

enable

makes PowerBuilder applications multilingual

Understanding Enable: an interview with Gian Luca De Bonis

Before **TechWave 2007** we met with Gian Luca De Bonis who is CEO/CTO of Enable Development, producer of the framework tool that makes PowerBuilder applications multilingual. Key benefits claimed include the retention of complete developer control, no duplication of source code and dynamic change of the active language. We wanted to know the rationale behind Enable, something about its technical characteristics and what sort of effort is required at the implementation stage.

Can you tell us why you have developed Enable?

GLDB: Many PowerBuilder users with monolingual applications would like to expand into international markets, but have been held back by the costs of developing and maintaining multilingual software. I don't mean the translation effort, but rather the technical and practical issues. Enable provides a complete and affordable solution to these problems.

But doesn't the PB Translation Toolkit resolve this issue?

GLDB: TTK generates separate monolingual source code for each foreign language. This is fine for small, stable programs, but many substantial business applications evolve continuously, with new releases every few months. In this latter case, experience of the code regeneration, recompilation and redistribution cycle shows that the maintenance costs can escalate dramatically, as a function of the frequency of releases and the number of languages administered. With Enable, there is just one source code and one executable application, so the extra cost of new releases is pretty much limited to managing the new translated text and distributing the updated multilingual database. Even the addition of an extra language does not involve coding changes of any kind.

You mentioned some technical issues?

GLDB: Yes, the localization of software generally calls for major re-engineering work. The most well-known technique, commonly applied in C++ and Java, consists in replacing strings with identifiers that have long, descriptive names, such as ID_MSG_PRESS_KEY_W422: a somewhat lengthy process in most cases. These

strings are read into memory when the application is run. The selected language cannot be changed dynamically, since reloading the language strings would also mean having to refresh all the open windows, requiring a major overhaul of the source code.

By contrast, the implementation of Enable has no impact on code legibility.

Many users avoid such issues by installing an emulator...

GLDB: Bearing in mind that emulators are not always easy to configure, this approach is good for users without access to the application's source code, since they can rely on an overlay that intercepts calls to the operating system, searches a database for the required strings and replaces them with the related translations. Ignoring the economic costs involved and certain quality issues, this situation is less than satisfactory for PB developers since they lose control over their software at runtime. A key strength of Enable, on the other hand, is that it becomes part of the developer's own application framework, thus ensuring that full control is maintained at all times.

Before, did you imply that Enable allows dynamic change of the active language?

GLDB: That's right! This takes place:

- under the control of the developer, who decides what the user can do;
- throughout the application covering all types of control and all properties;
- in windows, reports, generated files and communications with peripheral devices;
- on opening windows, but also when windows are already open.

Indeed, several languages can be active at any one time, for the generation of reports in languages other than the user interface. This allows the user to retain control of the application, unlike with emulators which normally change the language of the entire user interface.

Yes, but where does all this leave the users of the software?

GLDB: Right where they were. Enable achieves this dynamic change without interrupting the work flow or affecting the functioning of the application. The DataWindow and Edit buffers remain unaltered, as do the end-user's settings: everything is retained when the language is changed. Rapid switching between languages is therefore possible, thus facilitating the work of support personnel, for example, who can change the application's language without interfering with the user's activities, and then switch it back again when they have finished.

What about the technical characteristics of Enable?

GLDB: Enable has been designed specifically for PowerBuilder, in all its forms: Client/Server, Distributed, PocketBuilder. All recent versions of PowerBuilder are



supported, from 8 to 11, and Enable also functions with PowerBuilder 5, 6.5 and 7, which can be deployed upon request. This compatibility encompasses "right-to-left" languages and all types of characters.

Just as background, Enable's multilingual database (.ENA file) is a relatively small flat file in binary Unicode format which contains the settings for the project and both the source and translated strings. It is not dependent on the version of PowerBuilder used.

How are strings extracted from the application?

GLDB: All localization tools extract strings from the application, but the quality is often variable. In PowerBuilder applications, strings are used to access/change DataWindow properties (Describe and Modify methods) and may contain SQL statements and DataWindow syntax. In addition, many strings are created dynamically, directly from the source code.

Enable responds to these complexities by using **Enable Extractor**, a rapid and accurate parser based on PB grammar that gives the developer considerable control over what to extract. In addition, Enable can also capture strings at runtime in order to cover any special situations not identified by analysis of the source code. Lastly, support for parametric phrases leaves the source code untouched while allowing translators to reposition the parameters as necessary within the strings concerned.

Does Enable support all languages?

GLDB: Enable supports an unlimited number of languages, as well as jargon variations of the official language, such as English for end users (which is frequently different to that used by developers).

How is an application translated?

GLDB: There are three ways to translate an application using Enable:

1. export/import (using Excel, XML or clipboard interfaces), ensuring that the work of multiple translators is synchronized and that translator memories can be used;
2. via the translation function within Enable Author (suitable for making minor changes)
3. working directly on the live application with Enable Runtime Explorer (discussed below), which can also be made available to translators and power users.

Some tools translate each control separately which, apart from the maintenance issues, can mean having to translate the same terms many times over. To avoid this, Enable translates phrase by phrase, taking account of exceptions in context.

So translation contexts and rules are covered too?

GLDB: Yes, indeed. Enable fully supports the recognition of contexts so that translations can be made more accurate, by adapting to the precise context in which the phrase occurs. Contexts represent one of the more important localization issues and, within Enable, the contexts recognized correspond to PowerBuilder objects: Windows, DataWindows, Menus and visual custom Classes.

Enable also supports rules for defining which elements should be translated, both at a general level and on a context-sensitive basis. These rules are defined in Enable's multilingual database and no changes to the source code are required.

Can you summarize how Enable is implemented?

GLDB: There are three steps to making a PowerBuilder application multilingual.

Step one is to add the Enable initialization code to the application, link it with the translation engine and then capture most of the source phrases using Enable Extractor, which I mentioned before. All this takes just a few minutes.

Step two represents the transformation cycle, during which the remaining source phrases are captured, exported for translation and re-imported. At this stage, **Enable Runtime Explorer** is activated to resize and move controls and DWOs, considering the revised space requirements of the translated text. Translations often need to be amended when seen in context on screen and Enable Runtime Explorer can do this on the fly!

Step three is about finalizing and deploying the new multilingual application. This involves dealing with special cases, such as calls to methods that change the appearance of windows after they have been opened or handling non-displayed elements. The Quick Start Guide explains how to proceed. Royalty-free deployment simply requires the addition of the translation engine's PBDs and DLL to the application, together with the multilingual database.

To close, can you tell us how long an Enable implementation takes?

GLDB: Ignoring translation time, which is beyond our control of course, we generally find that an average business application can be 90% "Enabled" in just a few hours. The remaining parts of the application are then covered by dealing systematically with the various exception conditions.

The time required for subsequent maintenance releases is negligible, as I mentioned before, since the application framework is already primed and new code therefore automatically takes account of the related Enabling requirements.

Lastly, remember that Enable Runtime Explorer enhances the efficiency of all development work, since it helps to bridge the gap with the runtime environment. By popular request, Enable Runtime Explorer is now available as a stand-alone tool.



© 2007
Enable Development LLP
All rights reserved.

PowerBuilder is a
registered trademark
of Sybase Inc.

SYBASE™

BUSINESS
SOLUTIONS
ALLIANCE

Contact us
for more information:
www.enable-pb.com
info@enable-pb.com