Can Your App Speak to Your Customer?

"Sprechen ze Deutsch?"

BASIS International Ltd is, as its name suggests, an international company that does business either directly, or through its partners, in some 70 countries around the world. The installation and localization of the base product is delivered in seven languages. About a year ago, BASIS took on the challenge of translating or "localizing" text in the Barista® Application Framework and the Barista-built AddonSoftware® ERP suite. The task was to replace hard-coded strings found in the program code, selection lists, resource files, and messages with an efficient mechanism that would allow for translations to virtually any language.

Because this situation is not uncommon among the thousands of BASIS customers spread across the globe, BASIS decided to provide a generic solution in the form of a new set of translation utility classes based on Java resource bundle concepts and a GUI front end that helps developers identify and convert hard-coded strings in programs and resource files.

By Ralph Lance
Software Engineer

BBTranslator and BBTranslations

BBTranslator and BBTranslations are BBj® CustomObjects whose classes handle just about everything having to do with maintaining and finding translations contained in a *resource bundle*.

A resource bundle is a named set of property files. The property files are plain ASCII text files containing a key=value pair for each translatable phrase. Special characters are UTF-8 (Unicode) encoded and the en/decoding is handled by the translation classes automatically. One such line for a German translation for the key "Actual_Size" might look like the following:

```
Actual_Size=Normale Gr\u00F6\u00DFe
```

In our example, the resource bundle is named "barista." A default property file, **barista.properties**, always contains all translation keys and acts as the "fallback" property file, should a specific translation key not be found in one of the language specific property files. Language-specific property files are identified using the base bundle name plus the locale for a language. For example, generic German would be **barista_de.properties**. Locales can be dialect specific, if need be, by including a country and even a variant. For example, the locale **de_AT** would mean Austrian German. The translation search logic works its way from the most specific translation based on the target locale to the least specific and, as a last resort, to the entry in the default property file. Refer to the link that appears at the end of this article for more about locales.

Integrating the translation functionality into Barista is the same process for any BBj program and means having to:

1. Find hard-coded strings in the program code and ASCII resource files,
2. Build an appropriate translation key,
3. Store the key and its text in property files, and
4. Replace the string with a BBTranslator::getTranslation() method call in the programs, passing it the translation key to be used to find the translation. >>

For performance reasons, BASIS decided to generate language-specific sets of resource files for the Barista framework forms. Enter BBJabber!

BBJabber

BBJabber is a utility that uses BBTranslator classes to assist developers in identifying strings in non-tokenized BBj programs and ASCII resource files that may need to be translated. Suggested translation keys are built automatically and presented before final commitment. Developers can modify translation keys and exclude individual phrases from the translation process as shown in **Figure 1**. The Translator Object Name is the object name to be used in the code and defaults to "Translator", if nothing is entered.

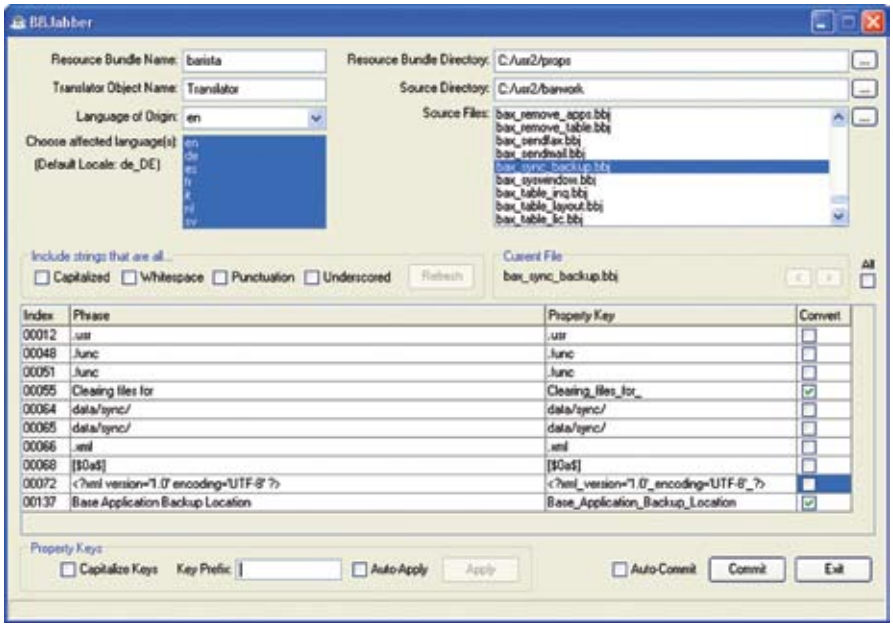

Figure 1. Managing phrases in the translation process

Committing changes for programs replaces the original string in the code with **`Translator!.getTranslation(<translation_key>)`** and stores the translation key and its text in the default property file and the property file of the "language of origin." The Language of Origin is simply the locale of the original text, e.g. **en** in the case of Barista.

BASIS ASCII Resource Files

Unlike BBj programs that are changed in place, committing changes for ASCII resource files will not change the ASCII resource file, but stores the translation keys and their text in the resource bundle. After translation, target language versions of the original ASCII resource files can then be generated. An example program, **`translateArcs.bbj`**, that performs this task appears in the online whitepaper *BBJabber Translation Utility*.

Because every program environment can be different, BBJabber does not include the code for the instantiation of the Translator Object. It is the responsibility of the programmer to provide for this, similar to the following code snippet:

```
rem --- Get Translator Object
use ::bbtranslator.bbj::BBTranslator
declare BBTranslator Translator!
Translator!=BBTranslator.getInstance("barista",stbl("+USER_LOCALE"),null(),dir("")+stbl("+DIR_SYR"))
```

Summary

To complete our localizing story, selection lists, messages, and elements used in building non-framework forms, including those belonging to vertical applications, are also functions that Barista writes to property files using keys built from Barista's own data dictionary information. BASIS Europe personnel has successfully and seamlessly translated the resulting 3,000 phrases in the property file to German, Italian and French. Translation into Spanish, Swedish, and Dutch was completed by a professional translation service. The result … Barista now speaks seven languages, thanks to these three new utilities! ■

For more information about locales, visit Sun Microsystems.

For more detailed information and a sample program that builds ASCII resource files in other languages, follow the online tutorial, *BBJabber Translation Utility*.