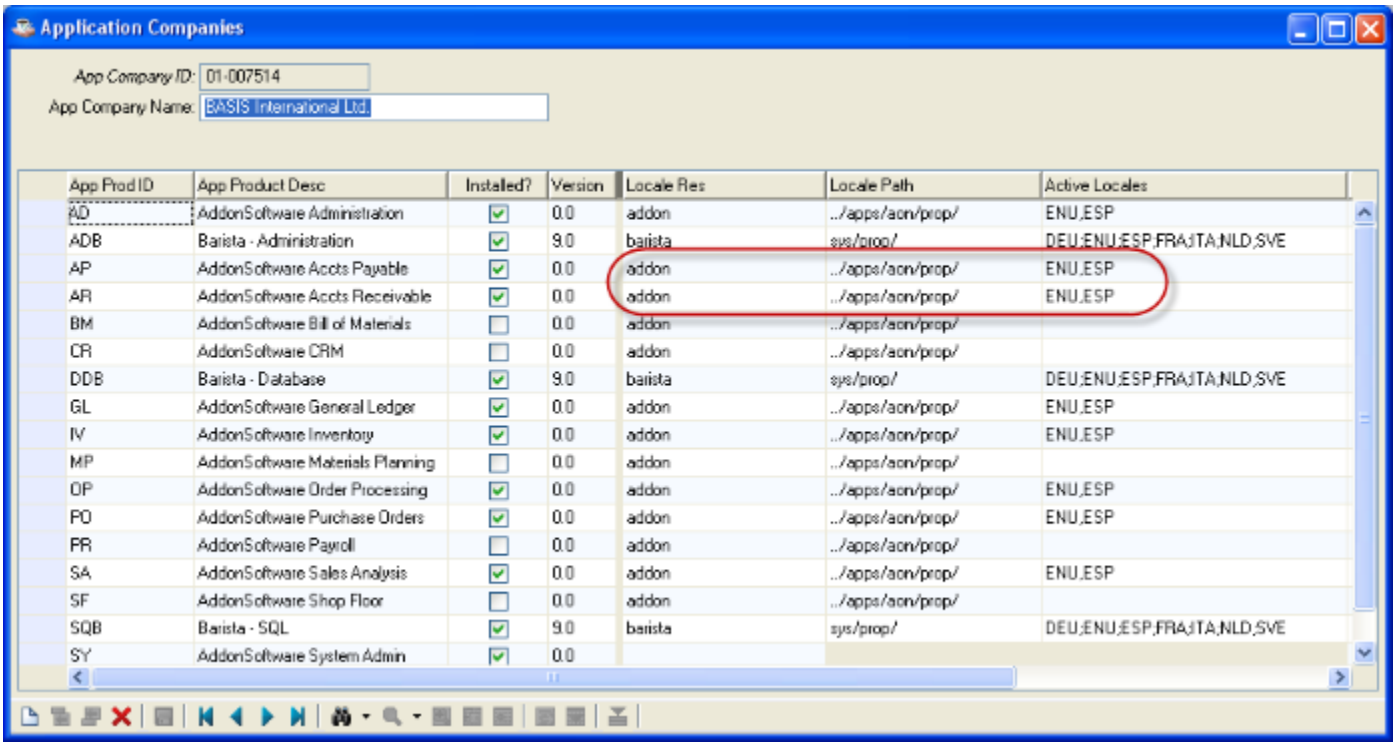
Barista Language Utilities

Overview

The Barista Application Framework® includes utilities to help application developers build Java resource bundles (http://en.wikipedia.org/wiki/Java_resource_bundle). These resource bundles facilitate internationalization by translating such things as labels, prompts, message text, and headings to the correct language for a given locale. The two utilities described use the BBTranslator custom object to create the resource bundle .properties files containing key/value pairs for the base (default) language.

Barista uses the completed .properties files – that is, files in which the additional language translations have been added – to create the .arc and .def files from which forms are run. In addition, Barista performs run-time translation of other "text label" data, such as messages, report headings or other literal text in back-end code.

The prefix, location and active locales for a given set of properties files is defined in the Barista Applications table. The image below indicates that the ../apps/aon/prop/ directory will contain at least three files: addon.properties (base/default), addon_en.properties, and addon_es.properties.



Edit Language Text

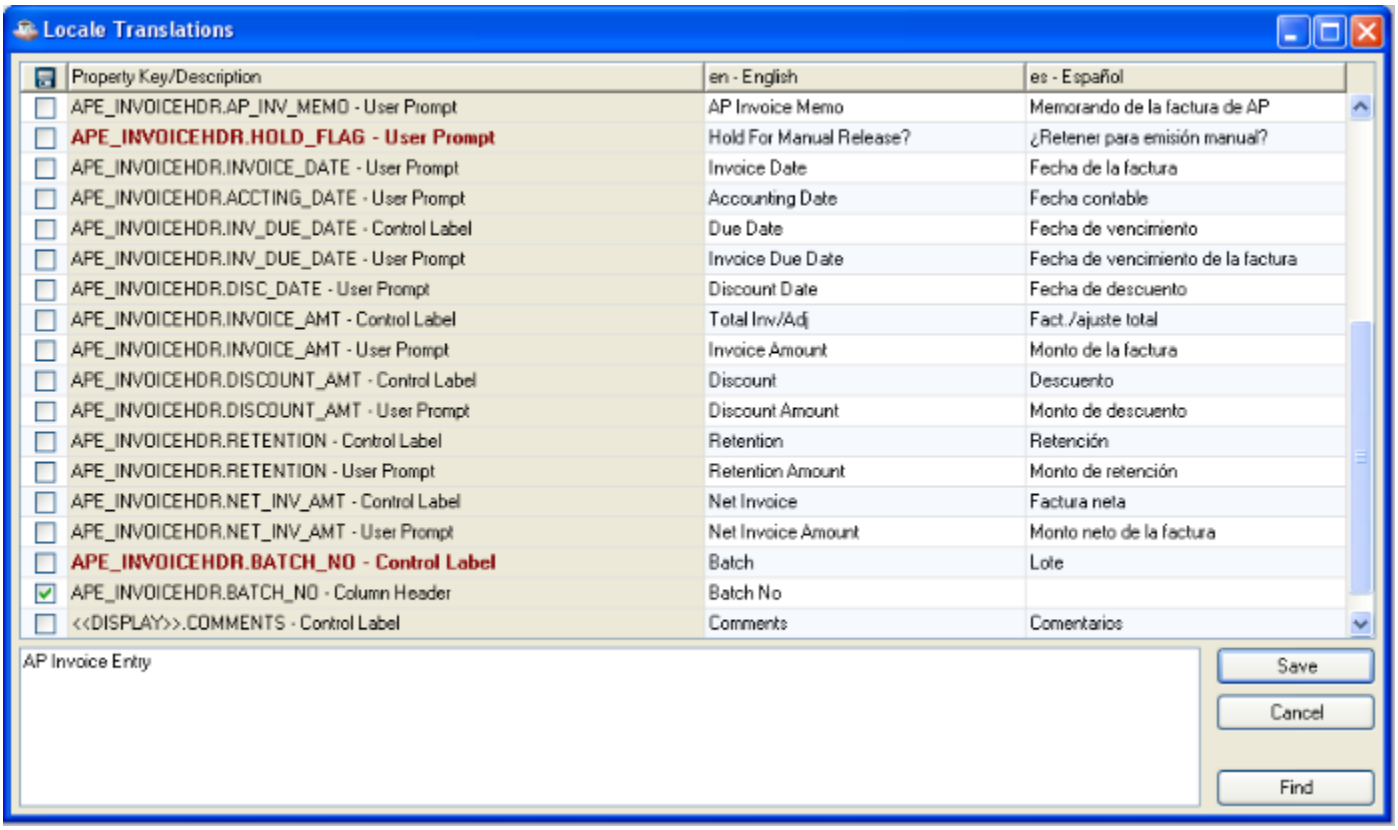
The Edit Language Text utility is accessible from the following forms, either via the Barista Edit menu, or by pressing F6. You should run the utility to check for missing and/or changed translations whenever you create or change any of these translated fields:

Form	Translated Fields
Element Types	Window Label, Column Header, User Prompt
Element List Definitions	Description
System Messages	Message Title, Display Options, Message Text
Form Designer	Window Title, Additional Options, Tab Definitions, Group Headings, User Prompts, Control Labels, Column Headers

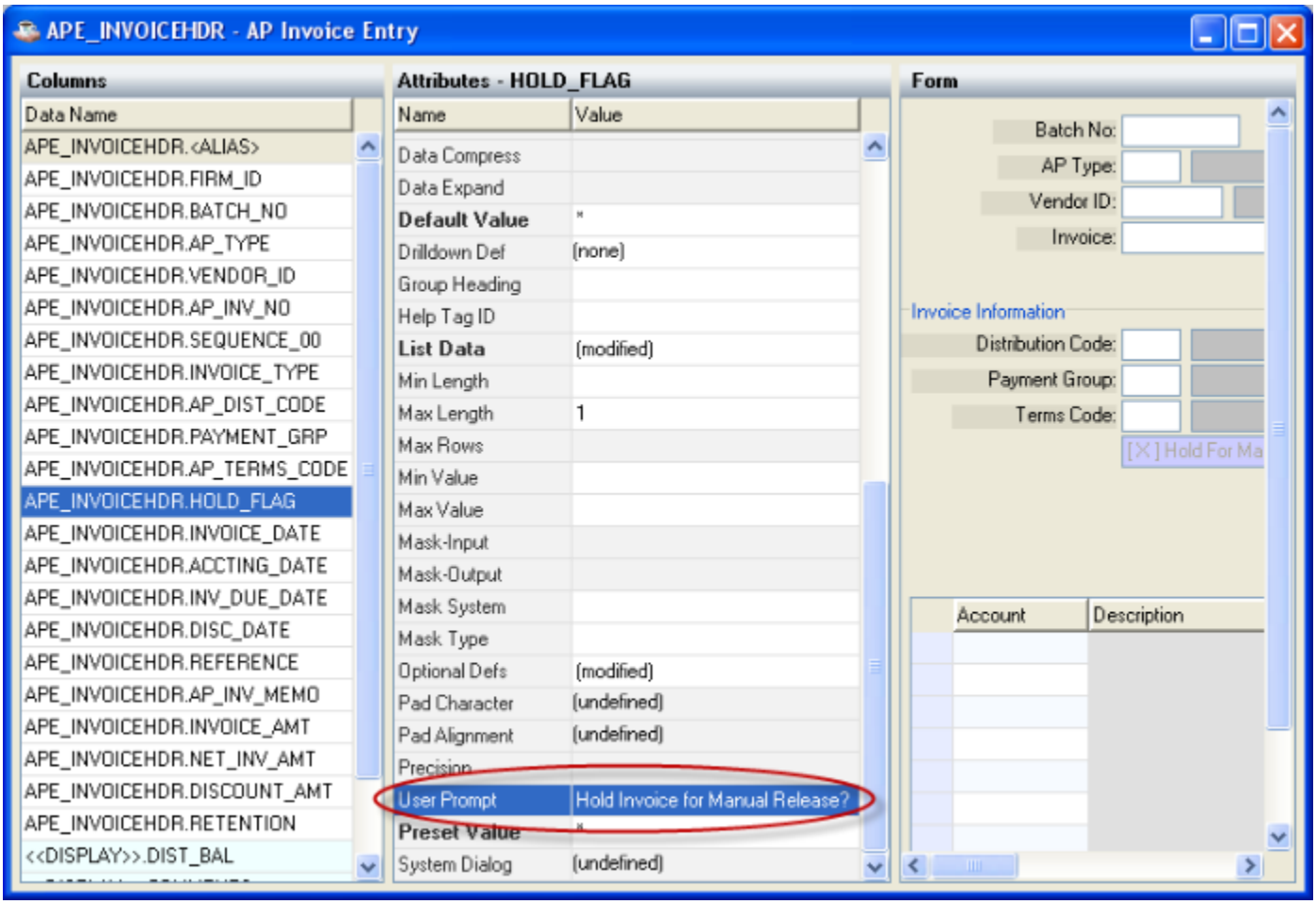
The Edit Language Text form shows all of the keys that are or should be in the .properties file. Keys that are already in the properties file, and for which the base language translation is up to date, display in normal font. Missing keys are also in normal font, but are automatically checked to be added to the .properties file when you click the Save button. Keys for which a conflict exists in the base translation are bolded.

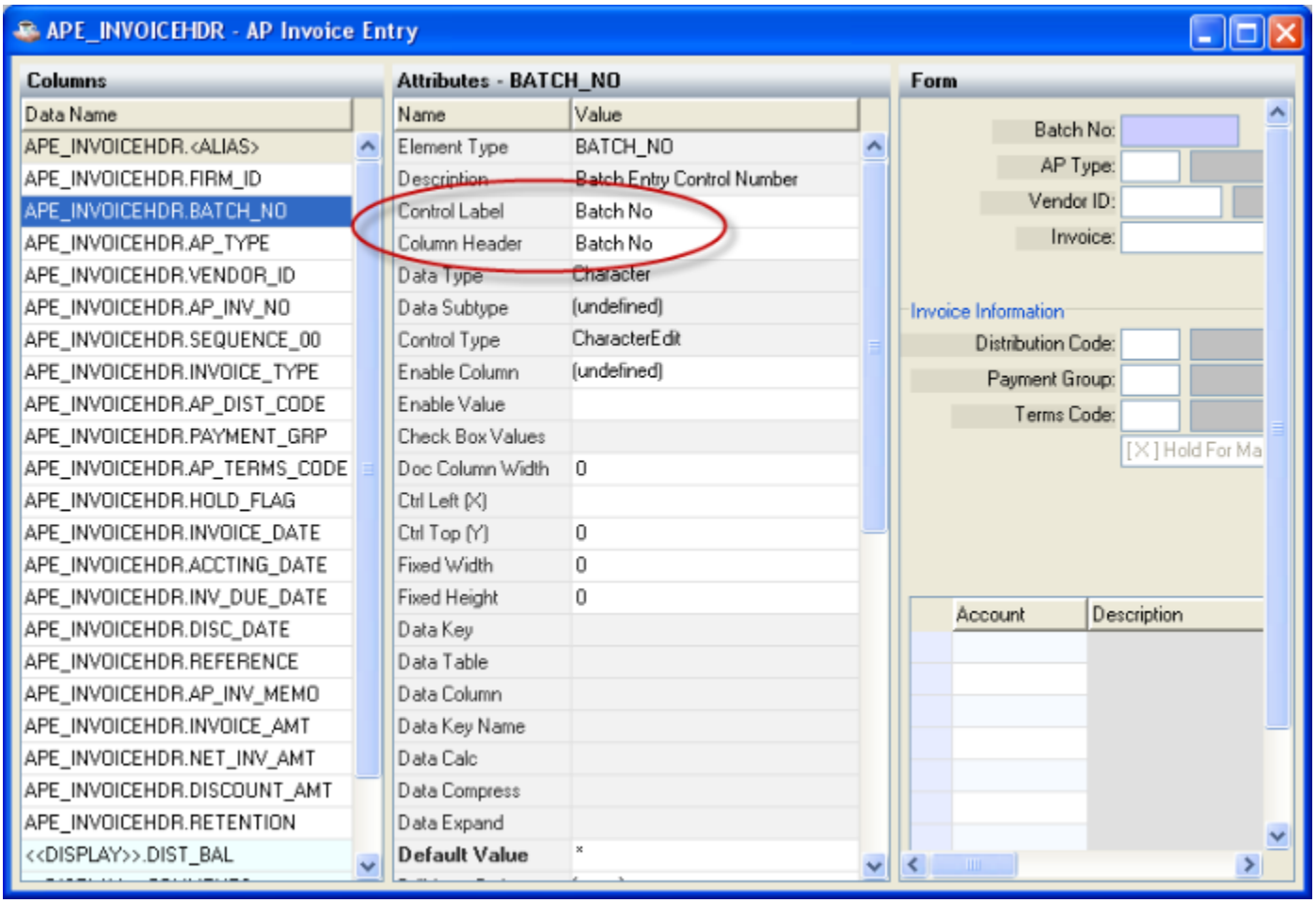
The screenshot below illustrates all of these states. Most of the lines show both the base language and translated text for the given Property Key/Description. Two of the lines are bolded, meaning that the base translation in the .properties file does not match the setting in the Form Designer or on the Element Type. One line has the

Save checkbox selected, indicating that this key and base translation aren't found in the .properties file at all.



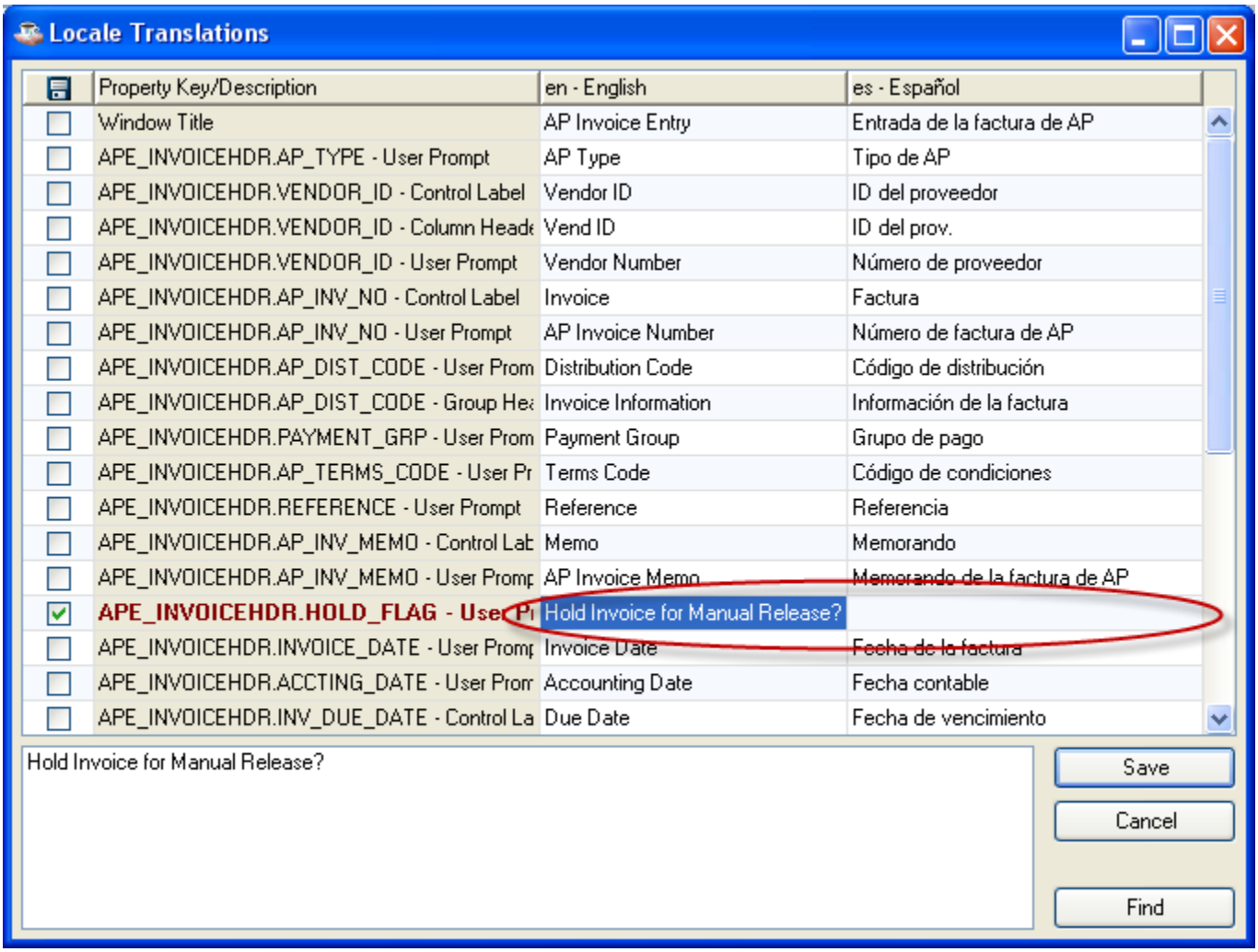
By analyzing the Form Designer for this form, we can see that the User Prompt for the HOLD_FLAG field has changed to "Hold Invoice for Manual Release?" and the Control Label for BATCH_NO now shows as "Batch No." The Column Header for this field was likely removed/blanked out before, and has now been added.


APE_INVOICEHDR - AP Invoice Entry
Columns
Data Name
APE_INVOICEHDR.<ALIAS>
APE_INVOICEHDR.FIRM_ID
APE_INVOICEHDR.BATCH_NO
APE_INVOICEHDR.AP_TYPE
APE_INVOICEHDR.VENDOR_ID
APE_INVOICEHDR.AP_INV_NO
APE_INVOICEHDR.SEQUENCE_00
APE_INVOICEHDR.INVOICE_TYPE
APE_INVOICEHDR.AP_DIST_CODE
APE_INVOICEHDR.PAYMENT_GRP
APE_INVOICEHDR.AP_TERMS_CODE
APE_INVOICEHDR.HOLD_FLAG
APE_INVOICEHDR.INVOICE_DATE
APE_INVOICEHDR.ACCTING_DATE
APE_INVOICEHDR.INV_DUE_DATE
APE_INVOICEHDR.DISC_DATE
APE_INVOICEHDR.REFERENCE
APE_INVOICEHDR.AP_INV_MEMO
APE_INVOICEHDR.INVOICE_AMT
APE_INVOICEHDR.NET_INV_AMT
APE_INVOICEHDR.DISCOUNT_AMT
APE_INVOICEHDR.RETENTION
<<DISPLAY>>.DIST_BAL
Attributes - HOLD_FLAG
Name
Value
Data Compress
Data Expand
Default Value
*
Drilldown Def
(none)
Group Heading
Help Tag ID
List Data
(modified)
Min Length
Max Length
1
Max Rows
Min Value
Max Value
Mask-Input
Mask-Output
Mask System
Mask Type
Optional Defs
(modified)
Pad Character
(undefined)
Pad Alignment
(undefined)
Precision
User Prompt
Hold Invoice for Manual Release?
Preset Value
*
System Dialog
(undefined)
Form
Batch No:
AP Type:
Vendor ID:
Invoice:
Invoice Information
Distribution Code:
Payment Group:
Terms Code:
[X] Hold For Ma
Account
Description



In the case of the changed fields, select the appropriate cell, and type or copy/paste the new base translation either directly in the cell, or into the text area at the bottom of the form. Note: when translating Barista System Messages, the Message Text *must* be edited using the larger text area. Message Text is a multi-line field, and therefore can't be edited directly in the grid because of the potential for embedded tabs and linefeeds.

As soon as any changes are made in these columns, Barista automatically checks the box in the Save column.



You needn't take any action on the non-bolded lines; as long as the Save checkbox is selected, these property keys will be added with a base translation when you click the Save button.

To facilitate translation to other languages (a process which is typically done using text comparison tools), it's best to duplicate new or changed entries in each additional language column *in the original/base language*. This way, the comparison tool can easily locate identical entries as candidates for translation.

BBJabber

In order for literal text in back-end code (reports, publics, updates, etc.) to be translated at run-time, you need to use the getTranslation() method of the BBTranslator object in your code (see http://www.basis.com/onlinedocs/documentation/flash/bbutil/bbtranslator.htm for more information). Barista automatically instantiates a BBTranslator object called Translate! when it begins a new process, so when running

back-end code, the Translate! object is available to you, i.e., you needn't create your own instance of the object.
There are several ways to invoke the getTranslation() method (see http://www.basis.com/onlinedocs/documentation/flash/bbutil/bbtranslator_gettranslation.htm for more detail):

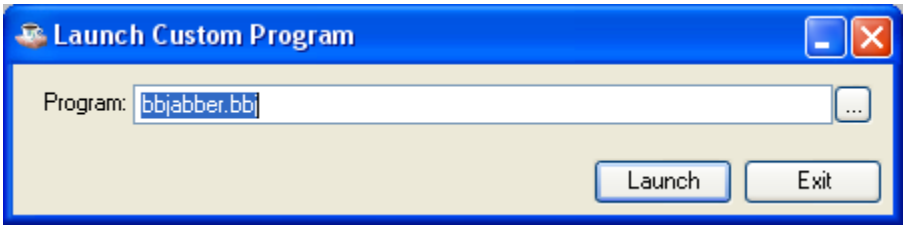
- getTranslation(string key): looks for the specified key, and returns the translation for the current locale. If a translation doesn't exist for the current locale, it returns the base translation. If the key isn't present in the .properties file at all, the key itself is returned.
- getTranslation(string key, string defaultValue): as above, but returns the defaultValue if the requested translation doesn't exist.
- getTranslation(string key, string defaultValue, boolean addIfNotFound): as above, but also adds the key and defaultValue to the .properties file if addIfNotFound = 1.

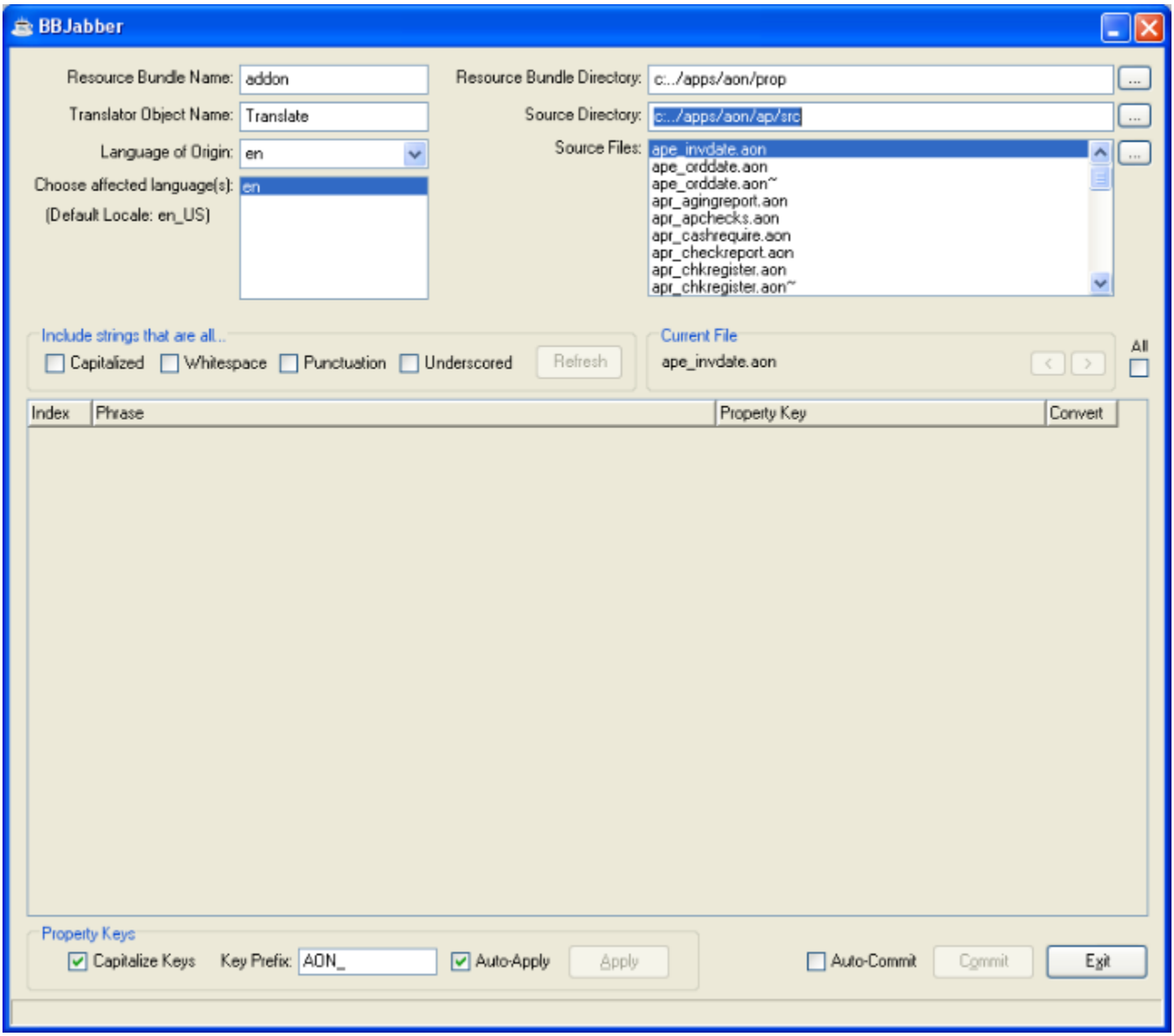
For example, rather than coding a statement like `heading1$="Customer ID"`, you can instruct Barista to translate at run time by using the format `heading1$=Translate!.getTranslation("CUSTOMER_ID")`, where `"CUSTOMER_ID"` is the .properties file key. This is fine if you have an up-to-date .properties file. If you aren't sure whether or not the `"CUSTOMER_ID"` key exists, you can code `heading1$=Translate!.getTranslation("CUSTOMER_ID","Customer ID")` instead. Finally, if you want to not only translate, but also build the .properties file at run time, you can code `heading1$=Translate!.getTranslation("CUSTOMER_ID","Customer ID",1)` so the default value will get added to the .properties file if need-be.

The BBJabber utility provides a batch-mode alternative, so you can process all desired text in one or more back-end programs at once, rather than using the getTranslation() method as you code. Using BBJabber is advantageous in that it not only performs a find/replace of specified text in the back-end program source, but *also adds the keys and base translations to the .properties file.*

If you always use the getTranslation() method when creating new code, your program will be internationalized even if you don't remember to run it through BBJabber. However, using BBJabber is highly recommended, as it provides more control over the quality/consistency of the .properties files, and can find literals that you may have missed while coding.

Launch BBJabber from Barista Utilities ➔ Miscellaneous ➔ Launch Custom Program:





BBJabber requires several pieces of information to get started. You can streamline initialization by including a carat-delimited string to your application .syn file as shown here for AddonSoftware®:

```
STBL=SET +BBJABBER_DFLTS=addon^c:../apps/aon/prop^Translate!^c:../apps/aon/
ap/src^1^AON_^1
```

This STBL sets the following run-time information:

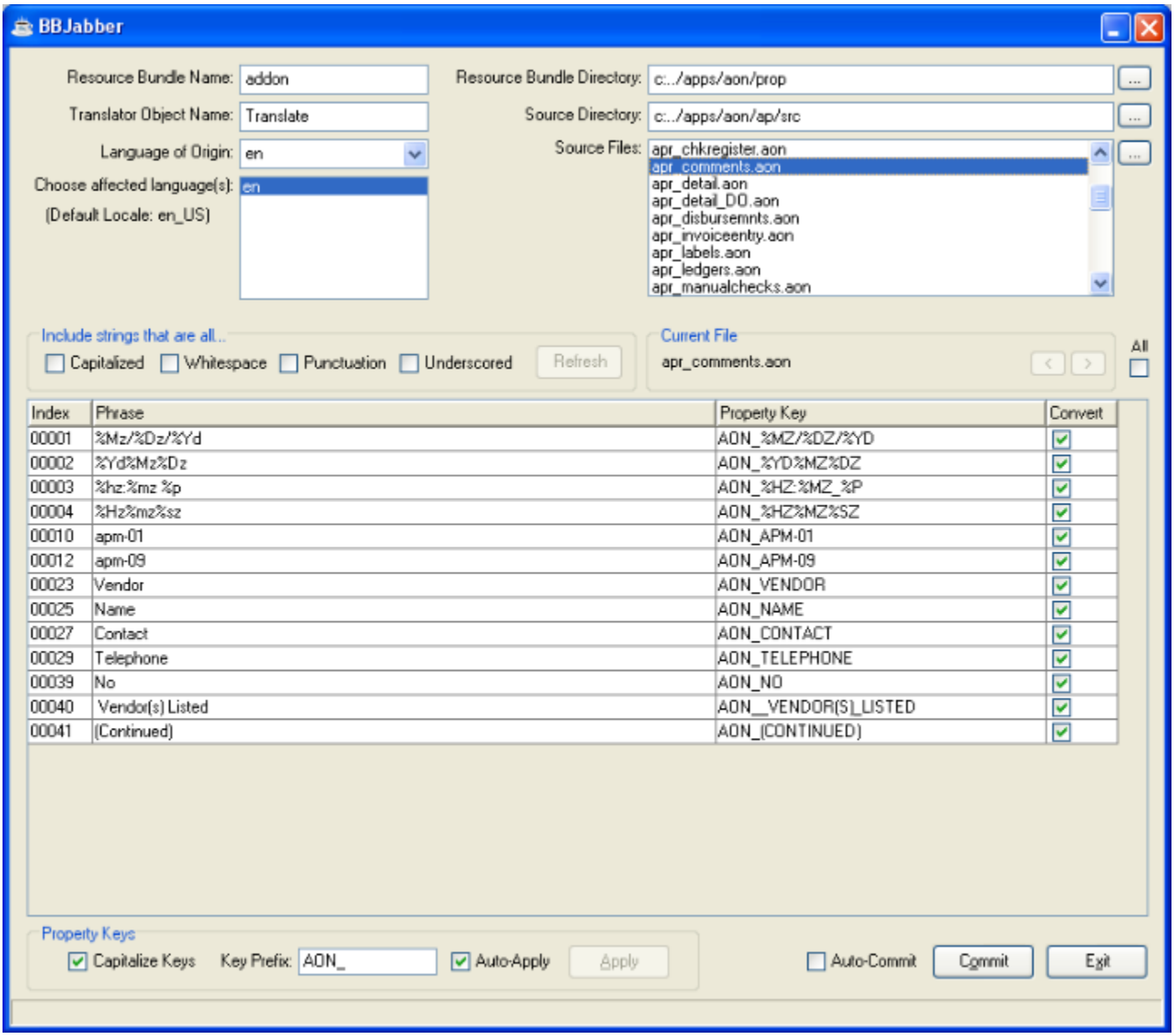
- **Resource Bundle Name** is the name of the various .properties files that comprise the language bundle. In this example, we will have files named addon.properties (the base file containing the default language), addon_en.properties (English), addon_es.properties (Spanish), etc.
- **Resource Bundle Directory** is the path where the Resource Bundle files are stored.
- **Translator Object Name** is the object name to use in the source files. Barista applications use Translate! by default, but if you wish to instantiate your own object, you can supply a different name.
- **Source Directory** is the directory containing the source files you want to process.
- **Source Files** is the list of files found in the Source Directory. This list is populated by BBJabber once the Source Directory is validated.
- **Language of Origin** is set by BBJabber based on the current locale.
- **Choose affected language(s)** is set by BBJabber.
- **Capitalize Keys** instructs BBJabber to convert all keys to upper case (indicated by the first "1" in the STBL above).
- **Key Prefix** allows you to assign a standard prefix on all keys, and is recommended for consistency and to avoid possible duplicates.
- **Auto-Apply** means that BBJabber applies the capitalization and prefixes to all keys automatically, rather than on a case-by-case basis by using the Apply button (indicated by the final "1" in the STBL above).

There are some other options you can set manually, if desired. These options have a more global impact, so should only be used by BBJabber experts:

All, if checked, selects all of the source files and the first file is loaded. In conjunction with the `Auto Commit` option, one can work through a directory of files very quickly. **Auto-Commit** tells BBJabber to save the modified source file back to the Source Directory without prompting.

In addition, while working with the program text, you can instruct BBJabber to include as translation candidates strings that are capitalized, or contain whitespace, punctuation, or underscores. These options are turned off by default.

Below is a BBJabber example using the AddonSoftware application. We have selected the source directory that contains back-end programs for the Accounts Payable module. When we double-click the apr_comments.aon item in the Source Files list, BBJabber reads through that file and presents a list of all string literals that meet our criteria. In this case, we have opted not to show literals that are capitalized, and/ or contain whitespace, punctuation or underscores. You can experiment with these settings to see which ones result in the most accurate list for your purposes.



BBJabber sets the Convert box by default. We can uncheck any strings that should not be translated. You can use either the mouse or spacebar to check/uncheck the box in each row. You may want to have the source file open in an editor to confirm whether or not BBJabber should convert a given string to use the getTranslation() method.

In our sample program, we don't want to alter the date format strings, or the two file names ("apm-01" and "apm-09"), but the other literals should be changed. Once we toggle the Convert checkboxes as needed, we can save our source file by pressing the Commit button. Alternatively, if we simply select another item from the Source Files list, we'll be prompted to save the previous file.

Let's review what BBJabber has done with our apr_comments.aon file. Below are "before" and "after" excerpts of the program code:

Before:

```
rem --- Column Headings

dim columns$[3,10]
  columns = 3
  columns$[0,0]="Vendor",columns$[0,1]="C",columns$[0,2]="10"
  columns$[1,0]="Name",columns$[1,1]="C",columns$[1,2]="50"
  columns$[2,0]="Contact",columns$[2,1]="C",columns$[2,2]="20"
  columns$[3,0]="Telephone",columns$[3,1]="C",columns$[3,2]="10"
```

After:

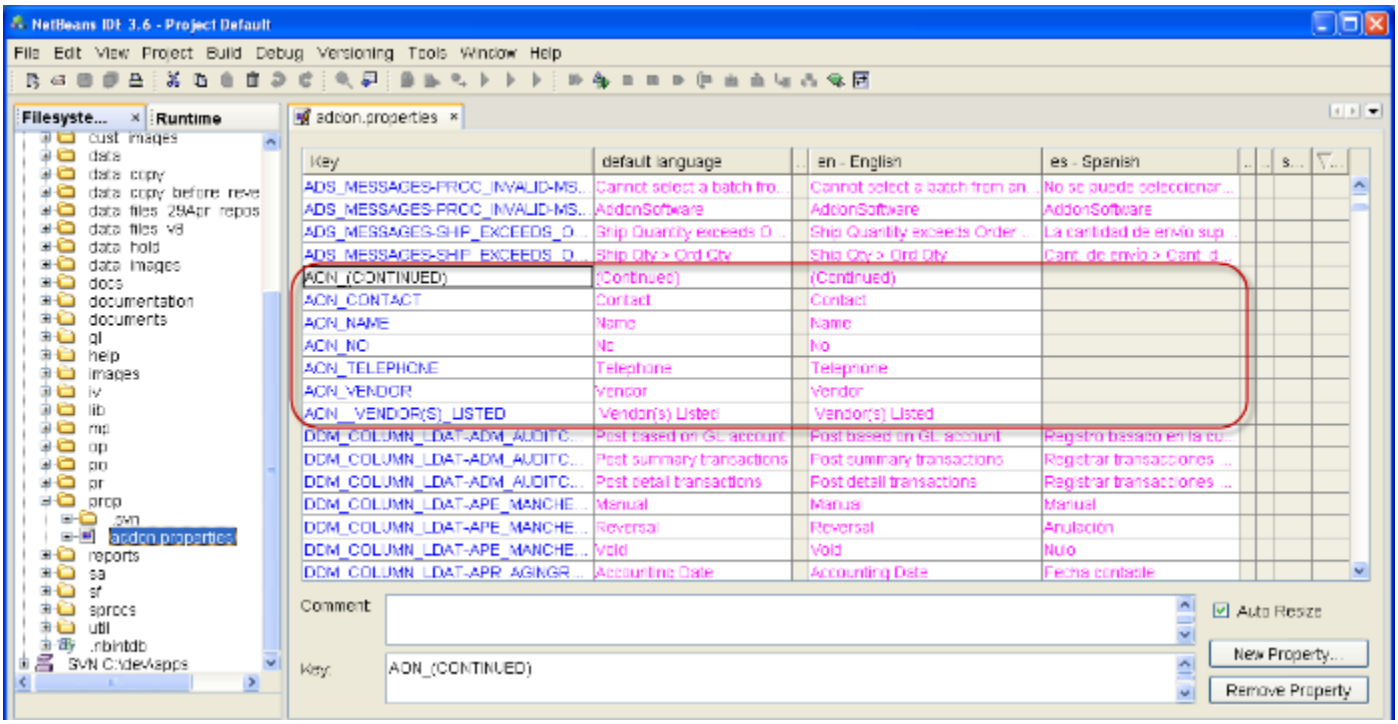
```
rem --- Column Headings

dim columns$[3,10]
  columns = 3
  columns$[0,0]=Translate!.getTranslation("AON_VENDOR"),columns$[0,1]
="C",columns$[0,2]="10"
  columns$[1,0]=Translate!.getTranslation("AON_NAME"),columns$[1,1]
="C",columns$[1,2]="50"
  columns$[2,0]=Translate!.getTranslation("AON_CONTACT"),columns$[2,1]
="C",columns$[2,2]="20"
  columns$[3,0]=Translate!.getTranslation("AON_TELEPHONE"),columns$[3,1]
="C",columns$[3,2]="10"
```

In addition, BBJabber has updated the addon.properties file with the following key/value entries. Note in the last translation that any spaces in keys are replaced with underscore characters, and the backslash ("\") in the translation value indicates that the space should be preserved:

```
AON_(CONTINUED)=(Continued)
AON_CONTACT=Contact
AON_NAME=Name
AON_NO=No
AON_TELEPHONE=Telephone
AON_VENDOR=Vendor
AON__VENDOR(S)_LISTED=\ Vendor(s) Listed
```

Now that we have the keys and base translations we need in the .properties file, we can use a tool such as the BASIS IDE to add additional language translations required for internationalization.


NetBeans IDE 3.6 - Project Default
File Edit View Project Build Debug Versioning Tools Window Help
Filesyste...
Runtime
addon.properties
Key
default language
en - English
es - Spanish
AON_(CONTINUED)
(Continued)
(Continued)
AON_CONTACT
Contact
Contact
AON_NAME
Name
Name
AON_NO
No
No
AON_TELEPHONE
Telephone
Telephone
AON_VENDOR
Vendor
Vendor
AON_VENDOR(S)_LISTED
Vendor(s) Listed
Vendor(s) Listed
Comment
Auto Resize
Key
AON_(CONTINUED)
New Property...
Remove Property