

BBJabber Translation Utility

April 2009

By

Ralph Lance
BASIS Europe Distribution GmbH
rlance@basis-europe.eu

[Introduction](#)

[Inside BBJabber](#)

[Instantiating a Translator Object](#)

[Editing Property Files](#)

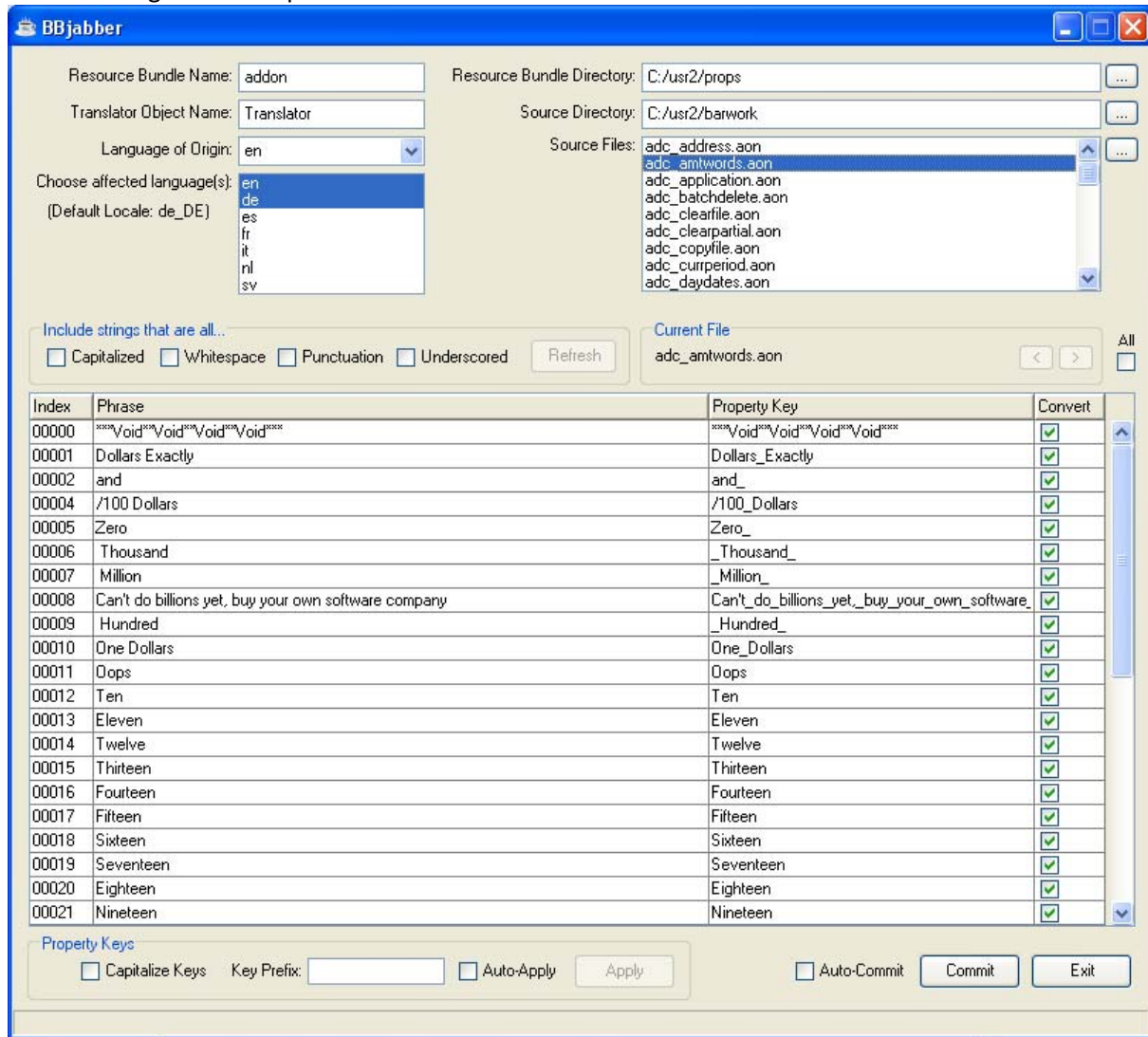
[Example Program for Translating Resource Files](#)

Introduction

[\(back to top\)](#)

BBJabber is a BBj® utility found under <bbj-install>/utils/translations/bbjabber.bbj. It uses simple rules to help increase the processing speed of finding hard-coded strings in existing resource and non-protected program source files, building a resource key, storing them in resource bundles, and replacing the strings in the program source code with a call to `Translator!.getTranslation()`, by default. It uses the classes found in the companion translation utility `bbtranslator.bbj` to add entries to properties files and to sort and save those files. For more information, review [BBTranslator](#) in the BASIS online help.

The following is an example of BBJabber:



Resource Bundle Name The root name of the property file, for example, addon creates the default property file `addon.properties`.

Translator Object Name The name of the object variable instantiated with `BBTranslator::getInstance()` and used in programs with the

getTranslation() method. If left empty, then it uses Translator and Translator!.getTranslation() to wrap the translation keys.

Language of Origin The language of origin represents the language of the original text used in the resource and program source code. It defaults to the language code of the client running BBJabber based on the client's locale and, like the default property file, is always included when writing out to the property files.

Affected Languages The list of affected languages is built depending on the bbjabber[_xx].properties files BBJabber finds. It represents the list of supported languages that BBTranslator returns.

Default Locale The current locale is that of the client as information.

Resource Bundle Directory This is the directory in which the property files are built.

Source Directory Resource and/or program source files found in this directory will be automatically loaded in the "Source Files" list box.

Note: Because program sources will be changed when committed, it is advisable to work with copies of original sources.

Source Files This is the list of the files to be processed:

Include strings that are all... By default, no strings that are composed completely of one of the following are included:

- all capitalized
- have only whitespace or punctuation characters
- have text segments bound with underscore(s), eg. ipe_table_name
- numbers or only one character (no option, done internally)

After changing any of these options, the [Refresh] button can be used to refresh the list of phrases found.

Current File The current file being processed. If more than one source file is highlighted, then you can use the navigation buttons [<] and [>] to go to the previous or next file name, respectively.

All If checked, then all of the source files are selected and the first file is loaded. In conjunction with the Auto Commit option, one can work through a directory of files very quickly.

Index (Column) The index of the internally held vector of all phrases (that means, being shown in the grid or not).

Phrase (Column) This is the original string as found in the resource or program source code.

Property Key (Column) BBJabber examines the original text and builds a default property key. (See "Capitalize Keys" and "Key Prefix" below)

By convention, property keys don't contain blanks, but use underscores. This is what the localization tools in the BASIS IDE and BBJabber do, by default. Property keys may contain blanks, but they'll be escaped with backslashes in the property file, as with ':' and '='.

Convert (Column)	An editable indication of whether to store the property and, if not a resource file, convert the original text in the program to use the <code>getTranslation()</code> method with the property key.
Capitalize Keys	Convert all property keys to uppercase (also in the internally-held vector of all phrases).
Key Prefix	A sequence of characters that are to be pre-appended to all property keys, e.g. <code>AON_</code> .
Auto-Apply	This checkbox dictates whether the <code>Capitalize Keys</code> and <code>Key Prefix</code> should be applied to each set of phrases read in automatically.
Apply	Apply the <code>Capitalize Keys</code> and <code>Key Prefix</code> to the current list of phrases.
Auto-Commit	Usually, the program makes the user aware if loaded phrases have not been written to the properties file(s) and the program changed. If this option is checked, then the program performs updates without asking. This is particularly useful in combination with multiple selected source files.
Commit	Save the entries to the properties file(s) and, if not a resource file, convert the original text in the program to use the <code>getTranslation()</code> method with the property key.
Exit	Ends BBJabber.

Optionally, the global string `+BBJABBER_DFLTS` can be used to provide defaults for the bundle name, properties file path, translation object name, program source path, capitalize keys checkbox, key prefix, and auto-apply checkbox.

The global string should be a caret-separated list ('^'), and contain:

- the bundle name (name of your default properties file),
- the path to the properties file,
- the name of the translation object (i.e., what's to the left of `.getTranslation()`),
- the path to source code being translated,
- whether or not property keys should be capitalized (0=no,1=yes),
- the prefix for all keys, if desired,
- whether or not to auto-apply capitalization and/or key prefix (0=no,1=yes).

```
SET +BBJABBER_DFLTS = addon^c:/usr2/props^Translator^c:/usr2/barwork^0^AON_1
```

Inside BBJabber

[\(back to top\)](#)

The extraction/conversion process is on a per file basis, due to the fact that a human should be making the decision as to what should be made available for translation. Double-click on the file to be processed, to load its string literals into the phrases grid, or multi-select source files and use the navigation buttons.

Here's a simple example of what happens in program source code when committing conversions:

1. PRINT "This is a test" is in the codeThe default property key `This_is_a_test` is built.
2. BBJabber sees it and presents it as a candidate to be replaced by the translation call.
3. You choose to convert it (checkmark in the column `Convert`).
4. A subsequent commit replaces the literal
with `Translator!.getTranslation("This_is_a_test")` in the program *in place* and also writes the original phrase under the property key `This_is_a_test` to the default, locale of origin, and selected affected locale resource bundles.

Note: Because program sources will be changed when committed, it is advisable to work with copies of original sources.

If a program has already been processed, then only those strings that have not been already converted are shown in the grid. The only thing you can't do is "unwrap". That's another reason why working with copies is advantageous. Besides, you will not necessarily have to change all programs, so a directory comparison shows you quickly which programs may need the instantiation of the Translator object as shown in the following section. A changed copy of the program can also be easily compared to the original and tested before replacing the original.

Instantiating a Translator Object

[\(back to top\)](#)

BBJabber takes all the literal strings in a program and changes them into a Translator object method that is called with a specific property key. The programmer still has to instantiate the Translator! object manually.

Note: It is the responsibility of the programmer to implement the instantiation of the Translator object itself in the program environment. See [BBTranslator](#) in the BASIS online help.

Example

```
rem --- Get German Translator Object (where "<bbj-install>/utils/translations/"  
is in PREFIX)  
use ::bbtranslator.bbj::BBTranslator  
declare BBTranslator Translator!  
Translator! = BBTranslator.getInstance("addon", "de", "en", "/usr2/props/")
```

Be careful about where in the program you perform this, especially when methods/routines are called directly and the program is not called or run from the top down.

Editing Property Files

[\(back to top\)](#)

The property files used by BBjabber and BBTranslator are plain text files that can be edited as a set from a number of property editors, including the BASIS IDE. Just mount the directory where the *.properties files are located in the IDE's File System panel and double click the default properties entry. It presents a side-by-side display of all the affected languages and handles the conversion of special characters to their (escaped) Unicode equivalents automatically.

Example Program for Translating Resource Files

[\(back to top\)](#)

The following program is an example of using the property files after translation to translate all resource files in a directory from their original language to one or more foreign languages:

```
rem translateArcs.bbj - Output resource files in foreign language(s)

use java.util.Locale

basisHome$=java.lang.System.getProperty("basis.BBjHome")
p=pos("\ "=basisHome$); while p; basisHome$(p,1)="/"; p=pos("\ "=basisHome$); wend
if pos(basisHome$+"/utils/translations/"=pfx)=0 prefix pfx + " " + $22$ +
basisHome$+"/utils/translations/" + $22$

use ::bbtranslator.bbj::BBTranslationBundle
use ::bbtranslator.bbj::BBTranslations

declare BBTranslationBundle transBundle!
declare BBTranslations srcLangTranslations!
declare BBTranslations fgnLangTranslations!
declare BBjString phrase!
declare BBjString encodedKey!

baseArcDir$="./arc/"
resBundleName$="myapp"
resBundleDir$="./prop/"
resBundleSaveDir$="./prop_new/"
srcLocale$="en"
srcLangDir$="enu/"

transBundle! = BBTranslationBundle.getBundle(resBundleName$,resBundleDir$,new
Locale(srcLocale$))
srcLangTranslations! = transBundle!.getTranslations(new Locale(srcLocale$))

need_to_save=0

rem languages_to_process$="*de*es*fr*it*nl*sv*"
languages_to_process$="*de*"

if pos("de"=languages_to_process$) then
    langdir$="deu/"
    fgnLangTranslations! = transBundle!.getTranslations(new Locale("de"))
    gosub build_arcs
```

```

endif

if pos("en"=languages_to_process$) then
    langdir$="enu/"
    fgnLangTranslations! = transBundle!.getTranslations(new Locale("en"))
    gosub build_arcs
endif

if pos("es"=languages_to_process$) then
    langdir$="esp/"
    fgnLangTranslations! = transBundle!.getTranslations(new Locale("es"))
    gosub build_arcs
endif

if pos("fr"=languages_to_process$) then
    langdir$="fra/"
    fgnLangTranslations! = transBundle!.getTranslations(new Locale("fr"))
    gosub build_arcs
endif

if pos("it"=languages_to_process$) then
    langdir$="ita/"
    fgnLangTranslations! = transBundle!.getTranslations(new Locale("it"))
    gosub build_arcs
endif

if pos("nl"=languages_to_process$) then
    langdir$="nld/"
    fgnLangTranslations! = transBundle!.getTranslations(new Locale("nl"))
    gosub build_arcs
endif

if pos("sv"=languages_to_process$) then
    langdir$="sve/"
    fgnLangTranslations! = transBundle!.getTranslations(new Locale("sv"))
    gosub build_arcs
endif

if need_to_save then
    print "Saving resource bundle..."
    beginTime = tim
    transBundle!.save(resBundleSaveDir$)
    endTime = tim
    print "Save took ",(endTime-beginTime)*3600," seconds"
endif
end

build_arcs:
ch=unt
open(ch)baseArcDir$+srcLangDir$+"."

getNextFile:
read record(ch,err=eod)f$
if pos(".arc"=f$)=0 then goto getNextFile
?f$

ch2=unt

```

```

open(ch2)baseArcDir$+srcLangDir$+f$
myfin$=fin(ch2)

erase baseArcDir$+langdir$+f$,err=*next
string baseArcDir$+langdir$+f$
ch3=unt
open(ch3)baseArcDir$+langdir$+f$

buf$=""
read record(ch2,siz=dec(myfin$(1,4)))buf$

rem Line terminator - Windows=$0D0A$/Mac=$0D$/Unix=0A$
line_term$=""
if line_term$="" then
  p=pos($0D0A$=buf$)
  if p=0 then
    p=pos($0D$=buf$)
    if p=0 then
      line_term$=$0A$
    else
      line_term$=$0D$
    fi
  else
    line_term$=$0D0A$
  fi
fi

getNextLine:
if buf$="" then goto eof

p=pos(line_term$=buf$)
if p=0 and buf$<>"" then
  write record(ch3)buf$+line_term$
  goto eof
fi

line$=buf$(1,p-1)
buf$=buf$(p+len(line_term$))

p=pos($22$=line$)
if p=0 then
  write record(ch3)line$+line_term$
  goto getNextLine
fi

newline$=""
while p
  newline$=newline$+line$(1,p)
  line$=line$(p+1)
  p2=pos($22$=line$)
  if p2 then
    phrase!=line$(1,p2-1)
    line$=line$(p2)
    ?"original phrase! [",phrase!,"]"
    encodedKey!=phrase!.replace(" ","_")
    ?"encoded key [",encodedKey!,"]"
    transVal$=fgnLangTranslations!.getTranslation(encodedKey!)

```

```

        ?"translated value [",transVal$,"]"
        if transVal$<>encodedKey!
            newline$=newline$+transVal$
        else
            if pos("_"=phrase!)=0 or pos(" "=phrase!)>0 or
str(phrase!)<>str(encodedKey!) then
                srcLangTranslations!.addTranslation(encodedKey!,phrase!)
                need_to_save=1
            endif
            newline$=newline$+phrase!
        endif
    fi
    p=pos($22$=line$)
wend
if line$<>" " then
    newline$=newline$+line$
    line$=""
fi
write record(ch3)newline$+line_term$
goto getNextLine

eof:
close(ch3)
close(ch2)
goto getNextFile

eod:
close(ch); ch=0
return

```

[\(back to top\)](#)