Application Translations (bpy.app.translations)

This object contains some data/methods regarding internationalization in Blender, and allows every py script to feature translations for its own UI messages.

Introduction

Warning

Most of this object should only be useful if you actually manipulate i18n stuff from Python. If you are a regular add-on, you should only bother about `contexts` member, and the `register()`/`unregister()` functions! The `pgettext()` family of functions should only be used in rare specific cases (like e.g. complex "composited" UI strings…).

To add translations to your python script, you must define a dictionary formatted like that: `{locale: {msg_key: msg_translation, ...}, ...}` where:

- locale is either a lang iso code (e.g. `fr`), a lang+country code (e.g. `pt_BR`), a lang+variant code (e.g. `sr@latin`), or a full code (e.g. `uz_UZ@cyrilic`).
- msg_key is a tuple (context, org message) - use, as much as possible, the predefined `contexts`.
- msg_translation is the translated message in given language!

Then, call `bpy.app.translations.register(__name__, your_dict)` in your `register()` function, and `bpy.app.translations.unregister(__name__)` in your `unregister()` one.

The `Manage UI translations` add-on has several functions to help you collect strings to translate, and generate the needed python code (the translation dictionary), as well as optional intermediary po files if you want some… See How to Translate Blender and Using i18n in Blender Code for more info.

Module References

```
import bpy

# This block can be automatically generated by UI translations addon, which also handles c
# See also https://developer.blender.org/docs/handbook/translating/translator_guide/#trans
# It can (should) also be put in a different, specific py file.

# ##### BEGIN AUTOGENERATED I18N SECTION #####
# NOTE: You can safely move around this auto-generated block (with the begin/end markers!)
#       and edit the translations by hand.
#       Just carefully respect the format of the tuple!

# Tuple of tuples ((msgctxt, msgid), (sources, gen_comments), (lang, translation, (is_fuzz
translations_tuple = (
    (("*", ""),
     ((), ()),
     ("fr_FR", "Project-Id-Version: Copy Settings 0.1.5 (r0)\nReport-Msgid-Bugs-To: \nPOT-
               (False,
                ("Blender's translation file (po format).",
                 "Copyright (C) 2013 The Blender Foundation.",
                 "This file is distributed under the same license as the Blender package."
                 "FIRST AUTHOR <EMAIL@ADDRESS>, YEAR."))),
     ),
    (("Operator", "Render: Copy Settings"),
     (("bpy.types.SCENE_OT_render_copy_settings",),
      ()),
```

```
        ()),
      ("fr_FR", "Rendu: copier réglages",
                (False, ())),
      ),
     (("*", "Copy render settings from current scene to others"),
      (("bpy.types.SCENE_OT_render_copy_settings",),
       ()),
      ("fr_FR", "Copier les réglages de rendu depuis la scène courante vers d'autres",
                (False, ())),
      ),
     # ... etc, all messages from your addon.
)

translations_dict = {}
for msg in translations_tuple:
    key = msg[0]
    for lang, trans, (is_fuzzy, comments) in msg[2:]:
        if trans and not is_fuzzy:
            translations_dict.setdefault(lang, {})[key] = trans

# ##### END AUTOGENERATED I18N SECTION #####

# Define remaining addon (operators, UI...) here.


def register():
    # Usual operator/UI/etc. registration...

    bpy.app.translations.register(__name__, translations_dict)


def unregister():
    bpy.app.translations.unregister(__name__)

    # Usual operator/UI/etc. unregistration...
```

bpy.app.translations.**locale**

The actual locale currently in use (will always return a void string when Blender is built without internationalization support).

bpy.app.translations.**locales**

All locales currently known by Blender (i.e. available as translations).

bpy.app.translations.**contexts_C_to_py**

A readonly dict mapping contexts' C-identifiers to their py-identifiers.

bpy.app.translations.**contexts**

Constant value bpy.app.translations.contexts(default_real=None, default='*', operator_default='Operator', ui_events_keymaps='UI_Events_KeyMaps', plural='Plural', id_action='Action', id_armature='Armature', id_brush='Brush', id_cachefile='CacheFile', id_camera='Camera', id_collection='Collection', id_curves='Curves', id_curve='Curve', id_fs_linestyle='FreestyleLineStyle', id_gpencil='GPencil', id_id='ID', id_image='Image', id_lattice='Lattice', id_library='Library', id_light='Ligh id_lightprobe='LightProbe', id_mask='Mask', id_material='Material', id_mesh='Mesh', id_metaball='Metaball', id_movieclip='MovieClip', id_nodetree='NodeTree', id_object='Object', id_paintcurve='PaintCurve', id_palette='Palette', id_particlesettings='ParticleSettings', id_pointcloud='PointCloud', id_scene='Scene', id_screen='Screen', id_sequence='Sequence', id_shapekey='Key', id_simulation='Simulation', id_sound='Sound', id_speaker='Speaker', id_text='Text', id_texture='Texture', id_vfont='VFont', id_volume='Volume', id_windowmanager='WindowManager', id_workspace='WorkSpace', id_world='World', editor_filebrowser='File browser',

editor_python_console='Python console', editor_preferences='Preferences', editor_view3d='View3D', amount='Amount', color='Color', constraint='Constraint', modifier='Modifier', navigation='Navigation', render_layer='Render Layer', time='Time', unit='Unit')

bpy.app.translations.**locale_explode(locale)**

Return all components and their combinations of the given ISO locale string.

```
>>> bpy.app.translations.locale_explode("sr_RS@latin")
("sr", "RS", "latin", "sr_RS", "sr@latin")
```

For non-complete locales, missing elements will be None.

PARAMETERS:

locale – The ISO locale string to explode.

RETURNS:

A tuple `(language, country, variant, language_country, language@variant)`.

bpy.app.translations.**pgettext(msgid, msgctxt=None)**

Try to translate the given msgid (with optional msgctxt).

Note

The `(msgid, msgctxt)` parameters order has been switched compared to gettext function, to allow single-parameter calls (context then defaults to BLT_I18NCONTEXT_DEFAULT).

Note

You should really rarely need to use this function in regular addon code, as all translation should be handled by Blender internal code. The only exception are string containing formatting (like 'File: %r'), but you should rather use `pgettext_iface()`/`pgettext_tip()` in those cases!

Note

Does nothing when Blender is built without internationalization support (hence always returns `msgid`).

PARAMETERS:

- **msgid** (*str*) – The string to translate.
- **msgctxt** (*str* | *None*) – The translation context (defaults to BLT_I18NCONTEXT_DEFAULT).

RETURNS:

The translated string (or msgid if no translation was found).

bpy.app.translations.**pgettext_data(msgid, msgctxt=None)**

Try to translate the given msgid (with optional msgctxt), if new data name's translation is enabled.

Note

See `pgettext()` notes.

PARAMETERS:

- **msgid** (*str*) – The string to translate.
- **msgctxt** (*str* | *None*) – The translation context (defaults to BLT_I18NCONTEXT_DEFAULT).

RETURNS:

The translated string (or `msgid` if no translation was found).

bpy.app.translations.**pgettext_iface(msgid, msgctxt=None)**

Try to translate the given msgid (with optional msgctxt), if labels' translation is enabled.

Note

See `pgettext()` notes.

PARAMETERS:

- **msgid** (*str*) – The string to translate.
- **msgctxt** (*str* | *None*) – The translation context (defaults to BLT_I18NCONTEXT_DEFAULT).

RETURNS:

The translated string (or msgid if no translation was found).

bpy.app.translations.**pgettext_n(msgid, msgctxt=None)**

Extract the given msgid to translation files. This is a no-op function that will only mark the string to extract, but not perform the actual translation.

Note

See `pgettext()` notes.

PARAMETERS:

- **msgid** (*str*) – The string to extract.
- **msgctxt** (*str* | *None*) – The translation context (defaults to BLT_I18NCONTEXT_DEFAULT).

RETURNS:

The original string.

bpy.app.translations.**pgettext_rpt(msgid, msgctxt=None)**

Try to translate the given msgid (with optional msgctxt), if reports' translation is enabled.

Note

See `pgettext()` notes.

PARAMETERS:

- **msgid** (*str*) – The string to translate.
- **msgctxt** (*str* | *None*) – The translation context (defaults to BLT_I18NCONTEXT_DEFAULT).

RETURNS:

The translated string (or msgid if no translation was found).

bpy.app.translations.**pgettext_tip(msgid, msgctxt=None)**

Try to translate the given msgid (with optional msgctxt), if tooltips' translation is enabled.

Note

See `pgettext()` notes.

PARAMETERS:

- **msgid** (*str*) – The string to translate.
- **msgctxt** (*str* | *None*) – The translation context (defaults to BLT_I18NCONTEXT_DEFAULT).

RETURNS:

The translated string (or msgid if no translation was found).

bpy.app.translations.**register(module_name, translations_dict)**

Registers an addon's UI translations.

Note

Does nothing when Blender is built without internationalization support.

PARAMETERS:

- **module_name** (*str*) – The name identifying the addon.
- **translations_dict** (*dict[str, dict[str, str]]*) – A dictionary built like that: `{locale: {msg_key: msg_translation, ...} ...}`

bpy.app.translations.**unregister(module_name)**

Unregisters an addon's UI translations.

Note

Does nothing when Blender is built without internationalization support.

PARAMETERS:

module_name (*str*) – The name identifying the addon.

Copyright © Blender Authors
Made with Furo