Base Language - Feature #3817

create resource bundles from string literals and implement optional support for setting values from the translation manager database

11/26/2018 12:21 PM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Hynek Cihlar	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**vendor_id:**	GCD

Description

Related issues:

Related to Base Language - Feature #3753: I18N additions	**Closed**
Related to Base Language - Feature #4762: localize system error messages	**New**

History

#2 - 11/26/2018 12:58 PM - Greg Shah

This is meant to be a Java-native replacement for the ability to swap translated string literals at runtime.

Background Information About Translation Manager

Some customers may have used the Translation Manager tool provided by Progress. There are 2 parts to the Translation Manager:

1. UI for defining translations of each translatable string in the application. To be translatable, a string literal must **not** have the "":U (untranslatable) option specified in the source code. A customer has reported that the UI is terrible and they don't use it. Instead they created their own tool to directly edit the Translation Manager database.
2. Compile-time feature which reads the translated strings from a Translation Manager database and creates different language segments in the r-code. At runtime, the CURRENT-LANGUAGE is set via startup parameters or explicitly set in the 4GL code using the CURRENT-LANGUAGE statement. Setting this value will cause the current language "pointer" to select that language's segment in any subsequently loaded r-code. The already loaded r-code is not affected. This means any persistently run r-code will always have the same value as when it was loaded. Based on whichever language segment is selected, any string literals that don't have "":U option specified will have been mapped in r-code to the redirected version in that segment.

Approach in FWD

- Untranslatable string literals will still be emitted as string literals.
- Translatable string literals will be mapped to a resource bundle approach.
  - For customers that have not implemented translation yet, at a minimum FWD can move translatable string literals to a resource bundle approach. This will only be a single resource bundle, but it can then be used as a basis to prepare for I18N.
  - For customers that use Translation Manager, we will implement the equivalent of the compile-time portion (and **not** the UI portion). New conversion inputs will be created from a dump of the translation manager database. The conversion will have to read these contents and use them to generate multiple resource bundles.
- We will need to honor the startup parameter and CURRENT-LANGUAGE such that the resource bundle is selected at runtime when this is set. More discussion of this below.

Areas to Investigate

We need 4GL testcases for the following:

- Write a basic testcase that demonstrates the behavior of "the CURRENT-LANGUAGE that is set when a program is first loaded is always used for that program, even when run subsequent times with a different CURRENT-LANGUAGE setting". We don't need a huge amount of translated text, but should support:
  - English
  - French
  - Dutch
  - German
  - Ukrainian
  - Russian
- Does the CURRENT-LANGUAGE have any effect on the current codepage? Or is there any direct relationship between CURRENT-LANGUAGE and the current codepage?
- What happens when CURRENT-LANGUAGE is set to:
  - unknown value

    - a value for which there is no r-code text segment
    - a value which does not match the default codepage at the OpenEdge level (CPINTERNAL?)
    - a value which does not match the default codepage at the operating system level
- Are all string literals anywhere in a program handled by translation manager or is it only certain strings like those in use as labels (my guess is all but we need to check)?
- How does appserver work with CURRENT-LANGUAGE? Does it do anything? How does the appserver mode work? At a customer there was some discussion that this may work in session mode (e.g. set the CURRENT-LANGUAGE in a connect procedure) and that it may not work in stateless mode.
- In the 4GL, is there a limit in:
    - The number of language segments that can be included in a single r-code file?
    - The size of the individual language segments (or total size of all segments) in a single r-code file?
- How does the 4GL reserve screen space for a given piece of translated text?
    - Presumably, the numeric part of the string options (e.g. "some text":30 or "other text":R25) defines the reserved space. The 4GL docs state that this number is "The number of characters reserved for the string contents in the text segment. The default is the length of the string itself. You might want to specify a longer length if you expect a translation of the string to be longer. The longest length you can specify is 5120 characters.". This really doesn't explain much.
    - Is this honored in the UI?
        - How does it affect layout?
            - Is the layout the same no matter which translation is used? For example, if 40 characters are reserved, then the label/text is always sized for 40 characters.
            - If not specified, does the 4GL calculate the longest of the alternate translations or does it just set the value based on the length of the original string constant?
            - If there is not enough space reserved for a given string, does it truncate the text?
        - Does it honor the justification string options (R right justify, C center, L == left justify and T is trim leading and trailing whitespace)?
        - What happens if there is no justification option? Does the locale define the default of left or right justification?
        - Does this affect labels (side labels, column labels...), frame/window titles and literal text widgets?
        - Are there any other widgets or widget elements affected?
        - Are there differences between ChUI and GUI?
        - How does this interact with options like COLON-ALIGNED?
    - Is this just the amount of space in the language segment for storing alternate translations?
- How does it handle multiple string literals that have the same text?
    - Is each instance treated as separate/independent or can you translate it once for all copies?
    - If there is a single shared translation:
        - What happens if different instances of the same text have different string options?
        - Can this be share across programs (the same text in multiple programs as a shared translation)?
    - Is there any way to translate on a word-by-word basis and have that substituted into multi-word strings? (I'm not sure this would be a great idea, but I'd like to know if it is possible).
- See items listed in #3753-11.
- Is Translation Manager still supported in OpenEdge v12? Someone suggested it may not be supported since there is no documentation as of the time of this writing.

FWD Implementation Questions

- To match the behavior of the 4GL we would have to read the current value at load-time of a class and set the active resource bundle somehow within just the scope of that class. This would never be changed over time, just set at load time. How would this work?
- As a 4GL enhancement, we may want to implement a more dynamic changing of the resource bundle whenever the DYNAMIC-CURRENT-LANGUAGE is set. This would globally change the resource bundle for the entire session.
- The 4GL essentially has resource-bundles that are specific to a single program. This may have an advantage in quick load times. I'm not sure it is strictly needed, but if we provide the dynamic switching capability it may be important to make this very fast. One example use case is to implement this in stateless appserver mode for each call, so that the call could be satisfied in the correct language environment.

#3 - 04/16/2019 06:50 AM - Greg Shah

- Related to Feature #3753: I18N additions added

#4 - 07/15/2019 06:48 AM - Marian Edu

Hi Greg,

there is a small issue we have with this one, we do not have any licenses for the 'translation manager/visual translation' so unless you can somehow provide a 'translation database' that we can use there is little we can do here :(

The least we will need if the translation database schema (.df file) and then we will try to figure out how this works and we can then use it with compile.

#5 - 07/19/2019 10:43 AM - Greg Shah

FYI, the Possenet code that we use does seem to have a (old) version of the Translation Manager, including the .df.

The customer we are working with right now also has posted details on what they are doing in #4044-14.

#6 - 07/15/2020 04:03 PM - Hynek Cihlar

I have good experience with GNU gettext (for Java). The great advantage is the ability to use the original texts in the sources instead of arbitrary IDs. I also had chance to work on localization using the classic Java resource bundles, which was very cumbersome compared to gettext. With the arbitrary IDs, it becomes difficult to resolve the texts, the IDs have to be managed, adding or removing translatable strings requires extra steps, which is a pain when doing many translatable changes.

The translation workflow is also straightforward. gettext has tools for extracting text strings from the sources, resulting in translation po files. These files can than be easily translated by translators, there are numerous great tools for handling po files. The translated po files are then deployed with the app (actually in Java, these must be converted into resource classes). The translation can be done incrementally, if a message translation is missing the default language (as used in the source code) is used.

Besides, gettext provides additional advanced features like plural handling.

#7 - 07/16/2020 02:35 PM - Greg Shah

This is very interesting. I wasn't familiar with gettext. Thank you for posting this!

I like that the results are still resource bundles (either .properties form or the .class form). We would certainly want to use the .class form and load them from the application jar.

Interestingly, you could still use the traditional java.util.ResourceBundle API with this so that makes this even more attractive, even though we probably would want to use the gnu.gettext.GettextResource API for the reasons you've already stated. If we used the ResourceBundle API, we might have had to deal with the MissingResourceException anyway. I guess we would have made our own wrappers for it. Using GettextResource API this is not needed.

Overall it seems like a very good idea. I don't see much of a down side here. The only thing is if a customer really wanted the classic Java ResourceBundle API + IDs approach, then we would need to emit things differently in the source code.

Hynek: If you know of any other "negatives" or considerations, post them here.

Some good references:

https://en.wikipedia.org/wiki/Gettext
https://www.gnu.org/software/gettext/gettext.html

https://www.gnu.org/software/gettext/manual/html_node/Java.html

#8 - 08/19/2021 07:43 PM - Greg Shah

- Assignee set to Hynek Cihlar

#9 - 08/20/2021 02:30 AM - Hynek Cihlar

I assume the scope includes translation of FWD runtime, too. Like error messages. Correct?

#10 - 08/20/2021 03:02 AM - Hynek Cihlar

Greg Shah wrote:

> FWD Implementation Questions
>
> - To match the behavior of the 4GL

Greg, what behavior are you referring to?

> we would have to read the current value at load-time of a class and set the active resource bundle somehow within just the scope of that class. This would never be changed over time, just set at load time. How would this work?

#11 - 08/20/2021 07:50 AM - Greg Shah

- Related to Feature #4762: localize system error messages added

#12 - 08/20/2021 07:51 AM - Greg Shah

> I assume the scope includes translation of FWD runtime, too. Like error messages. Correct?

No, we will handle that in #4762. But in #4762, we will use the same technique you use here. So please plan for this, but you don't have to implement it yet.

#13 - 08/20/2021 08:08 AM - Greg Shah

Hynek Cihlar wrote:

> Greg Shah wrote:

> > FWD Implementation Questions
> >
> > - To match the behavior of the 4GL
>
> Greg, what behavior are you referring to?

This from #3817-1:

> At runtime, the CURRENT-LANGUAGE is set via startup parameters or explicitly set in the 4GL code using the CURRENT-LANGUAGE statement. Setting this value will cause the current language "pointer" to select that language's segment in any subsequently loaded r-code. The already loaded r-code is not affected. This means any persistently run r-code will always have the same value as when it was loaded. Based on whichever language segment is selected, any string literals that don't have "":U option specified will have been mapped in r-code to the redirected version in that segment.

> > we would have to read the current value at load-time of a class and set the active resource bundle somehow within just the scope of that class. This would never be changed over time, just set at load time. How would this work?

Actually, we need to confirm that the behavior is as I understand it. In other words, we must confirm that for a given r-code file (e.g. some-procedure.r), that all instances of it on the stack will use the language segment corresponding to the CURRENT-LANGUAGE value at the time the .r file was **first loaded**. If the 4GL code runs the some-procedure.r multiple times and changes CURRENT-LANGUAGE in between those runs, the new value of CURRENT-LANGUAGE is not honored in the subsequent runs.

If all instances share the same **first load** CURRENT-LANGUAGE value, what happens if all instances of some-procedure.r are unloaded from memory and then the CURRENT-LANGUAGE is changed?

Or does each instance of some-procedure.r get loaded separately and use the language segment of the CURRENT-LANGUAGE at the time they were loaded?

Based on the findings, we must match that same behavior.

#14 - 08/20/2021 09:28 AM - Hynek Cihlar

Depending on the CURRENT-LANGUAGE scope, the resource bundle will have to be scoped accordingly. The possible outcomes are class, class instance or application. For class or class instance we will have to break up the resource bundles so that each class receives its own. The bundle could be initialized in a static or instance initializer and then accessed by the translation API.

#15 - 08/20/2021 09:34 AM - Hynek Cihlar

Greg Shah wrote:

> - The 4GL essentially has resource-bundles that are specific to a single program. This may have an advantage in quick load times. I'm not sure it is strictly needed, but if we provide the dynamic switching capability it may be important to make this very fast. One example use case is to implement this in stateless appserver mode for each call, so that the call could be satisfied in the correct language environment.

The dynamic switching logic could work on a lazy-initialization basis. DYNAMIC-CURRENT-LANGUAGE would just simply set a flag that a dynamic language is in effect with the actual language. The bundle dereferencing logic would check the flag and if set, it would resolve the correct bundle and cache it for subsequent calls.

#16 - 08/20/2021 10:52 AM - Hynek Cihlar

In order to estimate the efforts I came up with the following sub tasks:

1. Resolve the unknowns laid out by Greg in the previous points. (2 MD)
2. Implement conversion rules to produce po files for all the identified translatable strings. (1 MD)
3. Implement conversion rules to emit required Java code - any bundle initialization logic, gettext calls for all translatable strings. (1 MD)
4. Implement bundle management logic - this includes bundle scoping and initialization, and bundle resolution. (3 MD)
5. Imlement wrapper API for GNU gettext. The logic will automate the bundle resolution besides the obvious text translation. (1 MD)
6. Implement DYNAMIC-CURRENT-LANGUAGE language extension. (2 MD)
7. Document the translation workflow. This is the user documentation what to do to translate an app. (1 MD)

#17 - 08/20/2021 10:59 AM - Greg Shah

The plan is good.

> 3. Implement conversion rules to emit required Java code - any bundle initialization logic, gettext calls for all translatable strings. (1 MD)

As much as possible, I'd like to keep the initialization logic in the runtime. The BlockManager could be enhanced to provide callbacks except we have no good hook for the instantiation of the class itself. So I can see the need for a static or instance initializer there. If we use class-level bundles, then some kind of conversion time information will be needed, though this could be provided via class-level annotations. Or we could use a naming standard where the class-level bundle can be calculated based on the business logic class name. I think the annotations approach would be most flexible.

#18 - 08/20/2021 11:06 AM - Hynek Cihlar

Greg Shah wrote:

> The plan is good.
>
> > 3. Implement conversion rules to emit required Java code - any bundle initialization logic, gettext calls for all translatable strings. (1 MD)
>
> As much as possible, I'd like to keep the initialization logic in the runtime. The BlockManager could be enhanced to provide callbacks except we have no good hook for the instantiation of the class itself. So I can see the need for a static or instance initializer there. If we use class-level bundles, then some kind of conversion time information will be needed, though this could be provided via class-level annotations. Or we could use a naming standard where the class-level bundle can be calculated based on the business logic class name. I think the annotations approach would be most flexible.

I like the BlockManager callback idea. Introducing generic callbacks and calling them from class static and instance initializers.

#19 - 09/02/2021 07:01 AM - Hynek Cihlar

- Status changed from New to WIP

#20 - 09/02/2021 07:04 AM - Hynek Cihlar

Greg, should I create new task branch for the changes?

#21 - 09/02/2021 07:21 AM - Greg Shah

Yes, but you should base it on 3821c.

#22 - 09/02/2021 08:48 AM - Greg Shah

- File progress_software_knowledge_base_article_frequently_asked_questions_regarding_the_translation_manager_open_source_initiative_20201120.pdf added

- File progress_software_knowledge_base_article_translation_manager_and_visual_translator_are_now_open_source_20201120.pdf added

Hynek alerted me to the following interesting articles:

https://knowledgebase.progress.com/articles/Article/Translation-Manager-and-Visual-Translator-are-now-Open-Source
https://knowledgebase.progress.com/articles/Knowledge/Frequently-Asked-Questions-FAQ-regarding-the-Translation-Manager-OpenSource-initiative

I've archived both articles as PDFs attached to this task, in case the knowledge base URLs get moved or the contents deleted.

The key point is that the Translation Manager and the Visual Translator are both open source now under the Apache 2.0 license. We are allowed to convert and use these projects without any restrictions. PSC no longer supports these tools in any way. This looks like it was done so that they could retire the code (end of life it).

It turns out that the github URLs referenced are incorrect. The correct link is https://github.com/gquerret/adetran (at least at this time).

Hynek: Please archive a version of this open source project so that we are assured that we always have access.

#23 - 09/02/2021 09:10 AM - Hynek Cihlar

- File gquerret_adetran_ Progress OpenEdge Visual Translator and Translation Manager archive.pdf added

- File adetran-master.zip added

The attached is the zipped archive of the https://github.com/gquerret/adetran project sources. I'm also attaching the landing page in pdf.

#24 - 09/14/2021 08:31 AM - Hynek Cihlar

Here are a few notes about building the Adetran project. These are on top of the existing documentation.

Building Translation Manager (Windows)

- The used build system is Gradle in version 6.5. Do not attempt to use more recent version as the used pluggins are not compatible witg 7.x. Use JDK 14, anything more recent doesn't seem to work with Gradle 6.x.
- It is OK to use OpenJDK (https://openjdk.java.net/projects/jdk/14/). Download the zip archive, unzip in c:\java\jdk14 and set PATH environment variable to c:\java\jdk14\bin.
- When building the project, make sure you do so from the shell with all the OE environment variables set. Use proenv from the Windows start menu.
- To build, run gradlew.bat build.

Running Translation Manager

Adetran project root contains two wrapper procedure files to run Visual Translator and Translation Manager, launch_tranman.p and launch_visual_translator.p. Before running these with prowin -p launch_tranman.p, open them and comment out the line with session:exit-code = 0, if this assignment gives a runtime error.

#25 - 09/16/2021 01:07 PM - Hynek Cihlar

See my findings below. I will update this note as I go.

Greg Shah wrote:

> Areas to Investigate
>
> - Write a basic testcase that demonstrates the behavior of "the CURRENT-LANGUAGE that is set when a program is first loaded is always used for that program, even when run subsequent times with a different CURRENT-LANGUAGE setting". We don't need a huge amount of translated text, but should support:

The effective language of an external procedure is determined based on the actual CURRENT-LANGUAGE when the number of the procedure instances is 0 and the procedure is instantiated. I.e. there can't be two instances of the same external procedure with different languages. Also the language of an external procedure can be changed when all its instances are deleted, CURRENT-LANGUAGE is changed and the procedure is instantiated again. As a corollary, an external procedure ran non-persistently, with no outstanding persistent instances, can be invoked with different language for every invocation.

TODO Appserver.

> - Does the CURRENT-LANGUAGE have any effect on the current codepage? Or is there any direct relationship between CURRENT-LANGUAGE and the current codepage?

CURRENT-LANGUAGE can be set to any text value. It can be set to undefined or a non-existent language name. The language name is a free text set in Translation Manager and has no connection to any ISO standard or any code page value. If a language name is used that is not compiled in r-code (including undefined) the untranslated texts are used.

- What happens when CURRENT-LANGUAGE is set to:
  - unknown value
  - a value for which there is no r-code text segment
  - a value which does not match the default codepage at the OpenEdge level (CPINTERNAL?)
  - a value which does not match the default codepage at the operating system level

See above.

- Are all string literals anywhere in a program handled by translation manager or is it only certain strings like those in use as labels (my guess is all but we need to check)?

Most literals. For example procedure names in RUN are not translated. TODO: Check other statements, file paths, etc.

- How does appserver work with CURRENT-LANGUAGE? Does it do anything? How does the appserver mode work? At a customer there was some discussion that this may work in session mode (e.g. set the CURRENT-LANGUAGE in a connect procedure) and that it may not work in stateless mode.

TODO

- In the 4GL, is there a limit in:
  - The number of language segments that can be included in a single r-code file?
  - The size of the individual language segments (or total size of all segments) in a single r-code file?

TODO

- How does the 4GL reserve screen space for a given piece of translated text?
  - Presumably, the numeric part of the string options (e.g. "some text":30 or "other text":R25) defines the reserved space. The 4GL docs state that this number is "The number of characters reserved for the string contents in the text segment. The default is the length of the string itself. You might want to specify a longer length if you expect a translation of the string to be longer. The longest length you can specify is 5120 characters.". This really doesn't explain much.
  - Is this honored in the UI?
    - How does it affect layout?
      - Is the layout the same no matter which translation is used? For example, if 40 characters are reserved, then the label/text is always sized for 40 characters.
      - If not specified, does the 4GL calculate the longest of the alternate translations or does it just set the value based on the length of the original string constant?
      - If there is not enough space reserved for a given string, does it truncate the text?
    - Does it honor the justification string options (R right justify, C center, L == left justify and T is trim leading and trailing whitespace)?
    - What happens if there is no justification option? Does the locale define the default of left or right justification?
    - Does this affect labels (side labels, column labels...), frame/window titles and literal text widgets?
    - Are there any other widgets or widget elements affected?
    - Are there differences between ChUI and GUI?
    - How does this interact with options like COLON-ALIGNED?
  - Is this just the amount of space in the language segment for storing alternate translations?

I just scratched the logic of text expansion. So far I know that:

- Only some widgets and their states are expanded. For example TEXT widget doesn't seem to be expanded. Widget labels are expanded but only in the form of side-labels.
- The expansion is global, same for all languages.
- Adding the string max length option ("string":10) will disable expansion for this particular string.
- The amount of the expansion is calculated based on a an internal expansion table, which defines how meny blank chars is added to the string depending on its original length. The expansion can be modified with a globally defined percentual value such that the resulting string length equals to [original length] + [defined percentual value] * [entry in the internal table]. The percentual value is defined during compilation of the

procedure files in Translation Manager. Perhaps we could make this a runtime option in FWD.

- Justification is TODO.
- ChUI is TODO.

> - How does it handle multiple string literals that have the same text?
>   - Is each instance treated as separate/independent or can you translate it once for all copies?

Yes, they are independent. Translation Manager however can help with these translations through its Glossary feature.

> - Is there any way to translate on a word-by-word basis and have that substituted into multi-word strings? (I'm not sure this would be a great idea, but I'd like to know if it is possible).

No, it treats every string as a translation entry. Translation Manager does provide a Glossary feature, but I believe this only handles same entries.

#26 - 09/16/2021 01:12 PM - Hynek Cihlar

To support the conversion path for customers using OE translation workflow, the translations can be "dumped" from Translation Manager to CSV files and converted to PO files, which are directly used by GNU gettext for writing translations and resource bundle generation.

#27 - 09/16/2021 01:42 PM - Greg Shah

> As a corollary, an external procedure ran non-persistently, with no outstanding persistent instances, can be invoked with different language for every invocation.

Even if there are no instances run persistently, what happens if there is a non-persistent instance on the call stack when you change CURRENT-LANGUAGE and then run another non-persistent instance? Will the new CURRENT-LANGUAGE be honored in the new instance while the old instance is still on the stack and is still using the old CURRENT-LANGUAGE?

#28 - 09/16/2021 01:56 PM - Greg Shah

Please note that we will need to update the font metrics calculation to process all the translations. And we will need some mechanism to store the largest such size for each side-label.

#29 - 09/17/2021 05:10 AM - Hynek Cihlar

Greg Shah wrote:

> > As a corollary, an external procedure ran non-persistently, with no outstanding persistent instances, can be invoked with different language for every invocation.
>
> Even if there are no instances run persistently, what happens if there is a non-persistent instance on the call stack when you change CURRENT-LANGUAGE and then run another non-persistent instance? Will the new CURRENT-LANGUAGE be honored in the new instance while the old instance is still on the stack and is still using the old CURRENT-LANGUAGE?

In this case the new value of CURRENT-LANGUAGE is not honored. It is honored only when the procedure is removed from call stack and called again.

#30 - 09/17/2021 05:10 AM - Hynek Cihlar

Greg Shah wrote:

> Please note that we will need to update the font metrics calculation to process all the translations. And we will need some mechanism to store the largest such size for each side-label.

Right, good point.

#31 - 09/27/2021 02:30 PM - Hynek Cihlar

- File 2021-09-27_20-28.png added

The attached file shows Translation Manager Compile dialog. This is used to compile 4GL sources with the specified languages. Note the Growth Table field, which is used to define the amount of string expansion.

#32 - 09/29/2021 08:59 AM - Hynek Cihlar

Hynek,

After conversion, is the translation table mapping to 4GL source code line/column numbers? Why is this needed?

I ask because if someone moves away from programming in 4GL and writes direct Java code, the legacy source line/column would have no more meaning.

Thanks,

Constantin

#33 - 09/29/2021 09:02 AM - Hynek Cihlar

Hynek Cihlar wrote:

> Hynek,
>
> After conversion, is the translation table mapping to 4GL source code line/column numbers? Why is this needed?
>
> I ask because if someone moves away from programming in 4GL and writes direct Java code, the legacy source line/column would have no more meaning.
>
> Thanks,
>
> Constantin

The source reference is needed for the translators to have access to the original location. This should be useful during the conversion phase when the PO files are generated from the 4GL sources. Once/if the customer moves away from the 4GL sources the work flow will be a bit different, the PO files will be generated from the Java sources.

#34 - 09/29/2021 09:10 AM - Greg Shah

> Once/if the customer moves away from the 4GL sources the work flow will be a bit different, the PO files will be generated from the Java sources.

Does gettext have features to implement marking of string literals as untranslatable and so forth like the 4GL has "string options"?

We will have to document how this can work for post-conversion development. This must include a plan to use the gettext PO file generation only on the files which are no longer being converted or which started as Java (never converted). If you have ideas about how this can work, please post them here.

#35 - 09/29/2021 09:17 AM - Hynek Cihlar

Greg Shah wrote:

> > Once/if the customer moves away from the 4GL sources the work flow will be a bit different, the PO files will be generated from the Java sources.
>
> Does gettext have features to implement marking of string literals as untranslatable and so forth like the 4GL has "string options"?
>
> We will have to document how this can work for post-conversion development. This must include a plan to use the gettext PO file generation only on the files which are no longer being converted or which started as Java (never converted). If you have ideas about how this can work, please post them here.

The way this normally works in C, Java and similar languages is to pass the translatable literals in the function or method responsible for performing the translation. The method typically has a unique name as it is then regexed later to generate the PO files. GNU Gettext provides utils for grabbing these literals from a source file and producing the corresponding PO file.

As an example the following class field

String aVar = "This is a translatable message";

will be wrapped as follows:

String aVar = __tr("This is a translatable message");

Any untranslatable strings are simply not wrapped by the translation method.

#36 - 09/29/2021 09:24 AM - Hynek Cihlar

Hynek Cihlar wrote:

> Any untranslatable strings are simply not wrapped by the translation method.

GNU gettext is pretty flexible. The outcome of this issue should also be a set of guidelines to help with the translation tasks during conversion and post-conversion.

#37 - 09/30/2021 08:04 AM - Greg Shah

> The attached file shows Translation Manager Compile dialog. This is used to compile 4GL sources with the specified languages. Note the Growth Table field, which is used to define the amount of string expansion.

I just checked with a customer that uses translation manager with a desktop GUI application. They use Growth Table and set the value to 40%.

#38 - 10/01/2021 08:11 AM - Hynek Cihlar

- % Done changed from 0 to 80

3821c revision 13017 adds core runtime and conversion support.

The conversion support when explicitly enabled generates set of PO files with all the translatable literals gathered during conversion. The translatable strings are also wrapped in translation method calls.

The runtime support contains implementation of CURRENT-LANGUAGE session attribute, language switching, language resource bundle resolution and the translation of messages.

Missing pieces are:

- string expansion
- conversion of exported Transaction Manager database in PO files and inclusion in the conversion workflow
- documentation and guidelines

As mentioned above, i18n conversion support must be explicitly enabled, by default the support is off and the conversion should produce the same java sources as before this check in. To enable add the global parameter i18n-enable in p2j.cfg.xml and set it to true.

Please review, especially the conversion part.

#39 - 10/01/2021 08:18 AM - Greg Shah

Both Constantin and I will review this.

What is the process to export the translation manager database and provide it to the conversion? Is it provided as .d files or in some kind of .csv export? Where are they to be placed for conversion?

#40 - 10/01/2021 08:30 AM - Hynek Cihlar

Greg Shah wrote:

> Both Constantin and I will review this.
>
> What is the process to export the translation manager database and provide it to the conversion? Is it provided as .d files or in some kind of .csv export? Where are they to be placed for conversion?

This part is not finished yet. The expected worflow is to export the translations from Thransaction Manager to a csv file and generate PO files from it.

#41 - 10/01/2021 09:27 AM - Hynek Cihlar

Hynek Cihlar wrote:

> This part is not finished yet. The expected worflow is to export the translations from Thransaction Manager to a csv file and generate PO files from it.

I plan to check this in later in the day.

#42 - 10/01/2021 10:54 AM - Greg Shah

Code Review Task Branch 3821c Revision 13017

This is very good. I think the result is quite elegant and is layered in on our existing approach very well.

1. Does the 4GL Translation Manager operate on the fully preprocessed files? My guess is that it is aware of includes and allows references to strings in those. Our approach is based on the cache files, so it seems to me that there may be a "disconnect" between our approach and the original. For strings that are sourced from include files, this could mean that the same string appears hundreds or thousands of times in the application. I think this means we would duplicate the string in the PO files that same number of times. If I understand correctly, I think we cannot do that because it means that the people doing the translation must translate it many times in FWD but just once in the 4GL. That is not workable.

2. Do I understand correctly that annotations/i18n.rules will get changes to integrate the existing translations that were exported from the 4GL Translation Manager?

3. In annotations/i18n.rules, why not change sprintf("%s/%s.%s", fdir, this.getAnnotation("classname"), "po") to be sprintf("%s/%s.po", fdir, this.getAnnotation("classname"))?

4. The current process overwrites the .po file every time. This will need to change when translators start implementing changes post-conversion. In other words, post-conversion translators will be editing the .po files right? We will continue to run conversion on (at least some) files indefinitely. We can't blow away all those changes each time.

5. In frame_generator.xml the do_format changes add ref as a parameter to set_attr_ex_ex. It is not clear if that should be ref2 when ref2 is not null. I have the same question for gen_init_widget and ref3.

6. In frame_generator.xml, the list returned from build_implicit_label can have the 2nd element as null. This will happen for non-temp-tables, I think. I think the calling locations don't have the proper null safety.

7. In frame_generator.xml, I'm worried that the output_format_string changes may conflict with the precedence order we calculate in get_override_format_string.

8. FYI, we have an existing approach using tempIdx to implement deferred refid support. You've solved the same problem with your frefid. I'm not asking you to rewrite this, but I do wonder if the tempIdx could have been extended for your case (I suspect so).

9. SymbolResolver.annotateField() is called for both temp and non-temp fields. I thought that you only wanted to reference temp-table fields.

10. I think that the I18nOps internal use of TranslationManager.getInstance() is going to add performance overhead that will add up. What if we saved a reference to the TranslationManager instance as an instance member of each business logic class. Then we could make direct calls to it instead of having a context local lookup for every time a translated String literal is accessed.

11. initClassI18n would benefit by using Java AST templates.

12. annotations/i18n.rules, include/i18n.rules and TranslationManager.java need the GPL license added.

#43 - 10/01/2021 11:36 AM - Hynek Cihlar

Greg Shah wrote:

> Code Review Task Branch 3821c Revision 13017
>
> This is very good. I think the result is quite elegant and is layered in on our existing approach very well.
>
> 1. Does the 4GL Translation Manager operate on the fully preprocessed files? My guess is that it is aware of includes and allows references to strings in those. Our approach is based on the cache files, so it seems to me that there may be a "disconnect" between our approach and the original. For strings that are sourced from include files, this could mean that the same string appears hundreds or thousands of times in the application. I think this means we would duplicate the string in the PO files that same number of times. If I understand correctly, I think we cannot do that because it means that the people doing the translation must translate it many times in FWD but just once in the 4GL. That is not workable.

Actually TM does operate on fully preprocessed files. As a consequence one cannot import include files in TM, only procedure files. Btw. this is enforced by file extensions - *.p, *.w. Thus string literals in include files will be imported multiple times for each procedure file. TM handles this with the Glossary feature, the translator translates each unique phrase only once.

> 2. Do I understand correctly that annotations/i18n.rules will get changes to integrate the existing translations that were exported from the 4GL Translation Manager?

Correct. Btw. I also have to modify the TM sources as it doesn't export procedure file names.

> 3. In annotations/i18n.rules, why not change sprintf("%s/%s.%s", fdir, this.getAnnotation("classname"), "po") to be sprintf("%s/%s.po", fdir, this.getAnnotation("classname"))?

True, will fix.

> 4. The current process overwrites the .po file every time. This will need to change when translators start implementing changes post-conversion. In other words, post-conversion translators will be editing the .po files right? We will continue to run conversion on (at least some) files indefinitely. We can't blow away all those changes each time.

The generated PO files should serve only as input files for the translators I think. They should not serve as the storage for the translated outputs. I expect the workflow as follows: project is converted, the generated PO files are sent to translators, they translate and produce n sets of PO files for the n target languages, n sets of resource bundles are generated from the PO files and JARed together with the Java classes (or JARed in separate jar(s) so that the project doesn't need to be recompiled/rejared).

> 5. In frame_generator.xml the do_format changes add ref as a parameter to set_attr_ex_ex. It is not clear if that should be ref2 when ref2 is not null. I have the same question for gen_init_widget and ref3.

I think you may be right, I will look at this in more detail.

> 6. In frame_generator.xml, the list returned from build_implicit_label can have the 2nd element as null. This will happen for non-temp-tables, I think. I think the calling locations don't have the proper null safety.

I will check.

> 7. In frame_generator.xml, I'm worried that the output_format_string changes may conflict with the precedence order we calculate in get_override_format_string.

I tested the precedence order with test cases, but I will double check I got all the cases right.

> 8. FYI, we have an existing approach using tempIdx to implement deferred refid support. You've solved the same problem with your frefid. I'm not asking you to rewrite this, but I do wonder if the tempIdx could have been extended for your case (I suspect so).

I tried to use tempIdx but it seemed too complicated and at the time it was quicker to use UUIDs. I will put a TODO in the sources to evetually merge this.

> 9. SymbolResolver.annotateField() is called for both temp and non-temp fields. I thought that you only wanted to reference temp-table fields.

Non-temp field should not be annotated with frefid-* as its backing node should not hold fdefid- annotations.

> 10. I think that the I18nOps internal use of TranslationManager.getInstance() is going to add performance overhead that will add up. What if we saved a reference to the TranslationManager instance as an instance member of each business logic class. Then we could make direct calls to it instead of having a context local lookup for every time a translated String literal is accessed.

Yes, I was worried about this, too. Good idea, I will change this.

> 11. initClassI18n would benefit by using Java AST templates.

OK, I will think of how to apply templates here.

> 12. annotations/i18n.rules, include/i18n.rules and TranslationManager.java need the GPL license added.

Right, will fix.

#44 - 10/01/2021 11:44 AM - Hynek Cihlar

Hynek Cihlar wrote:

> Btw. I also have to modify the TM sources as it doesn't export procedure file names.

Perhaps I should contribute the change back in the github project.

#45 - 10/01/2021 11:50 AM - Greg Shah

> > Btw. I also have to modify the TM sources as it doesn't export procedure file names.
>
> Perhaps I should contribute the change back in the github project.

I have no objection. Do you think they will find the contribution to be useful?

#46 - 10/01/2021 12:04 PM - Hynek Cihlar

Greg Shah wrote:

> > > Btw. I also have to modify the TM sources as it doesn't export procedure file names.
> >
> > Perhaps I should contribute the change back in the github project.
>
> I have no objection. Do you think they will find the contribution to be useful?

I was more thinking about our workflow. We either have to somehow deliver the modified TM to them so that they can export valid data or we will have to implement a one-shot utility in 4GL to export the data.

#47 - 10/01/2021 12:17 PM - Greg Shah

Give it a try.

#48 - 10/02/2021 04:05 PM - Hynek Cihlar

A few notes about the import of Translation Manager translation database.

There are some limitations in the TM's model. The way I disambiguate duplicit messages in FWD is with a context id, which is an index of the message appearance in the source code. First occurrence of a string receives index 0, the next 1, etc. The advantage of this scheme is that the index will stay valid as long as the relative order of the strings won't change.

Unfortunately TM doesn't keep enough contextual data so that this scheme could be matched. The only usable piece of data relevant to this problem is line number of the source string. Even the line number data is not reliable as some message entries in TM database are missing line numbers completely. This is the case for some duplicit messages.

Thus I implemented the matching logic based on line numbers. When the line numbers don't match I match the strings in a particular procedure with the first TM same-string entry with missing line number. When no match I write a message in the standard error stream. When multiple same-string entries appear on the same line I use the first TM entry (in the exported csv order).

As a consequence it will be very important to have the TM database up to date with the converted 4GL sources.

#49 - 10/02/2021 04:08 PM - Hynek Cihlar

Hynek Cihlar wrote:

> As a consequence it will be very important to have the TM database up to date with the converted 4GL sources.

I should also note that this is a problem only when duplicit source messages are translated differently. When all the duplicit messages have the same translation, I just use the same translation for all the entries skipping the line number matching.

#50 - 10/03/2021 03:36 PM - Hynek Cihlar

- File adetran-master _extended_export.zip added

3821c revision 13021 adds support for import of the translation data exported from Translation Manager. Please review.

This is how to use it on a project:
1. Export translation data using the modified Translation Manager attached to this note. In the export dialog select UTF-8 encoding. Export all the target languages, one csv file per target language.
2. Add global parameter tm-translations to the project's p2j.cfg.xml. The value is comma separated list of exported CSV files. Specify file paths relative to the project root.
3. Run conversion. Look for the messages Failed to resolve TM translation for in the conversion output, these indicate a particular translation from the sources being converted didn't match with the exported translations.
4. When the conversion finishes the dir <project root>/translations will contain po files with all the translatable strings encountered in the converted sources and the matched translations. Resolve the unmatched translations and any untranslated strings by editing these po files.
5. From the project root execute p2j/tools/i18n/gen-bundles.sh. This will generate Java resource bundles from the po files and output them in <project root>/build/classes.

6. Execute ant jar to pack the resource bundles in the project jar file.
7. Execute ant deploy.prepare to deploy the project jar.

#51 - 10/03/2021 03:55 PM - Hynek Cihlar

- % Done changed from 80 to 90

#52 - 10/03/2021 07:17 PM - Greg Shah

Code Review Task Branch 3821c Revision 13021

The changes are all good.

#53 - 10/04/2021 03:52 AM - Hynek Cihlar

Hynek Cihlar wrote:

> Greg Shah wrote:
>
> > 3. In annotations/i18n.rules, why not change sprintf("%s/%s.%s", fdir, this.getAnnotation("classname"), "po") to be sprintf("%s/%s.po", fdir, this.getAnnotation("classname"))?
>
> True, will fix.

Fixed in 3821c revision 13021.

> > 5. In frame_generator.xml the do_format changes add ref as a parameter to set_attr_ex_ex. It is not clear if that should be ref2 when ref2 is not null. I have the same question for gen_init_widget and ref3.
>
> I think you may be right, I will look at this in more detail.

ref2 and ref both point to the same node and I didn't find any case where they would differ. Still I changed ref to ref2 as the argument to set_attr_ex_ex, this indeed seems more correct. In 3821c revision 13022.

ref3 in gen_init_widget is not used for converting translatable literals, if I interpret the code right.

> > 6. In frame_generator.xml, the list returned from build_implicit_label can have the 2nd element as null. This will happen for non-temp-tables, I think. I think the calling locations don't have the proper null safety.
>
> I will check.

This should be OK. The 2nd element (and potential null) is in all cases passed to set_attr_ex_ex where the null is checked.

> > 7. In frame_generator.xml, I'm worried that the output_format_string changes may conflict with the precedence order we calculate in get_override_format_string.
>
> I tested the precedence order with test cases, but I will double check I got all the cases right.

I'm looking at this.

> 10. I think that the I18nOps internal use of TranslationManager.getInstance() is going to add performance overhead that will add up. What if we saved a reference to the TranslationManager instance as an instance member of each business logic class. Then we could make direct calls to it instead of having a context local lookup for every time a translated String literal is accessed.

Yes, I was worried about this, too. Good idea, I will change this.

Fixed in 3821c revision 13022.

> 11. initClassI18n would benefit by using Java AST templates.

OK, I will think of how to apply templates here.

> 12. annotations/i18n.rules, include/i18n.rules and TranslationManager.java need the GPL license added.

Right, will fix.

Fixed in 3821c revision 13021.

Please review 3821c revision 13022.

#54 - 10/04/2021 06:17 AM - Greg Shah

Code Review Task Branch 3821c Revision 13022

It looks good.

#55 - 10/04/2021 03:22 PM - Hynek Cihlar

Hynek Cihlar wrote:

> Hynek Cihlar wrote:
>
>> Greg Shah wrote:
>>
>>> 7. In frame_generator.xml, I'm worried that the output_format_string changes may conflict with the precedence order we calculate in get_override_format_string.
>>
>> I tested the precedence order with test cases, but I will double check I got all the cases right.
>
> I'm looking at this.

I double checked all the format and label cases and compared the conversion output between 13016 and 13017, and I didn't find any differences. Except one regression I fixed in 3821c revision 13032.

#56 - 10/04/2021 03:25 PM - Greg Shah

Code Review Task Branch 3821c Revision 13032

It is good.

#57 - 10/08/2021 10:43 AM - Constantin Asofiei

Hynek, this review is for 3821c revisions 13032,13022,13021,13020,13017:

- The change in ProcedureManager.addProcedure - static classes use a java.lang.Object referent when they are loaded. So I don't think tm.applyLanguage(((TransparentWrapper) h.getResource()).get().getClass()); will work.
- TranslationManager - the instance is context-local, but you use ConcurrentHashMap - why is this needed?
- incremental conversion support - I can't tell how this is affected by incremental conversion. Can you test this?
- can you post some snippets how the converted code will look like, when translation is enabled? Also, how the bundles will look like?

#58 - 10/11/2021 03:26 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek, this review is for 3821c revisions 13032,13022,13021,13020,13017:
>
> - The change in ProcedureManager.addProcedure - static classes use a java.lang.Object referent when they are loaded. So I don't think tm.applyLanguage(((TransparentWrapper) h.getResource()).get().getClass()); will wor.

Interesting, I'm looking at this.

> - TranslationManager - the instance is context-local, but you use ConcurrentHashMap - why is this needed?

I'm not yet clear on all the possible use cases for TranslationManager. I assume it could be directly referenced from hand written code, not just the generated business and UI logic.

> - incremental conversion support - I can't tell how this is affected by incremental conversion. Can you test this?

Good point. This is already on my todo list.

> - can you post some snippets how the converted code will look like, when translation is enabled? Also, how the bundles will look like?

The bundles are generated by msgfmt, one of the GNU gettext commands. The code is generated with performance in mind. For example when there are no collisions between the individual translatable messages the implementation is based on String hashes and simple array lookups.

The following 4GL

```
message "Hello world!".
display "Hello world from a frame!" with frame f.
```

is converted as follows.

Converted Java codeConverted Java code

```
public class I18nSimple
{
   public static final long __trid = I18nOps.initReferent(com.goldencode.testcases.hynek.I18nSimple.class);

   public static final TranslationManager __tm = TranslationManager.getInstance();

   I18nSimpleF fFrame = GenericFrame.createFrame(I18nSimpleF.class, "f");

   public static String __tr(String msg)
   {
      return __tm.translate(__trid, msg);
   }

   public static String __tr(long ctxt, String msg)
   {
      return __tm.translate(__trid, ctxt, msg);
   }

   /**
    * External procedure (converted to Java from the 4GL source code
    * in hynek/i18n_simple.p).
    */
   @LegacySignature(type = Type.MAIN, name = "hynek/i18n_simple.p")
   public void execute()
   {
      externalProcedure(I18nSimple.this, new Block((Body) () ->
      {
         fFrame.openScope();
         message(__tr("Hello world!"));

         FrameElement[] elementList0 = new FrameElement[]
         {
            new Element(__tr("Hello world from a frame!"), fFrame.widgetExpr1())
         };

         fFrame.display(elementList0);
      }));
   }
}

...

public interface I18nSimpleF
extends CommonFrame
{
   public static final long __trid = I18nOps.initReferent(com.goldencode.testcases.hynek.I18nSimple.class);

   public static final TranslationManager __tm = TranslationManager.getInstance();

   public static final Class configClass = I18nSimpleFDef.class;

   public static String __tr(String msg)
   {
      return __tm.translate(__trid, msg);
   }

   public static String __tr(long ctxt, String msg)
   {
      return __tm.translate(__trid, ctxt, msg);
   }

   public void setExpr1(character parm);

   public void setExpr1(String parm);

   public void setExpr1(BaseDataType parm);

   public FillInWidget widgetExpr1();

   public static class I18nSimpleFDef
   extends WidgetList
   {
      FillInWidget expr1 = new FillInWidget();

      public void setup(CommonFrame frame)
      {
         frame.setDown(1);
         expr1.setDataType("character");
         expr1.setFormat(__tr("x(50)"));
         expr1.setFormat(__tr("x(50)"));
      }

      {
         addWidget("expr1", "", expr1);
      }
   }
}
```

Note setFormat issued twice in the frame definition. This is not a regression from the code changes.

This is the content of the po file generated by the conversion rules with translationsThis is the content of the po file generated by the conversion rules with translations

msgid ""
msgstr ""
"Language: Czech\n"
"Content-Type: text/plain; charset=UTF-8\n"

#: ./hynek/i18n_simple.p:1:9
msgid "Hello world!"
msgstr "Ahoj světe!"

#: ./hynek/i18n_simple.p:2:9
msgid "Hello world from a frame!"
msgstr "Ahoj světe z formuláře!"

#: ./hynek/i18n_simple.p:2:44
msgid "x(50)"
msgstr "x(50)"

And the generated bundleAnd the generated bundle

```
/* Automatically generated by GNU msgfmt.  Do not modify!  */
public class I18nSimple_Czech extends java.util.ResourceBundle {
  private static final java.lang.String[] table;
  static {
    java.lang.String[] t = new java.lang.String[12];
    t[0] = "";
    t[1] = "Language: Czech\nContent-Type: text/plain; charset=UTF-8\n";
    t[2] = "Hello world!";
    t[3] = "Ahoj sv\u011bte!";
    t[4] = "x(50)";
    t[5] = "x(50)";
    t[10] = "Hello world from a frame!";
    t[11] = "Ahoj sv\u011bte z formul\u00e1\u0159e!";
    table = t;
  }
  public java.lang.Object handleGetObject (java.lang.String msgid) throws java.util.MissingResourceException {
    int hash_val = msgid.hashCode() & 0x7fffffff;
    int idx = (hash_val % 6) << 1;
    java.lang.Object found = table[idx];
    if (found != null && msgid.equals(found))
      return table[idx + 1];
    return null;
  }
  public java.util.Enumeration getKeys () {
    return
      new java.util.Enumeration() {
        private int idx = 0;
        { while (idx < 12 && table[idx] == null) idx += 2; }
        public boolean hasMoreElements () {
          return (idx < 12);
        }
        public java.lang.Object nextElement () {
          java.lang.Object key = table[idx];
          do idx += 2; while (idx < 12 && table[idx] == null);
          return key;
        }
      };
  }
  public java.util.ResourceBundle getParent () {
    return parent;
  }
}
```

#59 - 10/11/2021 04:10 AM - Constantin Asofiei

Hynek Cihlar wrote:

> Constantin Asofiei wrote:
>
> > Hynek, this review is for 3821c revisions 13032,13022,13021,13020,13017:
> >
> > - The change in ProcedureManager.addProcedure - static classes use a java.lang.Object referent when they are loaded. So I don't think tm.applyLanguage(((TransparentWrapper) h.getResource()).get().getClass()); will wor.
>
> Interesting, I'm looking at this.

See ObjectOps.getStaticInstance how to resolve the java.lang.Class from a java.lang.Object referent. In addition, static properties/variables/etc are always defined context-local - but looking how the translation-related static fields are defined and used, I don't think this will be a problem.

Something else to note: how is a def var ch as char init "bla". (and other cases where the INIT clause is used) handled by the conversion? I think we need some tests with instance/static vars/properties in a legacy class, too.

And from the sample you posted: NameConverter I think already strips the leading underscore when converting a legacy name, so the __tr, __tm and other names should not collide.

#60 - 10/11/2021 07:51 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > Constantin Asofiei wrote:
> >
> > > Hynek, this review is for 3821c revisions 13032,13022,13021,13020,13017:
> > >
> > > - The change in ProcedureManager.addProcedure - static classes use a java.lang.Object referent when they are loaded. So I don't think tm.applyLanguage(((TransparentWrapper) h.getResource()).get().getClass()); will wor.
> >
> > Interesting, I'm looking at this.

It seems that any first reference of a class will fix its language for good. This is in contrast with external procedures. Once a procedure is removed from the call stack and there are no outstanding persistent references, any future procedure invocation will cause fixing its language again.

I'm testing all possibe cases to see if it is enough to just reference the class type or its field/proc/method must be referenced/invoked, or if the fixing behavior is more complex.

> Something else to note: how is a def var ch as char init "bla". (and other cases where the INIT clause is used) handled by the conversion? I think we need some tests with instance/static vars/properties in a legacy class, too.

String literals in init clauses are wrapped in __tr method as the other translatable literals.

#61 - 10/11/2021 08:26 AM - Greg Shah

I'd like to consider approaches that would eliminate the generation of the __tr() methods in every class/frame-def/menu-def.

What if we implemented something like this in TranslationManager?

```
   public TranslationHelper getHelper(final long trid)
   {
      final ResourceBundle bundle = referentBundleMap.get(trid);

      return (String msg) -> this.translateWorker(bundle, -1, msg);
   }

   public TranslationContextHelper getContextHelper(final long trid)
   {
      final ResourceBundle bundle = referentBundleMap.get(trid);

      return (long ctxt, String msg) -> this.translateWorker(bundle, ctxt, msg);
   }
```

TranslationHelper and TranslationContextHelper would be functional interfaces that provide a tr(String) and tr(long, String) methods respectively. The TranslationManager.translateWorker(ResourceBundle, long, String) would be a simplified version of TranslationManager.translate(long, long, String) that avoids the extra map lookup for the resource bundle.

The converted code would have something like this:

```
   private static final TranslationHelper __th = __tm.getHelper(__trid);

   private static final TranslationContextHelper __tch = __tm.getContextHelper(__trid);
```

The primary downside is the call to the helper is not as nice.

```
         message(__tr("Hello world!"));
```

becomes:

```
         message(__th.tr("Hello world!"));
```

Doing this is a bit faster and it has less generated code to maintain in each class, but the invocation is not as nice. Thoughts?

#62 - 10/11/2021 08:27 AM - Greg Shah

> It seems that any first reference of a class will fix its language for good.

Are you talking about the 4GL or FWD?

#63 - 10/11/2021 12:07 PM - Constantin Asofiei

Hynek Cihlar wrote:

> String literals in init clauses are wrapped in __tr method as the other translatable literals.

Please double-check with OO static/instance vars/properties, in these cases the conversion moves around ASTs, and that code is pretty fragile.

#64 - 10/11/2021 04:13 PM - Hynek Cihlar

Greg Shah wrote:

> > It seems that any first reference of a class will fix its language for good.
>
> Are you talking about the 4GL or FWD?

I was referring to 4GL.

#65 - 10/11/2021 04:19 PM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > String literals in init clauses are wrapped in __tr method as the other translatable literals.

> Please double-check with OO static/instance vars/properties, in these cases the conversion moves around ASTs, and that code is pretty fragile.

Yes, all these work OK.

#66 - 10/11/2021 04:29 PM - Hynek Cihlar

Greg Shah wrote:

> Doing this is a bit faster and it has less generated code to maintain in each class, but the invocation is not as nice. Thoughts?

Each approach has an advantage of its own. With the generated __tr method the resulting code is a bit cleaner, with the invocation helpers the two _tr methods are not needed. I incline more towards the generated __tr.

Btw. the resource bundle resolution in the invocation helpers would require an invalidation mechanism in the cases the procedure langugage needed to change.

#67 - 10/11/2021 04:41 PM - Greg Shah

> Btw. the resource bundle resolution in the invocation helpers would require an invalidation mechanism in the cases the procedure langugage needed to change.

Fair point. Let's stay with your current approach, at least for now. As you say, all the calling locations will read better.

#68 - 10/12/2021 05:46 AM - Hynek Cihlar

Constantin, 3821c revision 13058 contains fixes for legacy class language application. Plus some other more or less related changes. Please review.

#69 - 10/12/2021 05:51 AM - Constantin Asofiei

Hynek Cihlar wrote:

> Constantin, 3821c revision 13058 contains fixes for legacy class language application. Plus some other more or less related changes. Please review.

I'm OK with the changes.

#70 - 10/12/2021 02:28 PM - Greg Shah

What is the list of remaining items for this task?

#71 - 10/13/2021 02:52 AM - Hynek Cihlar

Greg Shah wrote:

> What is the list of remaining items for this task?

The remaining tasks are string expansion, which is WIP, and user documentation.

#72 - 10/13/2021 04:44 AM - Hynek Cihlar

I'd like to discuss the string expansion.

OE/4GL doesn't expand all the widgets the same. Some do expand, some not. Typically labels and fill-ins do expand, but combo boxes, toggle box labels or buttons don't. Even worse, those widgets that do not expand contain a subset of widgets that allow expansion to some extent such that the widget size will not change but the expanded text will use some of the widget's padding.

This is unfortunate as the implementation can't use a simple rule by expanding every translatable string on the server and make the client side widgets unaware about string expansion.

Instead the client side widgets will have to implement string expansion themselves. This will allow for any exception in the standard expansion behavior. For this to work, the client side widgets will need to have access to the original untranslated strings (beside the growth factor). Which means the value returned by __tr will have to hold both, the original string as well as the translation. Either somehow encoded in the String value, or in a character base data type or a new base data type extended from character.

Any thoughts?

#73 - 10/13/2021 07:43 AM - Greg Shah

> Which means the value returned by __tr will have to hold both, the original string as well as the translation. Either somehow encoded in the String value, or in a character base data type or a new base data type extended from character.

I really don't want to implement this way. It will be hard to understand and worse, it will make a mess of either the String data or the BDT hierarchy. I prefer to keep the knowledge of translations out of the BDT classes. I also don't want to make a mess of your nice, clean __tr() methods.

Since this is a behavior that is exclusive to the widget hierarchy, I want to limit the impact to that set of classes. It seems to me that the server side widgets can implement the necessary calls to the TranslationManager to obtain the data needed. Why not register the trid and TranslationManager instance with the frame in the business logic? Then the server side widgets can check if translations are active and if so, call in to calculate the proper additional values needed. The results can be passed down to the client as configuration.

> For this to work, the client side widgets will need to have access to the original untranslated strings (beside the growth factor).

Please help me understand why the client side widgets must know this. Can't the server side widgets calculate the proper values/limits and send down just the minimum data needed?

#74 - 10/13/2021 01:39 PM - Hynek Cihlar

Greg Shah wrote:

> > Which means the value returned by __tr will have to hold both, the original string as well as the translation. Either somehow encoded in the String value, or in a character base data type or a new base data type extended from character.
>
> I really don't want to implement this way. It will be hard to understand and worse, it will make a mess of either the String data or the BDT hierarchy. I prefer to keep the knowledge of translations out of the BDT classes. I also don't want to make a mess of your nice, clean __tr() methods.

I agree, this is unacceptable.

> Since this is a behavior that is exclusive to the widget hierarchy, I want to limit the impact to that set of classes. It seems to me that the server side widgets can implement the necessary calls to the TranslationManager to obtain the data needed. Why not register the trid and TranslationManager instance with the frame in the business logic? Then the server side widgets can check if translations are active and if so, call in to calculate the proper additional values needed. The results can be passed down to the client as configuration.

Currently widget sizing is implemented on the client, moving this, or its part to the server would require a nontrivial effort.

I still think I need to somehow get both the strings, the original and the translation, down on the client to the individual widgets. The actual widget can then decide how it will implement the expansion.

For labels this should be relatively simple. I could modify the conversion rules and pass the widget instance to __tr. The resulting map of widgets and their original string literals would then be pushed down to client in a screen definition.

For widget values (fill-in also resizes based on the expansion of the initial value) this would be a bit more complicated I suppose.

> > For this to work, the client side widgets will need to have access to the original untranslated strings (beside the growth factor).
>
> Please help me understand why the client side widgets must know this. Can't the server side widgets calculate the proper values/limits and send down just the minimum data needed?

I believe this would be a nontrivial effort. Widget sizing requires access to font subsystems of the gui drivers, which eventually requires Swing, Web browser 2D context, etc.

#75 - 10/13/2021 01:52 PM - Constantin Asofiei

Hynek, can you provide some screenshots/example how the string expansion affects the UI, if you switch between two languages? I assume the same .r is used for both languages, right? And this would mean that TranslationManager is used to 'inject' the translations into the .r code?

In other words, can this expansion be understood as 'when switching languages, the runtime replaces the translatable strings, and the UI will be built using those strings'?

I'm trying to understand how this expansion (and switching languages) affects the automatic widget sizing and layout.

#76 - 10/13/2021 01:58 PM - Constantin Asofiei

And you mention that some widgets expand and some don't. I think this depends on how OE builds the UI:

- for the widgets which don't expand, would mean that their size is computed before the translation is applied, and after that the translated string is set. But what if the translated string doesn't fit the button?
- for those which expand, would mean that the translation is applied first, and after that everything is done as if that is the original string in the .r code.

#77 - 10/13/2021 02:20 PM - Greg Shah

> For labels this should be relatively simple. I could modify the conversion rules and pass the widget instance to __tr. The resulting map of widgets and their original string literals would then be pushed down to client in a screen definition.
>
> For widget values (fill-in also resizes based on the expansion of the initial value) this would be a bit more complicated I suppose.

Wouldn't it be easier to register the TranslationManager instance and the trid with the frame? Then the server side widgets can do the __tr() call as needed. Through the normal widget processing, they have access to the original string which they could also pass to the client side. My point here is that we don't need to change all calls in the business logic. We just need to provide enough state to the server side frame so that the contained widgets can handle the rest inside the runtime.

If 2 strings and the expansion % is needed at the client for each widget, that is fine. I want to limit the impact on the converted code.

#78 - 10/13/2021 02:30 PM - Hynek Cihlar

- File 2021-10-13_20-16.png added

Constantin Asofiei wrote:

> Hynek, can you provide some screenshots/example how the string expansion affects the UI, if you switch between two languages?

See the attached screen shots. On the left, you will see demo_widgets.p with no translations compiled in. On the right, you will see the translated strings with effective expansion of 500% for strings up to 49 chars and 150% for strings 50 chars and up. The translations form the original string with "_1234567890" appended.

Btw. I'm puzzled with the translated fill-ins. They contain wrong values, somehow garbled. I haven't looked at this in more detail yet.

> I assume the same .r is used for both languages, right?

Correct.

> And this would mean that TranslationManager is used to 'inject' the translations into the .r code?

Correct.

> In other words, can this expansion be understood as 'when switching languages, the runtime replaces the translatable strings, and the UI will be built using those strings'?

I think so in a sense. But the string expansion calculation requires access to the original string too, to know the resulting expanded length.

#79 - 10/13/2021 02:48 PM - Hynek Cihlar

Greg Shah wrote:

>> For labels this should be relatively simple. I could modify the conversion rules and pass the widget instance to __tr. The resulting map of widgets and their original string literals would then be pushed down to client in a screen definition.
>>
>> For widget values (fill-in also resizes based on the expansion of the initial value) this would be a bit more complicated I suppose.
>
> Wouldn't it be easier to register the TranslationManager instance and the trid with the frame? Then the server side widgets can do the __tr() call as needed. Through the normal widget processing, they have access to the original string which they could also pass to the client side. My point here is that we don't need to change all calls in the business logic. We just need to provide enough state to the server side frame so that the contained widgets can handle the rest inside the runtime.

By doing this we break the GNU gettext workflow. It greps all translatable strings from the known translation method (in our case _tr). The _tr method can also "hold" any plural context info (which we currently don't implement but we could to provide more benefit to the FWD users). And last but not least, _tr also "holds" the context id to disambiguate duplicit strings.

Yes, we could keep the fancy __tr methods as markers. and do the actual translations on demand by server widgets (or the server runtime). This would require to create assignment of the context id to the widget instance during conversion.

#80 - 10/13/2021 03:50 PM - Constantin Asofiei

Hynek Cihlar wrote:

> I think so in a sense. But the string expansion calculation requires access to the original string too, to know the resulting expanded length.

Sorry, but this does not really make sense in terms of how it would affect the implicit widget ~~length~~ width.   You note somewhere that:

> string length equals to [original length] + [defined percentual value] * [entry in the internal table]

So you are thinking that this 'translated length' is what drives the widget ~~length~~ width?   I'm confused because OE does the layout based on the text metrics and the widget's font, and I really doubt it does it in any other way for translated case, to.

In your screenshot, there is something interesting: I don't see any : (colon) character for the labels. So, I wonder if the 'expansion' just means "add this amount of spaces to the translated string".

Please check this by getting the LABEL, SCREEN-VALUE or whatever attribute, for a widget, a log it somewhere - check both the LENGTH and the actual value of the attribute, something like: MESSAGE LENGTH(h:LABEL) "[" h:LABEL "]". - the idea is to not trim the whitespace.

#81 - 10/13/2021 03:57 PM - Constantin Asofiei

And something else which can be done to check if OE has the same mechanism of resolving the implicit widget size, in case of translated strings (beside the exceptions you mentioned where the original string is used):

- check the WIDTH-PIXELS and WIDTH-CHARS for the translated i.e. fill-in: label
- in a separate program, where no translation is used/applied, use the actual LABEL text for the widget and build a FILL-IN with that actual text (if the LABEL is fill-in_1234567890   - with trailing spaces, use that text exactly). Check the WIDTH-PIXELS and WIDTH-CHARS value for this LABEL.

#82 - 10/13/2021 04:53 PM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > I think so in a sense. But the string expansion calculation requires access to the original string too, to know the resulting expanded length.
>
> Sorry, but this does not really make sense in terms of how it would affect the implicit widget length.   You note somewhere that:
>
> > string length equals to [original length] + [defined percentual value] * [entry in the internal table]
>
> So you are thinking that this 'translated length' is what drives the widget length?   I'm confused because OE does the layout based on the text metrics and the widget's font, and I really doubt it does it in any other way for translated case, to.

No, what drives the widget size is the original string with an added number of spaces. The number of spaces is calculated from the expansion factor. Only then is the translated string placed in this calculated space, and the string may or may not fit.

> In your screenshot, there is something interesting: I don't see any : (colon) character for the labels. So, I wonder if the 'expansion' just means "add this amount of spaces to the translated string".

The invisible colons is just a coincidence. It is not shown for the strings, which are too long to fit in the calculated space. Note there is one colon visible, in the radio widget in the top frame.

> Please check this by getting the LABEL, SCREEN-VALUE or whatever attribute, for a widget, a log it somewhere - check both the LENGTH and the actual value of the attribute, something like: MESSAGE LENGTH(h:LABEL) "[" h:LABEL "]". - the idea is to not trim the whitespace.

The LABEL value is always the full translated (or original) text and right-padded with spaces to top the calculated expanded length. The WIDTH always shows the actual size, no surprise here either.

#83 - 10/14/2021 04:01 AM - Constantin Asofiei

I'm looking at the 4GL code for the TranslationManager (adetran-master.zip) and it looks like when translating, this 'fixes' the width of the widgets, and it no longer relies on the automatic layout. See src/main/abl/adetran/pm/_build.p line 673 for EDITOR:

```
          Flds = 'DEFINE VARIABLE ':U + _tpfield._widget +
                 ' AS CHARACTER VIEW-AS EDITOR':U + CR +
                 '  SIZE ':U + STRING(_tpfield._width) + ' BY ':U + STRING(_tpfield._height - .2) + CR .
      IF tDispType <> "c":U THEN
      Flds = Flds +
                       (IF _tpfield._tooltip <> '':U AND _tpfield._tooltip <> ? THEN
                        ' TOOLTIP "':U + _tpfield._tooltip + '"':U ELSE '':U) .
      Flds = Flds +
                 (IF _tpfield._scrollbar-h THEN '  SCROLLBAR-HORIZONTAL':U ELSE ' ':U) +
                 (IF _tpfield._scrollbar-v THEN ' SCROLLBAR-VERTICAL':U ELSE '':U) +
                 (IF _tpfield._label <> ?
                    THEN '  LABEL "':U + _tpfield._label + '"':U + CR
                    ELSE '':U) + ColorsFonts + '.':U.
          PUT UNFORMATTED Flds SKIP(1).
```

Hynek, I may be wrong, but can I assume that to add a translation, the 4GL code is generated/transformed? Do you have a sample for 'before-and-after' of i.e. demo_widgets.p source code?

I ask because what you describe still doesn't make sense. I don't see how they could mess up the UI so bad that another string is used somewhere to calculate the width/height.

If the translation alters the code and fixes the widget size, then we should find what the rules are, and fix the size in the same way.

#84 - 10/14/2021 04:56 AM - Constantin Asofiei

- File lbl_length.png added

I can't find if it re-generates 4GL code, but the COMPILE command is this (see _compdlg.w), and my previous assumption is most likely wrong:

```
      COMPILE VALUE(Full_Name)
              SAVE INTO VALUE(SaveInto)
              LANGUAGES (VALUE(Languages))
              TEXT-SEG-GROWTH = Growth.
```

So the TEXT-SEG-GROWTH is something which exists in OpenEdge, and not related to TranslationManager application.

From the docs https://docs.progress.com/bundle/openedge-abl-reference-117/page/COMPILE-statement.html:

> TEXT-SEG-GROW = growth-factor
> Specifies the factor by which ABL increases the length of strings. When you develop an application that is going to be translated, it is important to allow for the growth of the text in your widgets. If you use the TEXT-SEG-GROW option, > ABL increases the size of the text strings when it compiles your application.
> ABL uses the following formula to determine the length of strings:
>
> > New-length = Actual-length * (1 + (growth-factor/100 * (table-value/100)))

They provide also the expansion table.

I missed something from your screenshot at #3817-78 - the right-side has the translations built-in, and I somehow assumed that the expansion is done after you switch languages, but that is not the case.

So they do alter the 4GL code, but only via TEXT-SEG-GROWTH, and this can be seen only in the .r-code - which we don't have access to.

Back to assumptions: in the demo_widgets.p, do you have a screenshot where the languages are applied to the .r-code, but you use the 'original' language? I mean, to have the same text as the left-side window. Because it seems that the layout is the same once languages are applied, and the layout doesn't change if you switch languages, right?

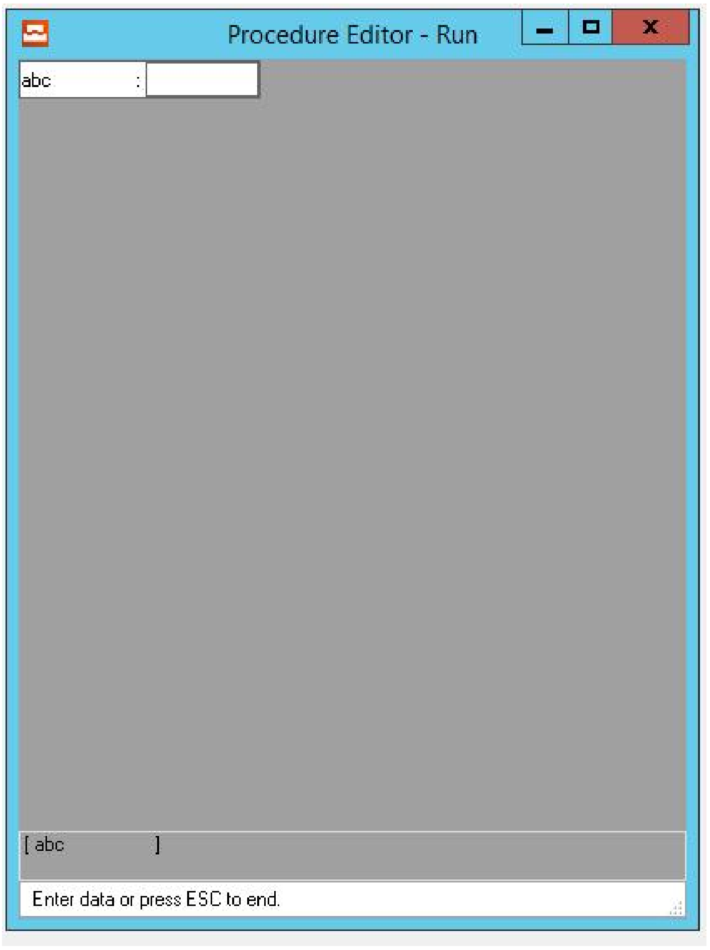
You mention somewhere that:

> Adding the string max length option ("string":10) will disable expansion for this particular string.

See this example:

```
def var ch as char.
form ch label "abc":20 with frame f1 side-labels.
display ch with frame f1.
message "[" ch:label "]".
update ch with frame f1.
```

which displays this:



```
abc          :
[ abc              ]
Enter data or press ESC to end.
```

So the actual string is modified when the string-length is specified at the literal.

So I wonder: if TEXT-SEG-GROW is used, maybe OE injects the :<string-length> and forces the label to append spaces, but only if it is smaller than this length.

But this does not explain why e.g. BUTTON is not affected. Please check if AUTO-RESIZE flag at the BUTTON is still set to 'true' as default, and if you switch the LABEL of a BUTTON it gets auto-resized, when languages are applied.

What I'm trying to figure out (and maybe prove) with all the above is that these layout/size differences we are seeing are a result of COMPILE changes made in the r-code, and not something related to the runtime.

#85 - 10/14/2021 05:12 AM - Constantin Asofiei

Hynek, something else: we may need to add support for RCODE-INFO:LANGUAGES attribute. This is the way OE saves and provides info about what languages are compiled into a .r file.

And test what happens if you set a language which is not compiled into the r-code.

#86 - 10/14/2021 05:15 AM - Hynek Cihlar

Constantin Asofiei wrote:

> I'm looking at the 4GL code for the TranslationManager (adetran-master.zip) and it looks like when translating, this 'fixes' the width of the widgets, and it no longer relies on the automatic layout. See src/main/abl/adetran/pm/_build.p line 673 for EDITOR:
> [...]

TM is capable to build mock UI for the translated procedures. This is to allow the translators to see the UI in action. The source code you posted is part of this feature implementation. I.e. it has nothing to do with the resulting r-code I think.

> Hynek, I may be wrong, but can I assume that to add a translation, the 4GL code is generated/transformed? Do you have a sample for 'before-and-after' of i.e. demo_widgets.p source code?

No, it doesn't touch the original 4GL sources.

> I ask because what you describe still doesn't make sense. I don't see how they could mess up the UI so bad that another string is used somewhere to calculate the width/height.
>
> If the translation alters the code and fixes the widget size, then we should find what the rules are, and fix the size in the same way.

I don't think the translation fixes the widget size. What it does, it only expands the string literals. The layout logic doesn't need to be altered, I think. It performs the same layout operation, just on the expanded literals. This introduces more space to handle translations, which are more chatty.

#87 - 10/14/2021 05:25 AM - Hynek Cihlar

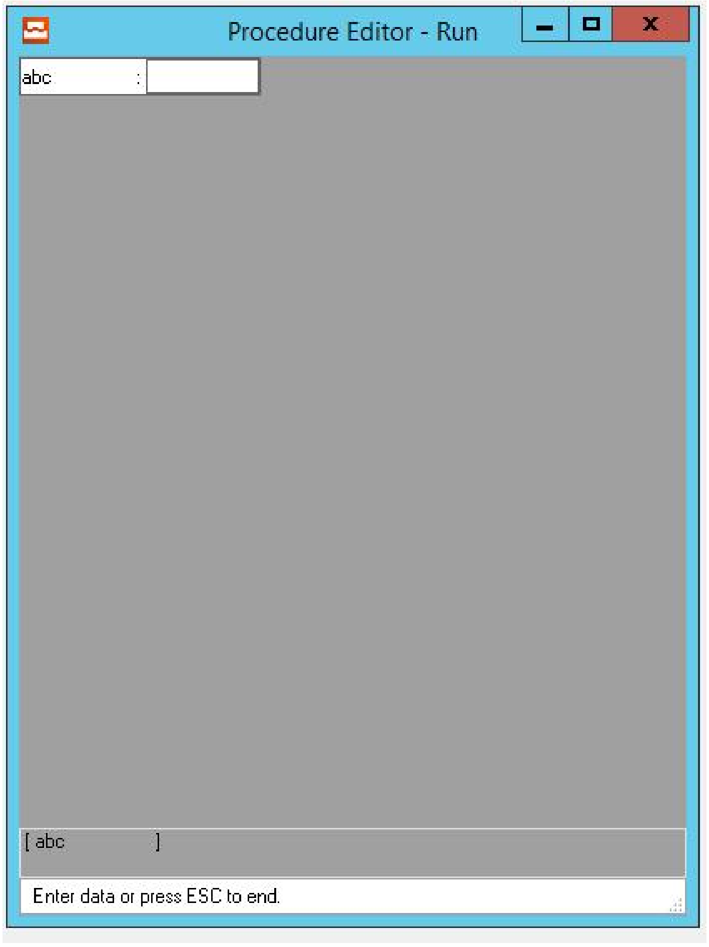
Constantin Asofiei wrote:

> Back to assumptions: in the demo_widgets.p, do you have a screenshot where the languages are applied to the .r-code, but you use the 'original' language? I mean, to have the same text as the left-side window.
>
> Because it seems that the layout is the same once languages are applied, and the layout doesn't change if you switch languages, right?

Exactly so. If the original language is used on the TM built r-code with growth factor applied, the layout will come up exactly the same as the translations. The original string literals will be expanded by extra spaces.

> You mention somewhere that:
>
>> Adding the string max length option ("string":10) will disable expansion for this particular string.

Yes, such a string literal won't be expanded.

> See this example:
> [...]
>
> which displays this:
>
> 
>
>
> So the actual string is modified when the string-length is specified at the literal.
>
> So I wonder: if TEXT-SEG-GROW is used, maybe OE injects the :<string-length> and forces the label to append spaces, but only if it is smaller than this length.

Seems like it.

> But this does not explain why e.g. BUTTON is not affected. Please check if AUTO-RESIZE flag at the BUTTON is still set to 'true' as default, and if you switch the LABEL of a BUTTON it gets auto-resized, when languages are applied.

Some of the widgets are straight Win32 controls, including buttons for example. This could be an oversight of their implementation. The auto-resize is not affected by the expansion.

> What I'm trying to figure out (and maybe prove) with all the above is that these layout/size differences we are seeing are a result of COMPILE changes made in the r-code, and not something related to the runtime.

IMHO the expansion really only appends spaces to the literals before it saves them in the TEXT section in the r-file. I don't think it does anything else, especially mangling with the frame layouts.

#88 - 10/14/2021 05:41 AM - Constantin Asofiei

Hynek, please check this: have a label with the original string as iiiii and its translation as WWWWW, no growth factor. How will the label look if the 'translation' language is used? My assumption is that its width will be based on the iiiii string.

I'm thinking that the layout is done with the original string (in some cases expanded, in other cases not), and the translation text is applied only after that.

For buttons and others, maybe the expansion is not applied at all. So this would mean that we can't expand the string at conversions time, and need to save the expansion value with the string (so the widget can ignore it).

#89 - 10/14/2021 05:51 AM - Greg Shah

> IMHO the expansion really only appends spaces to the literals before it saves them in the TEXT section in the r-file. I don't think it does anything else, especially mangling with the frame layouts.

Good. It seems like we can implement this purely on the server side. We can store the growth-factor as a frame configuration value and the server widget code can pad the string literals when they are loaded.

We could hard code the padding at conversion, but that seems to break the concept for any post-conversion development.

Am I missing anything?

#90 - 10/14/2021 05:52 AM - Greg Shah

BTW, the formula in the documentation seems wrong. They write it as:

New-length = Actual-length * (1 + (growth-factor/100 * (table-value/100)))

Their example (actual-length = 25, growth-factor = 50, table-value = 80 and new-length = 35), suggests that it should be this (the closing parenthesis has been moved for clarity):

New-length = Actual-length * (1 + (growth-factor/100) * (table-value/100))

#91 - 10/14/2021 07:12 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek, please check this: have a label with the original string as iiiii and its translation as WWWWW, no growth factor. How will the label look if the 'translation' language is used? My assumption is that its width will be based on the iiiii string.
>
> I'm thinking that the layout is done with the original string (in some cases expanded, in other cases not), and the translation text is applied only after that.

Yes, this seems to be the case.

#92 - 10/14/2021 07:17 AM - Hynek Cihlar

Greg Shah wrote:

> > IMHO the expansion really only appends spaces to the literals before it saves them in the TEXT section in the r-file. I don't think it does anything else, especially mangling with the frame layouts.
>
> Good. It seems like we can implement this purely on the server side. We can store the growth-factor as a frame configuration value and the server widget code can pad the string literals when they are loaded.

Perhaps I'm missing your point. The (widget) layout logic on the client must receive the padded original untranslated strings and do the layout based on them. When the layout is done the widgets must then use the translations, whichever are in effect for the external procedure (or legacy class).

#93 - 10/14/2021 07:29 AM - Greg Shah

> The (widget) layout logic on the client must receive the padded original untranslated strings and do the layout based on them. When the layout is done the widgets must then use the translations, whichever are in effect for the external procedure (or legacy class).

Yes, I think we are on the same page. All the translation processing and the actual expansion behavior can be on the server side. I think all we have to do is optionally pass a new configuration value for the widget, perhaps called something like layout_text_value which can be the padded original text value. And the translated value can be resolved on the server and simply set as the value or label for the widget using the normal processing.

#94 - 10/14/2021 07:48 AM - Hynek Cihlar

Greg Shah wrote:

> > The (widget) layout logic on the client must receive the padded original untranslated strings and do the layout based on them. When the layout is done the widgets must then use the translations, whichever are in effect for the external procedure (or legacy class).
>
> Yes, I think we are on the same page. All the translation processing and the actual expansion behavior can be on the server side. I think all we have to do is optionally pass a new configuration value for the widget, perhaps called something like layout_text_value which can be the padded original text value. And the translated value can be resolved on the server and simply set as the value or label for the widget using the normal processing.

Exactly :-).

#95 - 10/25/2021 04:44 AM - Hynek Cihlar

I can't crack the logic how OE calculates the extra width for the expanded strings.

My first naive idea was to simply measure the width of the text appended with empty spaces, but this doesn't come out correct, the result is always smaller than what OE produces. Then I tried measuring width of different characters, but didn't find a match either. Then I tried to use max char and average char widths from the font metrics, but no luck there either.

My fallback idea is to pre-calculate it and store the calculated values in directory. With two procedure files with the same frame/label layout and one of them translated with a known expansion value, I could compare the pixel widths of the expanded and non-expanded labels and from the known number of expanded chars calculate the pixel size for each char. This calculation would be done for every font configuration used and then stored in the directory.

#96 - 10/25/2021 03:06 PM - Greg Shah

> My first naive idea was to simply measure the width of the text appended with empty spaces, but this doesn't come out correct, the result is always smaller than what OE produces. Then I tried measuring width of different characters, but didn't find a match either. Then I tried to use max char and average char widths from the font metrics, but no luck there either.

The 4GL could be measuring the total string width using the WIN32 APIs and then scaling it by the growth factor. This would include all the kerning and would be variable based on the exact text of the string. That would explain why these other approaches did not match. Did you try this approach?

This would mean that the calculation could be unique per string, rather than something that is global for a font.

#97 - 10/26/2021 02:03 AM - Hynek Cihlar

Greg Shah wrote:

> > My first naive idea was to simply measure the width of the text appended with empty spaces, but this doesn't come out correct, the result is always smaller than what OE produces. Then I tried measuring width of different characters, but didn't find a match either. Then I tried to use max char and average char widths from the font metrics, but no luck there either.
>
> The 4GL could be measuring the total string width using the WIN32 APIs and then scaling it by the growth factor. This would include all the kerning and would be variable based on the exact text of the string. That would explain why these other approaches did not match. Did you try this approach?
>
> This would mean that the calculation could be unique per string, rather than something that is global for a font.

Different strings expanded by the same amount of chars yield the same added width. Thus OE either picks a character, appends it to the text multiple times to make the expanded text and then measures the text, or it uses a font metric value to advance the result by the number of the appended chars, or something else.

#98 - 10/26/2021 02:05 AM - Constantin Asofiei

Isn't the expansion made by adding spaces? And if the same number of spaces are added, the width should match.

#99 - 10/26/2021 02:41 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Isn't the expansion made by adding spaces? And if the same number of spaces are added, the width should match.

While the resulting label value on the screen does contain spaces, they are not used to measure the text width. At least measuring the text width with spaces doesn't yield the OE width.

#100 - 10/26/2021 03:07 AM - Constantin Asofiei

Hynek Cihlar wrote:

> While the resulting label value on the screen does contain spaces, they are not used to measure the text width. At least measuring the text width with spaces doesn't yield the OE width.

Have you tried using WIDTH-PIXELS on the side-label, fill-in, etc?

#101 - 10/26/2021 03:20 AM - Hynek Cihlar

Constantin Asofiei wrote:

> Hynek Cihlar wrote:
>
> > While the resulting label value on the screen does contain spaces, they are not used to measure the text width. At least measuring the text width with spaces doesn't yield the OE width.
>
> Have you tried using WIDTH-PIXELS on the side-label, fill-in, etc?

Yes.

#102 - 10/26/2021 03:44 AM - Constantin Asofiei

FILL-IN has a peculiar way of computing the width - see the nativeWidth() method for FillInGuiImpl. For LABEL, its width in pixels follows the text width (for that font).

So, if you use the same font, take the 'expanded text with spaces' for a LABEL, and use it in a standalone test without translations active, does this give in OE the same width-pixels?

You can also try using a fixed font (all chars have the same width) and check how the width compares.

OE either fixes the width (by measuring the initial text and adding a width multiplied with the number of added 'spaces' or some other weird formula), or it uses the expanded text to compute it automatically.

As a side note, determining the implicit width for the widgets was very complex, at least for FILL-IN. See the nativeWidth method at the GUI widgets, how this implicit width is computed, to be compatible with OE.

#103 - 10/26/2021 06:20 AM - Hynek Cihlar

Constantin Asofiei wrote:

> FILL-IN has a peculiar way of computing the width - see the nativeWidth() method for FillInGuiImpl. For LABEL, its width in pixels follows the text width (for that font).
>
> So, if you use the same font, take the 'expanded text with spaces' for a LABEL, and use it in a standalone test without translations active, does this give in OE the same width-pixels?

At this moment, this is how I implement the expansion. And it doesn't yield the same label width.

> You can also try using a fixed font (all chars have the same width) and check how the width compares.

I tried all the default fonts: Courier New and MS Sans Serif, and also some others. I even tried to derive the expansion logic from PIXELS-PER-COLUMN session attribute.

#104 - 11/02/2021 06:21 PM - Hynek Cihlar

3821c revision 13125 implements i18n string expansion and also contains other related changes.

The calculation of the expanded label pixel width turned out to be a simple text width calculation of the space-expanded string. I had misinterpreted the length expansion, which always led to incorrect character lengths and so the pixel widths were always off.

The expansion adds new p2j.cfg.xml parameter text-expansion and directory attribute text-expansion. This is the percent amount of how much the strings must be expanded. The conversion parameter is used to expand the literals being added to ui_strings.txt and the directory parameter is used by the runtime to expand the translated characters. I will add these details to the wiki documentation.

The remaining task for this issue is the wiki documentation. I plan to add the documentation once the review is finished.

Please review.

#105 - 11/03/2021 09:50 AM - Greg Shah

Code Review Task branch 3821c Revision 13125

Overall, this is very good.

1. In set_attr_ex_ex we have this code:

```
                  <action>uiStrings.add(aval)</action>
                  <action>uiStrings.add(aval.concat(":"))</action>
                  <action>uiStrings.add(aval.concat(": "))</action>
```

Why is aval2 only being added for the ": " case and not the others?

2. In I18nWorker, let's protect the used of Double.parseDouble(expStr) in case the expStr is not a valid double. Such a case should emit a warning and set a safe default behavior.

3. In ZCL.labelWidth(), the code appends ": " to the end of un. For labelText this is sometimes done at the caller and sometimes it is not done. Is it always safe to append this to un here?

4. I think the initClassI18n AST creation code would benefit from use of templates. This is for future reference. Do **not** rewrite the code at this time.

#106 - 11/03/2021 01:01 PM - Hynek Cihlar

Greg Shah wrote:

> Code Review Task branch 3821c Revision 13125
>
> Overall, this is very good.

> 1. In set_attr_ex_ex we have this code:
>
> [...]
>
> Why is aval2 only being added for the ": " case and not the others?

Because the value only applies to labels where we ever need to consider the ": " case.

> 2. In I18nWorker, let's protect the used of Double.parseDouble(expStr) in case the expStr is not a valid double. Such a case should emit a warning and set a safe default behavior.

I deliberately let the exception propagate up so that the conversion fails early. But if you think a warning is enough I will change it.

> 3. In ZCL.labelWidth(), the code appends ": " to the end of un. For labelText this is sometimes done at the caller and sometimes it is not done. Is it always safe to append this to un here?

The width should be always measured with ": " appended. untranslatedLabel doesn't contain the colon.

> 4. I think the initClassI18n AST creation code would benefit from use of templates. This is for future reference. Do **not** rewrite the code at this time.

OK.

#107 - 11/03/2021 01:22 PM - Greg Shah

> I deliberately let the exception propagate up so that the conversion fails early. But if you think a warning is enough I will change it.

Yes, please do.

#108 - 11/07/2021 01:07 PM - Hynek Cihlar

Please review the documenation at Translations.

#109 - 11/07/2021 01:07 PM - Hynek Cihlar

- Status changed from WIP to Review

- % Done changed from 90 to 100

#110 - 11/08/2021 06:35 AM - Hynek Cihlar

Greg Shah wrote:

> > I deliberately let the exception propagate up so that the conversion fails early. But if you think a warning is enough I will change it.
>
> Yes, please do.

In 3821c revision 13140.

#111 - 11/08/2021 03:45 PM - Greg Shah

Documentation Review

It is very good.

1. I moved this into our Conversion Handbook, in the existing Internationalization chapter as the Translations section. I have temporarily "included" the original wiki page (renamed) so that it is easy to maintain and review there.

2. I made some edits, mostly to the formatting but a few were content related. Please review. If you are OK with the changes, I will move them into that chapter permanently instead of using the include macro.

3. Please add the following:

- Show an example of the 4GL and converted code so that the reader can understand what you mean by "enclose translatable literals in translation methods" and in general how translatable literals in the 4GL are converted to translatable literals in the Java code. Please also mention which runtime classes contain the critical implementation. I guess this stuff might be in an "Implementation Details" section.
- Some description of what Java resource bundles are and links to references that can explain how they work.
- Some suggestions about gettext translation utilities/tooling that is available. It is important for readers to understand that there are useful tools available and have some pointers to find them.
- It is not clear how to run <project_root>/p2j/tools/i18n/gen-bundles.sh. Are there parameters? The tool "lives" inside FWD but so how does it know where to find the .po files and where to output the bundles?

4. I think that some standardized changes are needed to our project configuration (the ant build.xml) to include support for translations. It seems to me that (when needed) it should run the gen-bundles.sh and include the resource bundles into the jar.

#112 - 11/09/2021 03:10 AM - Hynek Cihlar

Greg Shah wrote:

> Documentation Review
>
> It is very good.
>
> 1. I moved this into our Conversion Handbook, in the existing Internationalization chapter as the Translations section. I have temporarily "included" the original wiki page (renamed) so that it is easy to maintain and review there.
>
> 2. I made some edits, mostly to the formatting but a few were content related. Please review. If you are OK with the changes, I will move them into that chapter permanently instead of using the include macro.

The changes are good. I made only one minor change. Please move the page.

> 3. Please add the following:
>
> - Show an example of the 4GL and converted code so that the reader can understand what you mean by "enclose translatable literals in translation methods" and in general how translatable literals in the 4GL are converted to translatable literals in the Java code. Please also mention which runtime classes contain the critical implementation. I guess this stuff might be in an "Implementation Details" section.
> - Some description of what Java resource bundles are and links to references that can explain how they work.
> - Some suggestions about gettext translation utilities/tooling that is available. It is important for readers to understand that there are useful tools available and have some pointers to find them.

Will do all the points above.

> - It is not clear how to run <project_root>/p2j/tools/i18n/gen-bundles.sh. Are there parameters? The tool "lives" inside FWD but so how does it know where to find the .po files and where to output the bundles?

The tool takes advantage of the standard project structure and the directory layout mentioned in the wiki page. But it can also take arguments to override these. I will add more details to clarify this.

> 4. I think that some standardized changes are needed to our project configuration (the ant build.xml) to include support for translations. It seems to me that (when needed) it should run the gen-bundles.sh and include the resource bundles into the jar.

Good idea, will do.

#113 - 11/09/2021 06:56 AM - Greg Shah

> Please move the page.

Done.

#114 - 11/09/2021 07:40 AM - Greg Shah

I've exposed the download of the extended version of the TM in Translation Manager Extended Export Version and I modified the documentation to match.

#115 - 11/15/2021 12:32 PM - Hynek Cihlar

Greg Shah wrote:

> 4. I think that some standardized changes are needed to our project configuration (the ant build.xml) to include support for translations. It seems to me that (when needed) it should run the gen-bundles.sh and include the resource bundles into the jar.

Greg, did you mean the Ant target(s) should be added to the build file of an application project, like Hotel GUI?

#116 - 11/15/2021 02:19 PM - Greg Shah

> < did you mean the Ant target(s) should be added to the build file of an application project, like Hotel GUI?

Yes, exactly that.

#117 - 01/10/2022 02:44 PM - Hynek Cihlar

Hynek Cihlar wrote:

> Greg Shah wrote:
>
> > 3. Please add the following:
> >
> > - Show an example of the 4GL and converted code so that the reader can understand what you mean by "enclose translatable literals in translation methods" and in general how translatable literals in the 4GL are converted to translatable literals in the Java code. Please also mention which runtime classes contain the critical implementation. I guess this stuff might be in an "Implementation Details"

section.

- Some description of what Java resource bundles are and links to references that can explain how they work.
- Some suggestions about gettext translation utilities/tooling that is available. It is important for readers to understand that there are useful tools available and have some pointers to find them.

Will do all the points above.

Please see the updated Internationalization

#118 - 01/10/2022 09:10 PM - Greg Shah

- Status changed from Review to Closed

Nice!

Files

progress_software_knowledge_base_article_frequently_asked_questions_regarding_the_translation_manager_open_source_initiative_20201120.pdf	63.4 KB	09/02/2021	Greg Shah
progress_software_knowledge_base_article_translation_manager_and_visual_translator_are_now_open_source_20201120.pdf	45.2 KB	09/02/2021	Greg Shah
gquerret_adetran_ Progress OpenEdge Visual Translator and Translation Manager archive.pdf	266 KB	09/02/2021	Hynek Cihlar
adetran-master.zip	964 KB	09/02/2021	Hynek Cihlar
2021-09-27_20-28.png	60.9 KB	09/27/2021	Hynek Cihlar
adetran-master _extended_export.zip	964 KB	10/03/2021	Hynek Cihlar
2021-10-13_20-16.png	96.3 KB	10/13/2021	Hynek Cihlar
lbl_length.png	5.04 KB	10/14/2021	Constantin Asofiei