# Translation Strings Policy

This document provides guidelines for internationalization of the Bitcoin Core software.

## How to translate?

To mark a message as translatable

- In GUI source code (under `src/qt`): use `tr("...")`
- In non-GUI source code (under `src`): use `_("...")`

No internationalization is used for e.g. developer scripts outside `src`.

## Strings to be translated

On a high level, these strings are to be translated:

- GUI strings, anything that appears in a dialog or window

### GUI strings

Do not translate technical or extremely rare errors. Anything else that appears to the user in the GUI is to be translated. This includes labels, menu items, button texts, tooltips and window titles. This includes messages passed to the GUI through the UI interface through `InitMessage`, `ThreadSafeMessageBox` or `ShowProgress`.

## General recommendations

### Avoid unnecessary translation strings

Try not to burden translators with translating messages that are e.g. slight variations of other messages. In the GUI, avoid the use of text where an icon or symbol will do. Make sure that placeholder texts in forms do not end up in the list of strings to be translated (use `<string notr="true">`).

### Make translated strings understandable

Try to write translation strings in an understandable way, for both the user and the translator. Avoid overly technical or detailed messages.

### Do not translate internal errors

Do not translate internal errors, log messages, or messages that appear on the RPC interface. If an error is to be shown to the user, use a translatable generic message, then log the detailed message to the log. E.g., "A fatal internal error occurred, see debug.log for details". This helps troubleshooting; if the error is the same for everyone, the likelihood is increased that it can be found using a search engine.

### Avoid fragments

Avoid dividing up a message into fragments. Translators see every string separately, so they may misunderstand the context if the messages are not self-contained.

### Avoid HTML in translation strings

There have been difficulties with the use of HTML in translation strings; translators should not be able to accidentally affect the formatting of messages. This may sometimes be at conflict with the recommendation in the previous section.

### Plurals

Plurals can be complex in some languages. A quote from the gettext documentation:

```
In Polish we use e.g. plik (file) this way:
1 plik,
2,3,4 pliki,
5-21 pliko'w,
22-24 pliki,
25-31 pliko'w
and so on
```

In Qt code, use tr's third argument for optional plurality. For example:

```
tr("%n hour(s)","",secs/HOUR_IN_SECONDS);
tr("%n day(s)","",secs/DAY_IN_SECONDS);
tr("%n week(s)","",secs/WEEK_IN_SECONDS);
```

This adds `<numerusform>`s to the respective `.ts` file, which can be translated separately depending on the language. In English, this is simply:

```
<message numerus="yes">
    <source>%n active connection(s) to Bitcoin network</source>
    <translation>
        <numerusform>%n active connection to Bitcoin network</numerusform>
        <numerusform>%n active connections to Bitcoin network</numerusform>
    </translation>
</message>
```

Where possible, try to avoid embedding numbers into the flow of the string at all. E.g.,

```
WARNING: check your network connection, %d blocks received in the last %d hours (%d
expected)
```

versus

```
WARNING: check your network connection, less blocks (%d) were received in the last %n
hours than expected (%d).
```

The second example reduces the number of pluralized words that translators have to handle from three to one, at no cost to comprehensibility of the sentence.

### String freezes

During a string freeze (often before a major release), no translation strings are to be added, modified or removed.

This can be checked by executing `make translate` in the `src` directory, then verifying that `bitcoin_en.ts` remains unchanged.