Gettings Started

Installation

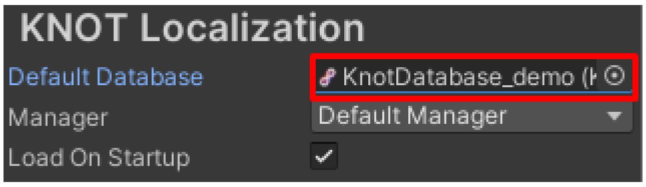
Download and import package. `KnotProjectSettings` asset will be created under your root `Assets` folder upon further interactions. You are able to move this asset to any subfolder in your project.

Creating Database

Open `Tools/KNOT Localization/Database Editor` window and create new `Database` asset as suggested.

> You can also create new Database via `Create/KNOT Localization/Database` project window context menu

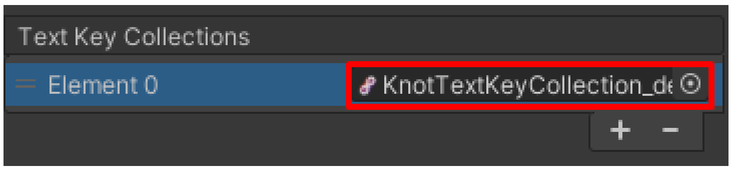
Mark your Database as Default in `Edit/Project Settings/KNOT/Localization`


KNOT Localization

Default Database	KnotDatabase_demo (I
Manager	Default Manager
Load On Startup	✓

Adding Key Collection

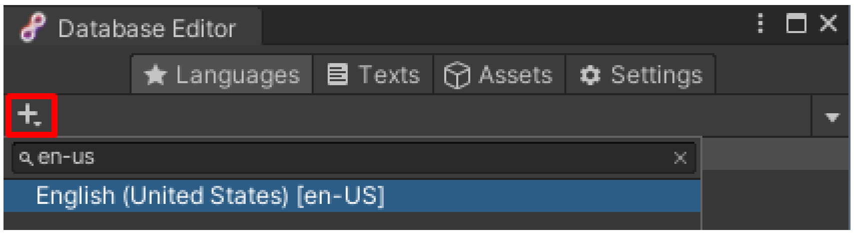
Open `Settings` tab, create and assign new `Text Key Collection` asset.


Text Key Collections

Element 0	KnotTextKeyCollection_de

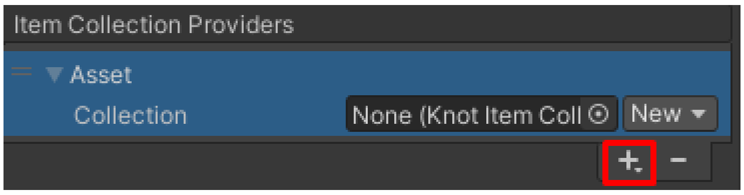
Adding Language

Open `Languages` tab. Click `+` to select language Culture Name and create new `Language`.



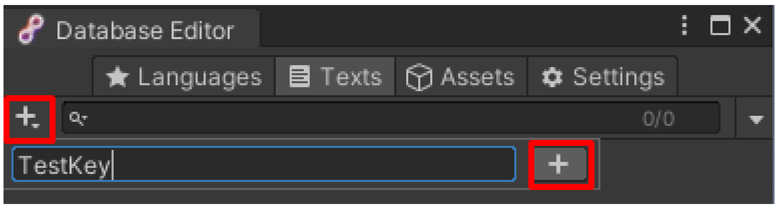
Adding Text Collection

Select newly created language and add `Asset` in `Item Collection Providers` list. Create and assign new `Text Collection` asset.

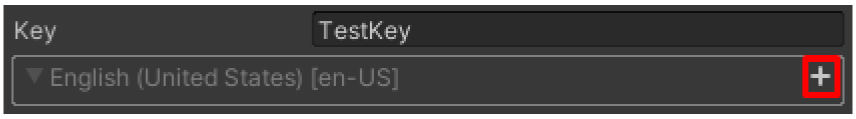


Adding Text Key

Open `Texts` tab and create new `Key`.

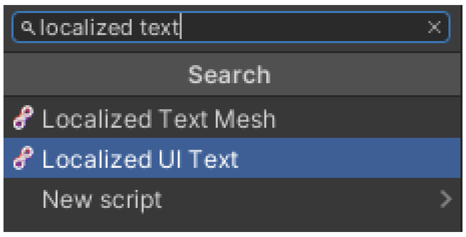


Select newly created key, click `+` to add localized value to the corresponding `Language` and type localized value.

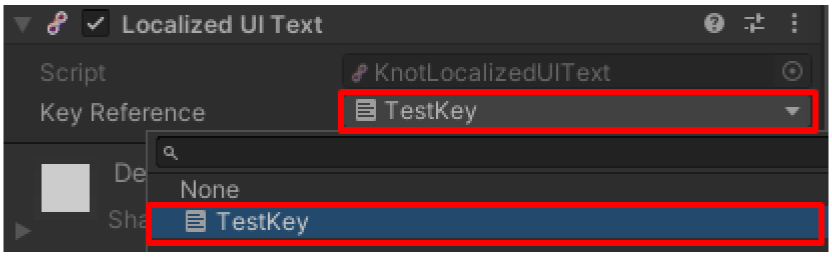


Testing

Make sure that your project has **Unity UI** package installed in Package Manager. Create new Game Object with Unity's native `Text` component and attach `Localized UI Text` component.



Assign previously created Key to `Key Reference` field.



Hit `Play`.

Runtime API

Changing Language

```
KnotLanguageData targetLanguage =
KnotLocalization.Manager.Languages.FirstOrDefault(d => d.SystemLanguage ==
SystemLanguage.English);
if (targetLanguage != null)
    KnotLocalization.Manager.LoadLanguage(targetLanguage);
```

Subscribing to Language Loaded callback

```
KnotLocalization.Manager.StateChanged += state =>
{
    if (state == KnotManagerState.LanguageLoaded)
    {
        //Selected or startup language is loaded
    }
};
```

Getting Text Value

```
string myLocalizedText = "myKey".Localize();

//or

string myLocalizedText = KnotLocalization.GetText("myKey");

//or

string myLocalizedText = KnotLocalization.Manager.GetTextValue("myKey").Value;

//or

KnotTextKeyReference myKeyRef = new KnotTextKeyReference("myKey");
string myLocalizedText = myKeyRef.Value;
```

Subscribing to Text Updated callback

```
void OnEnable()
{
    KnotLocalization.RegisterTextUpdatedCallback("myKey", TextUpdated);
}

void OnDisable()
{
    KnotLocalization.UnRegisterTextUpdatedCallback("myKey", TextUpdated);
}

void TextUpdated(string text)
```

```
{
    //Do something with localized text assigned to myKey
}
```

or

```
KnotTextKeyReference myKeyRef = new KnotTextKeyReference("myKey");
myKeyRef.ValueUpdated += text =>
{
    //Do something with localized text assigned to myKey
};
```

Accessing Metadata

Database

```
MyCustomMetadata myMetadata =
KnotLocalization.Manager.Database.Settings.Metadata.OfType<MyCustomMetadata>
().FirstOrDefault();
```

Selected Language

```
MyCustomMetadata myMetadata =
KnotLocalization.Manager.SelectedLanguage.Metadata.OfType<MyCustomMetadata>
().FirstOrDefault();
```

Text Key

```
MyCustomMetadata myMetadata =
KnotLocalization.Manager.GetTextValue("MyKey").Metadata.OfType<MyCustomMetadata>
().FirstOrDefault();
```

Creating simple Database from scratch at runtime

```
void Awake()
{
    KnotDatabase myDatabase = ScriptableObject.CreateInstance<KnotDatabase>();

    KnotLanguageData myLanguage = new KnotLanguageData(SystemLanguage.English);
    KnotTextCollection myTextCollection =
ScriptableObject.CreateInstance<KnotTextCollection>();
    myTextCollection.Add(new KnotTextData("myKey", "myText"));
    myLanguage.CollectionProviders.Add(new
KnotAssetCollectionProvider(myTextCollection));
    myDatabase.Languages.Add(myLanguage);

    KnotLocalization.Manager.SetDatabase(myDatabase);
    KnotLocalization.Manager.LoadLanguage(myLanguage);

    KnotLocalization.Manager.StateChanged += OnStateChanged;
}

void OnStateChanged(KnotManagerState state)
{
    if (state == KnotManagerState.LanguageLoaded)
    {
        Debug.Log(KnotLocalization.GetText("myKey")); //myText
    }
}
```

#Metadata

Metadata objects stores custom data and implements additional logic in localization pipeline.

Metadata can be part of three scopes:

- `Database`-wide metadata applies to all Languages and all Keys
- `Language`-wide metadata applies to all Keys for specific Language
- `Key` metadata affects only specific Key

> Editor-only Metadata will not be included in build

Implementing your own Metadata

You can define your own custom Metadata class by implementing `IKnotMetadata` interface.

Example:

```
[Serializable]
[KnotMetadataInfo("Prefix", KnotMetadataInfoAttribute.MetadataScope.Text,
AllowMultipleInstances = true)]
public class KnotPrefixMetadata : IKnotTextFormatterMetadata
{
    public string Prefix
    {
        get => _prefix;
        set => _prefix = value;
    }
    [SerializeField] private string _prefix;


    public KnotPrefixMetadata() { }
    public KnotPrefixMetadata(string prefix)
    {
        _prefix = prefix;
    }

    public object Clone() => new KnotPrefixMetadata(Prefix);

    public void Format(StringBuilder sb)
    {
        sb.Insert(0, Prefix);
    }

    public void Format(StringBuilder sb, CultureInfo cultureInfo) => Format(sb);
}
```

> [!NOTE] Custom IKnotMetadata implementation should have no constructor or at least one public constructor without arguments.
>
> If you decide to remove, change the name, namespace or assembly of custom class you will get serialization error and possibly lose data. As a temporary solution, add `[MovedFrom(false, null, "OldNamespaceName", "OldClassName")]` attribute to your class before making those changes.

Addons

TextMeshPRO

Adds `Localized Text Mesh PRO` and `Localized Text Mesh PRO (UI)` components.

> Requires com.unity.textmeshpro package installed

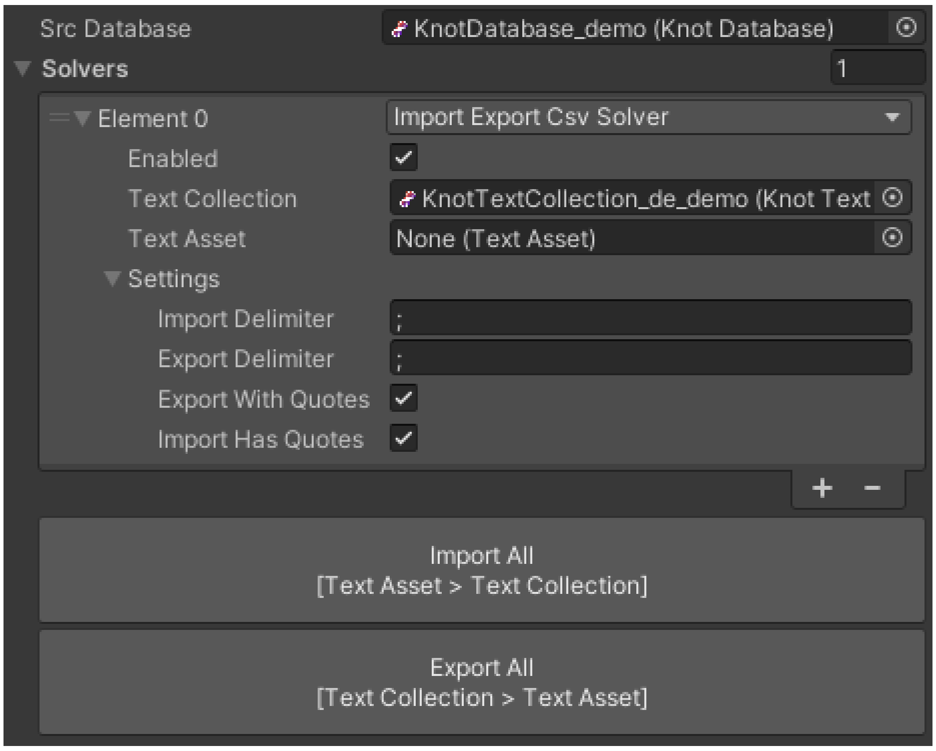
Addressables

`KnotAddressableCollectionProvider` lets you to assign Text or Asset Collection Provider as Asset Reference (either local or remote) to specific Language.

> Requires com.unity.addressables package installed

Import / Export (Experimental)

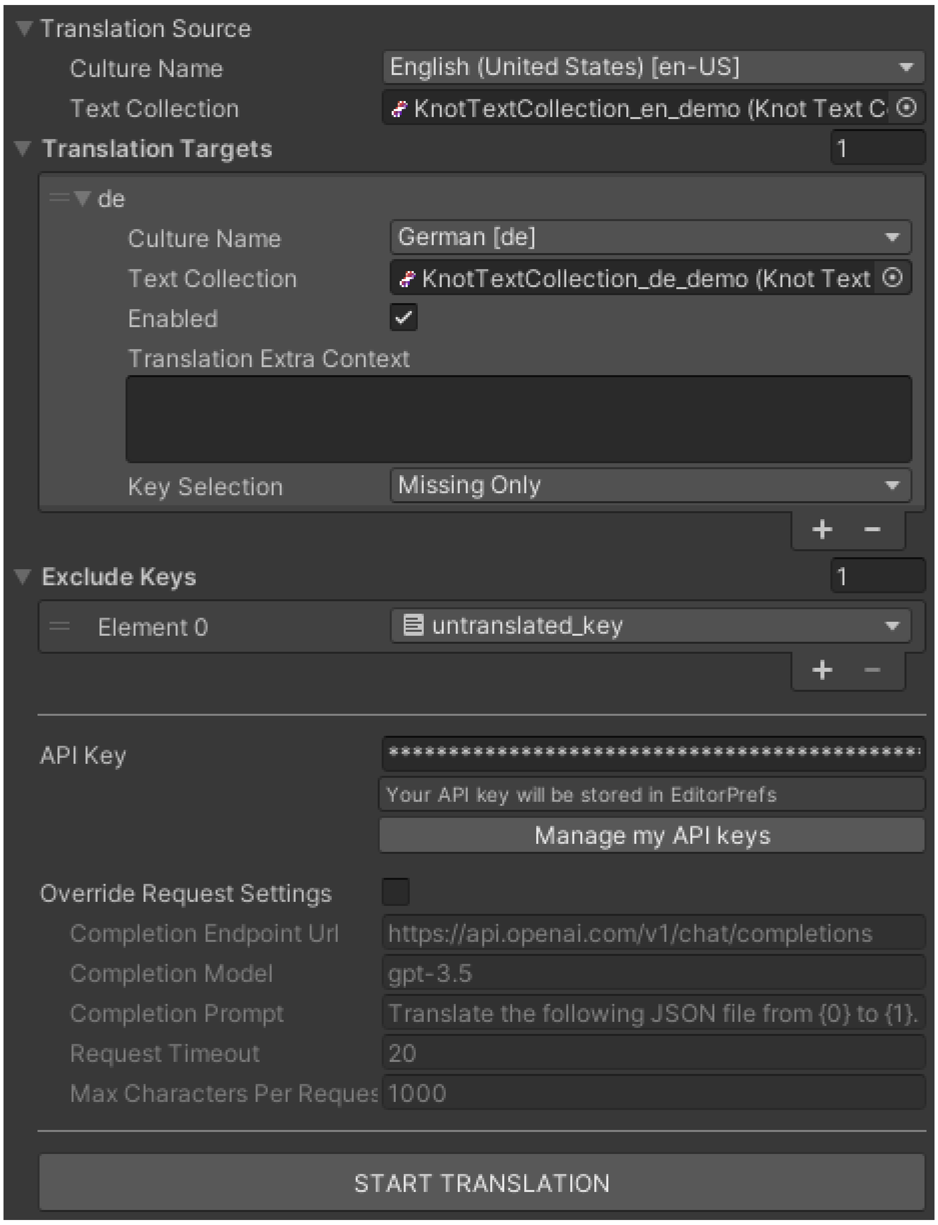
Transfer localization data between CSV and `KnotTextCollectionAsset`. You can use CSV Importer/Exporter by creating `KNOT/Localization/Addons/Import Export` asset.


Src Database: KnotDatabase_demo (Knot Database)
Solvers: 1
Element 0: Import Export Csv Solver
Enabled: ✓
Text Collection: KnotTextCollection_de_demo (Knot Text
Text Asset: None (Text Asset)
Settings
Import Delimiter: ;
Export Delimiter: ;
Export With Quotes: ✓
Import Has Quotes: ✓

+
−
Import All
[Text Asset > Text Collection]
Export All
[Text Collection > Text Asset]

OpenAI Autotranslator (Experimental)

Use OpenAI API for machine translation. You can use OpenAI Autotranslator by creating `KNOT/Localization/Addons/OpenAI Autotranslator Preset` asset.



Translation Source

Culture Name: English (United States) [en-US]

Text Collection: KnotTextCollection_en_demo (Knot Text C

Translation Targets: 1

de

Culture Name: German [de]

Text Collection: KnotTextCollection_de_demo (Knot Text

Enabled: ✓

Translation Extra Context

Key Selection: Missing Only

Exclude Keys: 1

Element 0: untranslated_key

API Key: ***

Your API key will be stored in EditorPrefs

Manage my API keys

Override Request Settings

Completion Endpoint Url: https://api.openai.com/v1/chat/completions

Completion Model: gpt-3.5

Completion Prompt: Translate the following JSON file from {0} to {1}.

Request Timeout: 20

Max Characters Per Reques: 1000

START TRANSLATION