tool like the Tabular Editor.

While it is technically possible to have human translators work on files generated by the Export Translations command of the Tabular Editor, the JSON-based format will likely be rejected by professional translators due to it being a non-standard format that is hard to work with. Once you begin writing custom code with TOM, however, you can generate translations files using what translation file format is required. If you are working with a professional translation team, you might be requires to generate translation files in a standard format such as RESX files or XLIFF files. You can also generate translations files in easy-to-use formats such as CSV files or XLSX files.

You can also automate the task of adding translations using the Tabular Object Model (TOM) which is an extension of Analysis Management Object (AMO) client library. You can read through Programming Datasets with the Tabular Object Model (TOM) if you need to learn the fundamentals about how to get started with this powerful API. In this article, we'll focus on using TOM to add and manage dataset translations. We'll begin by programming TOM using the Adavnced Scripting support in the Tabular Editor. After that, we'll examine creating a custom C# project in Visual Studio that can be configured as an External Tool for Power BI Desktop.

Program TOM using Advanced Scripting in Tabular Editor

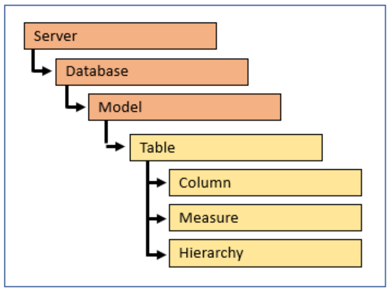
xsdidijiji

Program TOM in an External Tool for Power BI Desktop

Let's start with a simple TOM programming example which connects to a PBIX project file loaded into Power BI Desktop. After establishing a connection this code enumerates through the tables in a data model and prints the tabke name to the console window.

```
using System;
using Microsoft.AnalysisServices.Tabular;

class Program {

  const string connectString = "localhost:50000"; // update with port number for Power BI Desktop session

  static void Main() {

    Server server = new Server();
    server.Connect(connectString);

    Model model = server.Databases[0].Model;

    foreach(Table table in model.Tables) {
      Console.WriteLine("Table: " + table.Name);
    }

  }
}
```

ssssss



Within a table, you can further enumerate through its three main collections which includes

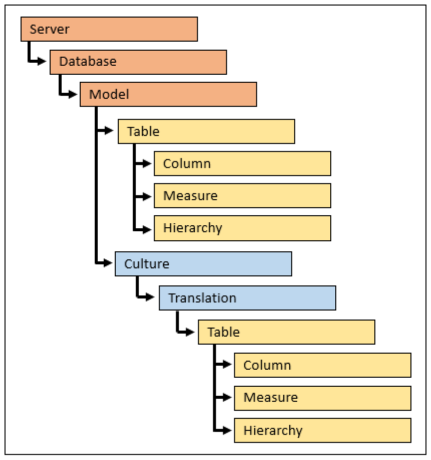
```
foreach (Table table in model.Tables) {
  Console.WriteLine("Table: " + table.Name);

  // enumerate through columns
  foreach (Column column in table.Columns) {
    Console.WriteLine("Column: " + column.Name);
  };

  // enumerate through measures
  foreach (Measure measure in table.Measures) {
    Console.WriteLine("Measure: " + measure.Name);
  };

  // enumerate through hierarchies
  foreach (Hierarchy hierarchy in table.Hierarchies) {
    Console.WriteLine("Hierarchy: " + hierarchy.Name);
  };
}
```

For

For each object, you can add translations for three properties which include **Caption**, **Description** and **DisplayFolder**.

You use the **Caption** property to add a translation for the name of an object such as a table, column, measure or hierarchy. If your data model uses display folders to organize columns and measures within tables, you need to add additional translation using the

When you create a new PBIX project, it has a default culture (en-US) but contains no translations. You must add translations for each spoken language.



Embed Reports with Specific Locales

Design and implement a content translation strategy

Setting the Language for Current User using RLS and UserCulture

ss