Language-Independent Python Programs Through XML-Based Text Management

A Concept for GIMP Plugins and Other Python Applications

Concept Description with Technical Appendix | Version: November 9, 2026

Core principle: Language determines what the user sees. The identifier determines what the program does.

1. Purpose and Core Principle

Supporting multiple languages should not require changes to program functions for individual languages. External XML language files therefore contain all user-facing text. The Python code, by contrast, uses fixed, language-independent identifiers (IDs).

Identifiers are never translated or evaluated by comparing their displayed text. This separation is especially important for selection fields: the interface displays translated labels, while the program logic receives the corresponding immutable ID.

2. Components of the Language System

Component	Responsibility
hnGebietsschema.py	Determines locale and language code through the Python locale module.
hnXMLParser.py	Reads a simple XML structure and passes recognized elements to a callback.
hnDiktionaer.py	Processes String, Liste and IdListe; provides central storage and access.
XML language files	Language-specific content for the relevant plugin.

Language files may be named HN-Form-zeichnen_DE.xml and HN-Form-zeichnen_EN.xml, for example. The calling plugin selects the actual filename; the three library classes do not perform this selection on their own.

3. Structure of a Language File

```
<?xml version="1.0" encoding="utf-8"?>
<Uebersetzung version="2026-11-09">
  <Element id="ZahlEingabe">
    <String>Bitte eine Ganzzahl eingeben:</String>
  </Element>
  <Element id="Textfarben">
    <Liste trenner=" | ">Vordergrund | Hintergrund</Liste>
  </Element>
  <Element id="Pfadanfang"><String>Am Pfadanfang</String></Element>
  <Element id="Pfadende"><String>Am Pfadende</String></Element>
  <Element id="Positionen">
    <IdListe trenner=" | ">Pfadanfang | Pfadende</IdListe>
  </Element>
</Uebersetzung>
```

The id attribute connects each text entry to the Python code. All language files must provide the IDs required by the program. The text contents are translated, not the identifiers.

4. The Three Data Types

4.1 String — One Text Value

```
<Element id="ZahlEingabe">
  <String>Bitte eine Ganzzahl eingeben:</String>
</Element>

# Dictionary result:
dikt["ZahlEingabe"] = "Bitte eine Ganzzahl eingeben:"
```

A translated language file can provide different text for the same identifier. Python still retrieves it by the same ID.

4.2 Liste — Multiple Display Texts

```
<Element id="Textfarben">
  <Liste trenner=" | ">Vordergrund | Hintergrund</Liste>
</Element>

# Dictionary result:
dikt["Textfarben"] = ["Vordergrund", "Hintergrund"]
```

The separator is specified by trenner; if omitted, the default is " | ". After splitting, leading and trailing whitespace is removed from each list element. A plain Liste has no individual IDs for its choices. If program logic relies on indices, the meaning assigned to each position must remain consistent across all language files.

4.3 IdListe — Identifiers and Display Texts Together

```
<Element id="Pfadanfang"><String>Am Pfadanfang</String></Element>
<Element id="Pfadende"><String>Am Pfadende</String></Element>
<Element id="Positionen">
  <IdListe>Pfadanfang | Pfadende</IdListe>
</Element>

# Two corresponding dictionary lists:
dikt["Positionen"]  = ["Pfadanfang", "Pfadende"]
dikt["_Positionen"] = ["Am Pfadanfang", "Am Pfadende"]
```

The ID list contains immutable keys. A second list of the corresponding translated contents is stored under an automatically generated key prefixed with an underscore. Both lists have the same order; each index links one display label to its identifier.

Index	Internal ID	Display text (DE)	Display text (EN)
0	Pfadanfang	Am Pfadanfang	At the start of the path
1	Pfadende	Am Pfadende	At the end of the path

An IdListe is resolved immediately while the file is being read. At that point, the referenced IDs must already exist in the dictionary, either because they appeared earlier in the XML or because they were added as default entries.

5. Language-Independent Program Execution

The selection field receives the translated labels. Subsequent processing uses the selected index to look up the corresponding language-independent ID. The following fragment illustrates the principle without assuming a specific GTK widget API:

```
anzeigetexte = diktObj.wert("_Positionen")
# Populate the GTK selection field with anzeigetexte ...
auswahl_index = 1  # Example: the second item
auswahl_id = diktObj.wert("Positionen")[auswahl_index]

if auswahl_id == "Pfadanfang":
    pass  # Logic for the start of the path
elif auswahl_id == "Pfadende":
    pass  # Logic for the end of the path
```

The program is controlled neither by the wording "Am Pfadende" nor by "At the end of the path", but solely by the fixed identifier Pfadende. The same approach works in GIMP, GTK, and other Python interfaces.

6. Creating an Additional Language File

A translator can copy an existing XML file and translate its visible text into the target language. The Python files do not need changes, provided the calling program can select and load the additional language file.

- Element IDs, XML tags, and attribute names remain unchanged.
- The contents of String entries may be translated.
- For Liste, translate the visible entries while preserving the separator, number of entries, and meaning of each position.
- IdListe contents consist of identifiers and must not be translated.
- IDs referenced by an IdListe must already be available when it is processed.
- Save the file in UTF-8 encoding.

7. Error Handling and Initial Values

Error messages may be needed before an external language file can be read. For this reason, the library classes contain predefined German and English messages. They select a version based on the language code obtained through hnGebietsschema; within these classes, all codes other than de use the English version.

Before reading, hnDiktionaer.lese() saves the existing dictionary contents. If an XMLParsFehler is caught, it restores those values and returns the error description; on successful completion, it returns an empty string. This does not guarantee recovery from every possible Python runtime error.

8. Conclusion

XML language files provide the display content, while Python logic operates on fixed IDs. In particular, IdListe connects freely translatable selection fields with language-independent processing. New translations can therefore be added as data without rewriting drawing functions for each language.

Language determines what the user sees. The identifier determines what the program does.

Technical Appendix

The information below describes the behavior of the three original modules provided. It is not intended as a general statement about every XML parser or Python application.

A. Actual API of hnDiktionaer

Method	Behavior
lese(dateipfad)	Returns "" on success or the error description if XMLParsFehler is caught.
wert(schluessel)	Returns the dictionary value; raises DiktionaerFehler if the key is missing.
existiert(schluessel)	Checks whether a key exists.
index(schluessel, element)	Returns an index; also returns 0 if the key or element is missing.
erweitere(dictionary)	Adds or overwrites entries using update().
addiere(schluessel, wert)	Adds or replaces an entry; strips leading/trailing whitespace from the key.
leere()	Removes all entries.

Caution with index(): 0 is both a valid index for the first list element and the return value when an entry is not found. If these cases must be distinguished, check existiert() or inspect the list contents first.

B. Parsing and Reference Resolution

hnXMLParser.parse(dateipfad, funktion) calls a callback for detected opening tags, supplying the tag name, attribute name, attribute value, and element content. hnDiktionaer passes its internal method __verarbXMLElement() for this purpose.

The required first root tag is Uebersetzung. hnDiktionaer handles the tags Element, String, Liste, and IdListe. Reading an IdListe creates both the ID list and a second list of resolved contents, stored under the underscore-prefixed key.

C. Parser Scope and Character Handling

hnXMLParser is a custom implementation for a restricted subset of XML, not a complete general-purpose XML processor. Among other limitations, it supports at most one attribute per tag. The XML file is read using utf-8-sig, and Windows CRLF line endings are converted to LF.

Input in XML text	Processing by __holeInhalt()
\n	LF, Unicode U+000A
\t	Tab, Unicode U+0009
<	<
>	>
&	&

In hnDiktionaer, Liste has additional replacement statements using chr(13) and chr(9). However, hnXMLParser has already converted the written sequences \n and \t into actual control characters, so these later replacements generally do not affect those original sequences in the normal data flow. The original documentation label #x13 does not denote the Unicode newline code point: hexadecimal U+0013 denotes DC3; decimal 13 corresponds to U+000D (CR).

The parser explicitly replaces the named sequences <, >, and &. The provided module does not implement general decoding of all XML character references (such as numeric references).

D. Locale and Language-File Selection

hnGebietsschema.holeSprachcode() uses locale.getlocale(), extracts the language portion before an underscore, and returns a lowercase code. If locale.getlocale() supplies no locale name, it returns en. For internal error messages, the dictionary and parser classes use German only for de and English otherwise.

The setze() method uses locale.setlocale(locale.LC_ALL, ...) to modify the global locale of the Python process. Loading a particular plugin language file is handled by program logic outside these three modules.

E. Guidance for Reuse

- Use only the XML structure explicitly supported by these classes; do not assume full XML standards compliance.
- Treat IDs as a stable programming interface and never translate them.
- Avoid forward references in IdListe: provide the required IDs before using them.
- Choose separators for Liste precisely and do not use them inside list entries.
- Handle the DiktionaerFehler exception when wert() is called with a missing key.
- When adapting to another environment, account for imports from HN_Bibliothek and the locale dependency.

F. Complete, Tested Python Example

The following console example demonstrates the language system without a GIMP or GTK interface. It was executed with the three unchanged modules supplied and the XML files shown below. The displayed text varied with the selected language, while the identifiers remained identical.

F.1 Directory Structure

```
Beispiel/
  demo.py
  demo_DE.xml
  demo_EN.xml
  HN_Bibliothek/
    __init__.py
    hnDiktionaer.py
    hnXMLParser.py
    hnGebietsschema.py
```

The __init__.py file may be empty. The three library modules are supplied as copies; the original GIMP project does not need to be changed.

F.2 German Language File demo_DE.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Uebersetzung version="2026-11-09">
  <Element id="Frage"><String>Wähle die Position:</String></Element>
  <Element id="Pfadanfang"><String>Am Pfadanfang</String></Element>
  <Element id="Pfadende"><String>Am Pfadende</String></Element>
  <Element id="Positionen"><IdListe>Pfadanfang | Pfadende</IdListe></Element>
</Uebersetzung>
```

F.3 English Language File demo_EN.xml

```
<?xml version="1.0" encoding="utf-8"?>
<Uebersetzung version="2026-11-09">
  <Element id="Frage"><String>Choose the position:</String></Element>
  <Element id="Pfadanfang"><String>At the start of the path</String></Element>
  <Element id="Pfadende"><String>At the end of the path</String></Element>
  <Element id="Positionen"><IdListe>Pfadanfang | Pfadende</IdListe></Element>
</Uebersetzung>
```

F.4 Program demo.py

```
import sys
from pathlib import Path
from HN_Bibliothek import hnDiktionaer

sprache = sys.argv[1].upper() if len(sys.argv) > 1 else "DE"
if sprache not in ("DE", "EN"):
    raise SystemExit("Unterstützte Sprachcodes: DE, EN")

ordner = Path(__file__).resolve().parent
datei = ordner / f"demo_{sprache}.xml"
diktObj = hnDiktionaer.hnDiktionaer()
fehler = diktObj.lese(str(datei))
if fehler:
    raise SystemExit(str(fehler))

print(diktObj.wert("Frage"))
texte = diktObj.wert("_Positionen")
for nummer, text in enumerate(texte, start=1):
    print(f"{nummer}: {text}")

nummer = int(input("1 / 2: "))
if nummer not in (1, 2):
    raise SystemExit("Ungültige Auswahl")

kennung = diktObj.wert("Positionen")[nummer - 1]
print("Interne Kennung:", kennung)
if kennung == "Pfadanfang":
    print("Aktion: Startpunkt verarbeiten")
elif kennung == "Pfadende":
    print("Aktion: Endpunkt verarbeiten")
```

Hard-coded texts for command-line control, input validation, and demonstration output are retained here to match the tested example program. They simplify the example; in a fully localized plugin, these user-facing messages would also come from the language system.

F.5 Running the Example and Interpreting Its Output

```
python demo.py DE
python demo.py EN
```

Selecting the second position in either language gives the same internal identifier, Pfadende. Only the displayed choices differ. The example deliberately uses a simple console selection; a GTK selection widget provides an index that can be resolved in the same way using diktObj.wert("Positionen").

F.6 Limitations of the Example

The example selects the language file using a command-line argument, not automatically through locale. This isolates the data processing and language-independent selection mechanisms. The hnGebietsschema, hnXMLParser, and hnDiktionaer classes remain identical to the source code supplied. The console does not execute actual GIMP drawing operations.