Polyglot Programming in Uniface

(Original creator: eknochs)

For a definition of Polyglot Programming I refer the reader to the 2006 definition from Neal Ford, which basically says that you should use the appropriate language to solve some specific problem, and that modern applications are becoming more complex, which in turn leads to the use of more than one language within that application. This helps avoid stretching the abilities of a single language to perform all tasks, at the expense of needing an additional skill (knowledge of other programming languages). In practice, most of us have learnt more than one programming language. The Polyglot Programming concept can also stretch to scripting languages like shell scripts. Thus we have probably been Polyglot Programming since well before 2006. Those applications which might be driven by DOS .bat (or Unix C-Shell) scripts to manipulate some simple files, build Uniface batch program command lines, execute them and then initiate some other batch programs, would fit the polyglot genre. Another type of legacy example exploits the Uniface URB call-in, call-out architecture through signature definitions. This extends Polyglot Programming across remote systems boundaries so that the choice of the other programming language can also be dictated by the technology of that remote platform (e.g. use C# on Windows platforms, and C++ on UNIX). Even though we’ve been comfortably Polyglot Programming for years, the Uniface mantra is to improve productivity by using a single development environment with high levels of abstraction in the programming, i.e. develop once and deploy anywhere.

This has meant that new Uniface functionality often removes the need for Polyglot Programming, e.g. better OS file handling proc commands, direct support for DBMS stored procedures (SSP signature implementation type) and so on. The idea is to bring back as much coding as possible into the UDE and Uniface repository, and so improve application maintainability. However, current and future application needs are steering us toward web 2.0 and mobile applications, which on the client side share a technology stack of HTML5 / CSS3 and JavaScript, delivered over HTTP. So we will see a swing back to Polyglot programming as JavaScript programming will be mixed in with Uniface programming. There is one difference this time. We are mixing the two languages inside the one UDE.

It’s true that Uniface 9 can’t syntax check your JavaScript, but there are definite advantages right now. Keeping all your application code in one repository, and subject to a uniform version control strategy, is going to make maintenance more efficient. Then there are all the usual code management facilities available for proc statements, like #include modules, pre-compiler directives to drive code generation, inheritance from the Application Model, and so on. It is also possible to share the JavaScript coding with Client Server and Windows GUI applications, with the availability of the HTML widget on Forms. Whilst it will continue to be possible to write functionally rich and complete applications in 100% Uniface, many of us will embrace the JavaScript API and develop polyglot systems in the future.