Rational. software

Exploiting Enterprise Generation Language for Business Oriented Developers

IBM Software Group

Contents

Abstract

The Java™ platform offers many attractive characteristics for building modern software systems. Programmers already experienced with object-oriented languages typically find Java relatively easy to learn and use. But developers familiar with procedural programming, 4th-generation languages (4GLs), and other traditional development technologies often find Java complex—so much so that they resist opportunities to use it. They instead continue developing with the programming technologies with which they are most comfortable.

Enterprise Generation Language (EGL) is specifically designed to help the traditional developer leverage all the benefits of the Java and COBOL platform, yet avoid learning all of its details. EGL is a simplified high-level programming language that lets you quickly write full-function applications based on Java and modern Web technologies. For example, developers write their business logic in EGL source code, and from there, the EGL tools generate Java or COBOL code, along with all the runtime artifacts you need to deploy the application to the desired execution platform.

EGL hides the details of the Java and COBOL platform and associated middleware programming mechanisms. This frees developers to focus on the business problem rather than on the underlying implementation technologies. Developers who have little or no experience with Java and Web technologies can use EGL to create enterprise-class applications quickly and easily.

This paper first provides some background into EGL's conceptual foundation. It then describes EGL at a high level, along with the motivations for employing such a language and its development environment. The paper goes on to present more details of EGL as it pertains to building applications. Finally, it provides some insight into the architecture behind an EGL-based application. After reading this article, you should have a good understanding of what EGL is, who would use it, and what value it brings to your users.

Conceptual foundation

Before delving into the details of EGL, let's review the conceptual foundation upon which the language was built. IBM® centers the vision for its application development tools on the themes of developer productivity and robust platform support. The vision has always been to provide an environment that enables developers to efficiently apply their business knowledge to creating applications that can operate across various execution platforms.

As far back as 1981, when IBM introduced a tool called Cross System Product, the core development tools product mission has been:

> ***To provide an integrated tools environment for the rapid development of scalable, robust, mission-critical applications using traditional enterprise skills to create applications capable of running under a variety of environments and topologies.***

Over the years, IBM products have evolved to continuously improve IBM's support of this mission. We've added many enhancements that improve development productivity In addition, we have incorporated new technologies that leverage advancements in the latest runtime platform support. Each improvement and advancement lets developers work on concerns farther away from implementation details and closer to the problem under consideration. This is the principle of abstraction at work.

Working at increasingly higher levels of abstraction helps achieve higher levels of productivity. Abstraction is also key for allowing developers to write code that can run on different target runtime platforms. The following list serves as basic guiding principles for how abstraction has helped each new product release further deliver our development tools mission:

- ***Language neutrality:*** *Factors other than what developers prefer or are most comfortable with often dictate which programming language they will use for an application. But what if we had a common language that was designed to generate applications into various other, more conventional languages? Such neutrality provides the developer with a common means of expressing application logic, which developers can later transform into the implementation language best suited for the selected target platform (e.g., COBOL or Java).*

- ***Platform neutrality:*** *As with language neutrality, platform neutrality lets you support the runtime platform best suited for the application. To effectively provide platform neutrality, you must support virtually any platform in the market–from the largest mainframe running to the smallest workstation or desktop PC. Abstraction provides a mechanism for developers to design and implement their applications with a language that is not tied to a specific technology. In doing so, you can generate the actual deployed application from this neutral development environment. As technology changes, the tooling vendor can provide drivers that transform the neutral application to the new target technology.*
- ***Code generation:*** *Code generation is the bridge between the abstract application written in a neutral language and a concrete implementation written in a conventional language. The generation technology also worries about how the concrete application gets deployed to a particular target runtime platform. Tooling vendors can provide generation drivers that automatically and transparently perform these transformations. These generators provide a high percentage of code that is associated with the application's structural "plumbing." Developers focus on business rules, which typically comprise a smaller percentage of the entire application code set. By separating business logic from infrastructural code, you can later cast the entire application into a new implementation technology by simply using a new set of code-generation drivers. This all results in a new realm of development productivity.*
- ***Debugging –*** *For a language like EGL to work in practice, the developer writing code at the abstract language level must be able to debug at that level. The tools environment should have a testing facility that includes a source-level debugger. The debugger permits stepping through the abstract program code using real data before the application is deployed into the target environment.*

EGL and Model Driven Architecture
Readers familiar with the Object Management Group (OMG) and its Model Driven Architecture (MDA) initiative will notice parallels to EGL. MDA is a form of model-driven development based on the Unified Modeling Language (UML) and other OMG standards. MDA calls for modeling the software lifecycle at distinct levels of abstraction, coupled with transformations that map and manage the relationships between those models.
MDA defines the notion of a Platform Independent Model (PIM) to which EGL matches quite nicely (albeit as a textual "model"). MDA also defines a Platform Specific Model (PSM), which corresponds to EGL-generated code (e.g., Java/J2EE or COBOL). The notion of MDA's model transformations is analogous to EGL's code generation.
These comparisons suggest that EGL can offer traditional developers an opportunity to practice what the MDA initiative is all about: separation of concerns, modularized reuse across the lifecycle, managing complexity, and the ultimate in productivity.

These guiding principles result in real, tangible benefits that can help increase the likelihood of a project's success. The benefits include:

- ***Less code to write:*** *Generating a large portion of the application code–particularly the infrastructural plumbing required as part of any target architecture, such as J2EE–shields developers from having to learn about or write special code for most of the application. The developer can instead focus on writing only the business rules.*
- ***Reduced training requirements:*** *Due to the time and cost required to train legacy developers, training proves to be a barrier for legacy developers moving into object-oriented programming and other new technologies. Code generation helps to reduce the cost and time needed to become proficient in designing and implementing applications.*
- ***Proxy to new technology:*** *As technology evolves (a change that we know is a constant), the training cost as well as the disruption caused by applying new technologies to applications can be very high. The neutral language application, combined with a code-generation driver for the new technology can help to make this transition much easier. In this way, you are keeping the application definition constant while leveraging improvements in implementation technology.*
- ***Improved quality and performance:*** *Code generation offers the benefit that it pre-tests a great portion of the generated code for quality and performance. The custom code that developers need to write for a given application includes only the code that defines the business rules. This can reduce bugs and improve quality and performance.*

What is EGL?

Any abstract programming language described in the previous section must meet the following goals:

- *Familiar to business-oriented developers*
- *Automatically manages lower-level programming details*
- *Transparently deploys to a set of potentially available execution platforms*

IBM's development tools previously included a high-level procedural language generally classified as a 4th Generation Language (4GL). Informix (now part of IBM) and many other companies also had their own 4GLs. All of these languages enabled business-oriented enterprise developers to design and implement applications without having to focus on the underlying technology. IBM's 4GL has evolved and has been renamed EGL, making it the modern version of IBM's 4th Generation Language legacy. Figure 1 below illustrates how EGL has evolved across several generations of IBM product technologies.

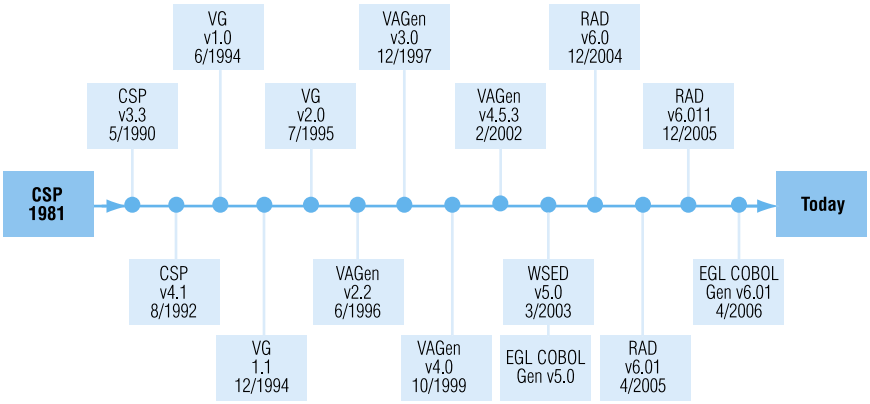


Figure 1. Evolution of EGL across its legacy

Today's EGL is the result of continuously enhancing the language with new constructs, integrations with new technologies, such as JavaServer™ Faces (JSF), and new code-generation drivers for the latest runtime platforms. EGL continues to provide developers with an unparalleled abstraction layer that enhances productivity and provides the conduit to multiple runtime platforms.

At the most basic level, EGL is a procedural programming language that enterprise-level or business-oriented developers can use to implement applications quickly. The word "generation" in the name implies two things:

- *Business logic written in EGL will be transformed into lower-level code.*
- *Runtime artifacts will be created to help execute the generated application on a desired target platform.*

EGL programs are written, tested, and debugged at the EGL source level, not on the generated code level. This means that you can defer actual code generation until you have satisfactorily tested the EGL application functionality. This aspect differentiates EGL from many other types of code generators. The EGL developer never changes the generated code—all changes are made at the EGL level.

In a broader and more comprehensive definition, EGL defines not only a language but a highly productive development environment. Integrated into several IBM Rational® products, EGL can help increase productivity not only with the language abstraction and simplicity but also with the integration of other key technologies, such as JSF and Eclipse.

Business value of EGL

EGL provides a simplified approach to application development based on these principles:

- ***Familiar programming model:*** *EGL provides an easy-to-learn programming paradigm embodied in a traditional procedural programming syntax that is familiar to business-oriented developers. The developer's view is abstracted to a level independent of the underlying implementation technology. It shields developers from the complexities of various supported runtime environments. This can result in reduced training costs and a great improvement in productivity.*

- ***Transparent code generation:*** *Developers write their business logic in EGL source code while the tools environment does the rest. These tools transform business logic into Java or COBOL, and optimize the infrastructure code for the target runtime platform. This results in less user-written code, which means faster turn-around time and fewer bugs in the deployed application. As an example, when generating Java code to run in the application server that will call a z/OS service, EGL automatically generates the Java classes necessary to invoke the associated CICS/COBOL program elements.*
- ***Runtime platform robustness:*** *Whenever a change to the target runtime platform occurs, only a new code-generation driver for the new platform is needed. This allows the application source code to remain constant while improvements in implementation technology are leveraged. For example, if a new Web services technology becomes available, you can reuse the same EGL source code—you only need to simply regenerate the application using the new driver.*
- ***End-to-end EGL-based debugging:*** *Source-level debugging is provided at the EGL level; therefore, you don't need to generate code before debugging it! This provides developers complete, end-to-end isolation from the complexity of the underlying implementation technology. Developers debug at the EGL level even if the application makes calls to other, non-EGL components. For example, if EGL calls a COBOL DB2-stored procedure that will execute in the z/OS, the EGL debugger works even when stepping into such components.*

Many companies are under pressure to quickly roll out new systems based on existing and emerging J2EE and Web Services standards; this is due to the obvious benefits of these technologies. While many available developers lack the technical skills needed in these areas, they are extremely valuable because of their expertise in the business domain, their understanding of business requirements, and their general experience in how to implement such systems. However, simply re-training this workforce in Java, J2EE, and related Web technologies is generally not practical or cost effective.

As a development environment, EGL can address many of the training and transition challenges. It allows you to leverage your current business-domain knowledgeable staff to use the latest technologies with minimal costs and effort. The result may allow your company to be more flexible and responsive to new business opportunities.

Who benefits from using EGL?

Simply put, anyone who needs to focus more on solving business problems and less on underlying implementation technologies can benefit from EGL. To be more specific, the list below features the most common types of business-oriented developers who benefit from using EGL:

- ***Informix 4GL Developers:*** *IBM has a special utility that helps migrate a large portion of your Informix 4GL-based applications to EGL, enabling you to begin working in a modern, extensible software development environment.*
- ***RPG Developers:*** *EGL offers a procedural language that is familiar to RPG developers. This will can enable developers to move to a modern platform while reaping the benefits of the latest technologies and still exploiting iSeries.*
- ***Visual Age Generator Developers:*** *EGL represents the next generation and logical migration path for these developers. The environment provides easy-to-use and highly automated migration capabilities that bring your valued Visual Age applications into a modern development environment—an environment in which the applications can leverage a modern set of runtime technologies*
- ***CA*** *Cool: Gen, CA Telon and CA Ideal Developers: EGL is particularly attractive for zSeries based systems, and an excellent replacement for orphaned CA tools. It allows zSeries to be exploited for runtime, but development can occur on a distributed platform, potentially reducing development costs.*

- ***Software AG Natural Developers:*** *Natural developers can quickly learn and become proficient in EGL whether they are Natural/ADABAS, Natural DB2 or Natural VSAM. EGL offers an Enterprise Modernization alternative by migrating Natural applications to a modern software development platform executing on zSeries, Unix or Windows.*
- ***Visual Basic Developers:*** *EGL offers similar but more powerful development efficiencies than Visual Basic, particularly in the areas of enterprise scalability and multi-platform runtime support.*
- ***COBOL/PLI Developers:*** *By generating COBOL from EGL, COBOL developers can move to a new platform that leverages the latest technologies. Moving to EGL within the IBM Rational development tools can free developers who have been trapped in legacy platforms and who can contribute greatly to new projects with their business domain expertise.*
- ***Database Developers:*** *EGL simplifies having to learn the database manipulation language and code the Create, Read, Update, Delete (CRUD) functionality by simply doing it for you.*
- ***Other 4GL Developers*** *(Oracle Forms, Natural, CA 4GL tools, etc.): A community of IBM Business Partners can help you transform your legacy 4GL applications to the IBM Rational development platform with EGL.*

EGL and the IBM Rational Software Development Platform

EGL is a key technology integrated into several IBM software development tools. The products below are organized in the Design and Construction discipline of the IBM Rational Software Development Platform. The choice of which product to use for EGL development is based on the role and scope of responsibilities that the user has within his or her project. Any of these products supports the full range of EGL core capabilities. The notes below explain some of the integrations and other facets of the lifecycle to consider in determining which offering is best for your needs.

- ***IBM Rational Application Developer for WebSphere® Software:*** *A comprehensive, integrated development environment for rapidly designing, developing, analyzing, testing, profiling, and deploying applications using Java, J2EE, Web, Web services, service-oriented architecture, and portal technologies. Rational Application Developer provides for EGL development. It supports EGL generation to Java and pre-existing J2EE components. This integration supports Web page development by integrating the EGL language with JSF controls. You can write controller logic in the page handlers associated with every JSP page. Business logic is written in EGL libraries and EGL programs. Rational Application Developer is our product for users who need to do either standalone EGL development or integrate EGL development with J2EE and portal development.*
- ***IBM WebSphere Developer Studio Client for iSeries, Advanced Edition:*** *An integrated development environment for developing Java, Web, Web services, and client/server applications specifically to run on the iSeries server. This product makes it easy to create, test, deploy, and maintain modern business applications requiring little Java, Web, or Web-service programming. This product includes EGL support for Java (or COBOL) generation.*
- ***IBM WebSphere Developer for Z:*** *An integrated development environment optimized for developing COBOL or PL/I native applications running on the z/OS. Developers can optionally create J2EE and Web applications integrated with legacy transactional environments (CICS and IBM IMSTM) and their associated languages (COBOL and PL/I). The product includes wizards that help developers write Web services against existing legacy COBOL CICS programs using either Java connectors or SOAP for CICS. This product also includes all the capabilities of Rational Application Developer and optional support for EGL generation to COBOL.*
- ***IBM Rational Software Architect:*** *A compressive design-and-construction tool that leverages model-driven development with UML, enabling users to create well architected applications and services. This product also includes all the capabilities of Rational Application Developer and offers UML modeling for users who want a model-driven approach to their EGL development.*

- ***IBM Rational COBOL Generation Extension for zSeries:*** *Is a pluggable extension for Rational Application Developer or Rational Software Architect that enables EGL to provide system options for IMS, ZOSBATCH and ZOSCICS COBOL generation.*
- ***IBM Rational COBOL Runtime for zSeries:*** *Provides the runtime libraries for programs that were developed with Enterprise Generation Language, a component of IBM Rational Application Developer and IBM Rational Software Architect, and the IBM Rational COBOL Generation Extension for zSeries and execute on z/OS. These libraries provide common runtime subroutines that are shared by all programs created with Enterprise Generation Language (EGL) and the IBM Rational COBOL Generation Extension for zSeries.*

Application development with EGL

The following sections describe EGL elements that are important to developing applications.

EGL language

The EGL language is a full-featured, procedural language that abstracts out the details of a target technology. EGL has verbs like "get," which simplify the programming model by providing a consistent specification to various target data sources. For example, a "get" statement can refer to records in a database or messages in a message queue. Developers are not required to learn and code technology-dependent database managers or message-oriented middleware programming.

Writing your applications in EGL can also protect your development investment. You can cast or generate the abstracted language into any other language. Currently, EGL can generate Java or COBOL. As technology changes and evolves, you protect your investment by having the ability to re-generate into a new, improved target platform or to entirely new platforms—without the need to modify your application.

EGL libraries

EGL has a construct called a library. An EGL library is simply a file that includes EGL code. EGL libraries allow application developers to easily decouple the business logic from other application code. EGL libraries provide various entry points—one per function. You can call these functions from other functions in other libraries or from EGL code in EGL programs or EGL page handlers.

The use of EGL libraries is optional, but it is the best way to reuse components. You can compare EGL libraries to COBOL subroutines or Java classes. Many EGL libraries are provided directly in the products with built-in functions. This is similar to Java classes provided by Java toolkits or frameworks. These libraries have the potential to greatly simplify and accelerate application development.

EGL programs and functions

Developers can also use EGL programs to code the business logic, but with a single entry point. EGL programs are similar to COBOL programs. An EGL program can be a main program, or can be called in the same way they are called in COBOL. EGL programs usually call EGL functions. You can compare an EGL function to a paragraph in the COBOL Procedure Division or to a Java method.

EGL page handler

When coding EGL applications to be deployed for the Web, the preferred method utilizes JSF. This framework has one or more Java Server Pages (JSP) screens. In an EGL-based application, every page will have a "shadow" page handler. The EGL page handler controls a user's runtime interaction with a Web page. Specifically, the page handler provides data and services to the page-displaying JSP. The page handler itself includes variables and the following kinds of logic:

- *An OnPageLoad function, which is invoked the first time the JSP renders the Web page*
- *A set of event handlers, each of which is invoked in response to a specific user action (specifically, by the user clicking a button or link)*

It is considered a best practice to not include any logic in the page handler. The page handler implements the controller component of the MVC pattern (discussed later). Although the handler might include lightweight data validations, such as range checks, you should invoke other programs or functions to perform complex business logic in order to follow MVC principles.

Database connectivity with EGL

Accessing data from databases can sometimes be challenging for developers who primarily want to provide users with the information to make the best business decisions.

To access data, a developer needs to:

- *Connect to a database*
- *Know and use the database schema*
- *Be proficient in SQL in order to get the appropriate data*
- *Provide the primitive functions to perform the basic CRUD database tasks*
- *Provide a test environment to efficiently test your application*

EGL provides capabilities that make this task easy for that business-oriented developer:

- ***Connectivity:** Wizards will take these developers through a step-by-step process of defining connectivity. You could locate the databases in remote locations, such as z/OS systems.*
- ***Database schema:** If you are using an already existing database, EGL provides an easy-to-use import capability that will make the schema structure available to your application.*
- ***SQL coding:** EGL provides the generation of SQL statements based on your EGL code. You can then use the generated SQL or alter it to suit your needs.*

- ***Primitive functions:*** *The EGL generation engine will automatically generate the typical CRUD functions that are the workhorse functions for database-driven applications.*
- ***Test capabilities:*** *The IBM development tools have a test environment that helps eliminate the complexities associated with deploying and running your application in complex target platforms.*

File access with EGL

You can also use EGL programs and libraries to access other data storage mechanisms, such as serial files, indexed and relative record files (VSAM), MQ Series queues, and other system files. This provides a large amount of flexibility in the types of data sources that you can use within an EGL application system.

Application architecture with EGL

To complete your application, you need to bring the above elements together in the context of an application architecture. When you complete this integration, you'll have a concrete, deployable application. The architecture that EGL generates is the J2EE architecture. Other important technologies you need to understand include Java Server Faces (JSF) and the Model View Controller (MVC) pattern.

JSF and EGL

JSF is a set of Java classes and JSP tag libraries that provide a framework for developing Web applications. Its implementation in Rational Web Developer and Rational Application Developer lets you drag and drop JSF controls onto a page canvas instead of having to implement pages using hand-coding techniques.

The integration of EGL and JSF produces an event-driven model in which a page-specific handler manages each request. The page handler can act on information submitted with the request, or it can forward the information to another handler for processing. This event-driven model greatly simplifies the building of Web applications. Control logic in the page handler is written in EGL. Business logic in the libraries and programs is also written in EGL.

This means that you don't need Java skills to write your application's user interface or business logic. The EGL code will generate all the Java code. The JSF with EGL duo makes for an extremely high productivity page development environment.

Model View Controller

The Model View Controller framework (referred to as MVC or "model 2") has many benefits and is often considered a best practice for developing Web applications.

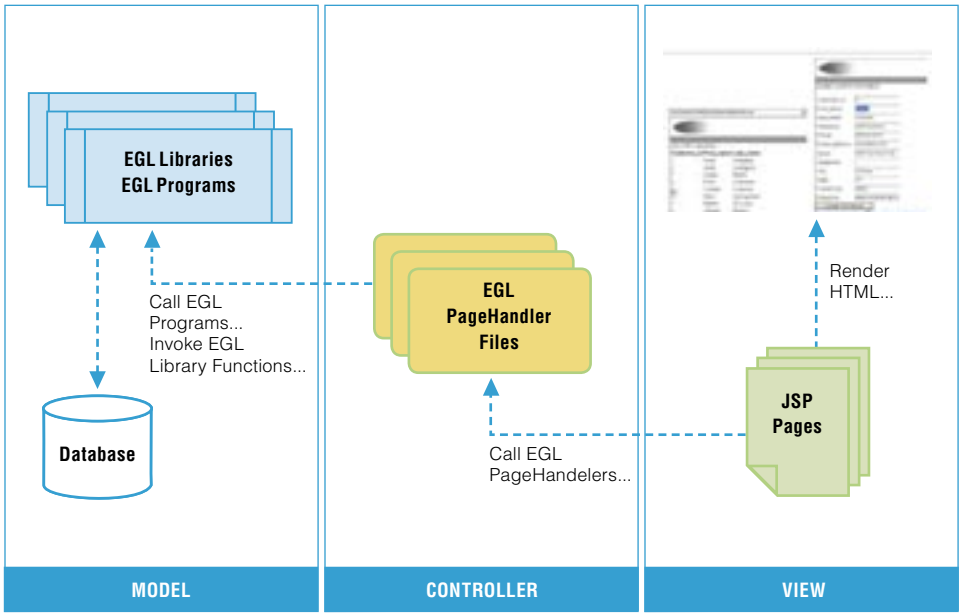
At runtime, the application server contains both the view and controller components of an MVC Web application, while a third tier (which can be inside or outside the application server) contains the model.

Model: You can find the business logic, which in most cases involves accessing data stores, such as relational databases, in the EGL libraries and programs.

View: The code responsible for the presentation layer consists of JSPs and Java beans that store data for the JSPs to use. Page creation is greatly simplified by using JSF controls available within the Rational Developer products' page editor.

Controller: The EGL page handlers contain the code that determines the overall flow of events.

Figure 2. Relationships between elements of an EGL application.



The beauty of the EGL development environment is that business-oriented application developers are not confronted with and do not need to understand how to implement the MVC pattern. The generation engine does that for them.

Text user interface support

In some cases, you will require existing text-based user interfaces. EGL provides an Eclipse-based WYSIWYG editor in order to construct text user interfaces (TUI, a.k.a. "green screens") for EGL. This allows EGL developers to define EGL TUIs that deploy as 5250 or 3270 applications running in iSeries or zSeries environments, respectively. You can also run these EGL TUIs on distributed Java platforms as required.

Walking through an EGL application scenario

Figure 3 illustrates an application scenario using EGL. The following steps will show the flow of information and tasks associated with a simple development scenario in the context of an EGL application.

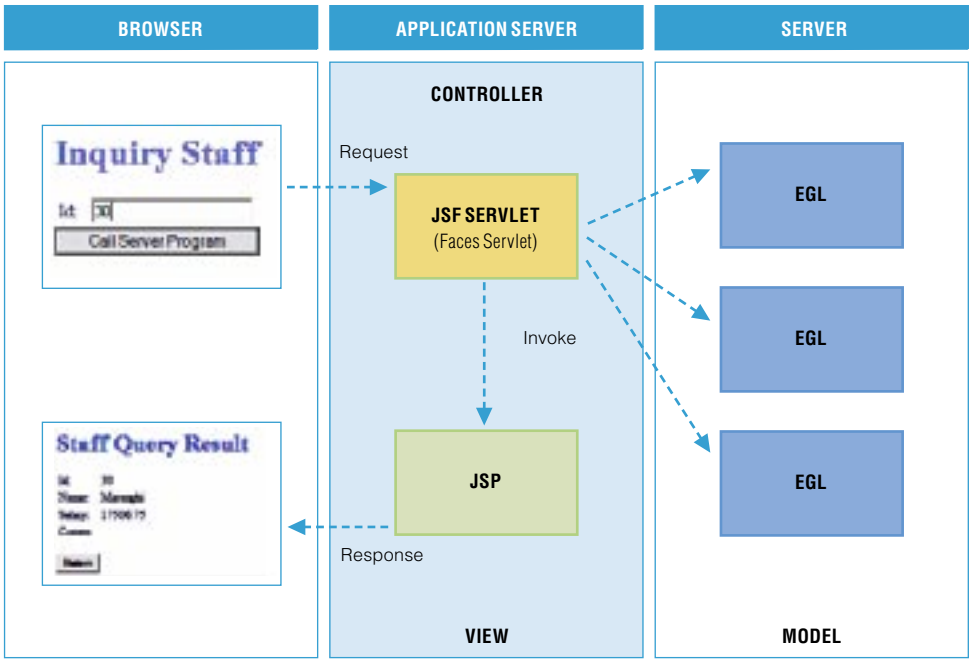


Figure 3. Typical EGL application scenario.

In this application:

1. The user types an ID and clicks a button.

2. Clicking the button creates a request that the JSF Servlet handles. The controller servlet in turn invokes the appropriate server program. The ID then passes to the server program.

3. The server program validates the ID by reading a DB2 database using the ID as the key to find. The server program can be a function within an EGL library or an EGL program.

4. If the server program finds the ID, it collects information (name, salary, and commission) and returns the ID.

5. The server sends the returned data to the Result JSP page, which displays it.

6. If server program does not find the ID, it creates an error message and returns the ID.

7. The server sends the error message to the Error JSP page, which displays it.

To accomplish this simple application, the developer will create the three pages using the drag-and-drop page editor shown in Figure 3. The controller logic that calls the appropriate server function is written as an EGL page handler and the server function that performs the validation is also written in EGL as an EGL library function. The EGL development environment will generate Java and pull all these pieces together into a J2EE-based application that you can deploy and run.

You can find details of this example by visiting http://www-106.ibm.com/developerworks/websphere/library/techarticles/0408_barosa/0408_barosa.html.

You can see with this simple walkthrough example that the complexity behind the new and evolving Web technologies is hidden from the developer by an easy-to-use page editor and an easy-to-use programming language called EGL.

COBOL Generation and deployment scenario

The following steps will show the flow of the generation and deployment tasks.

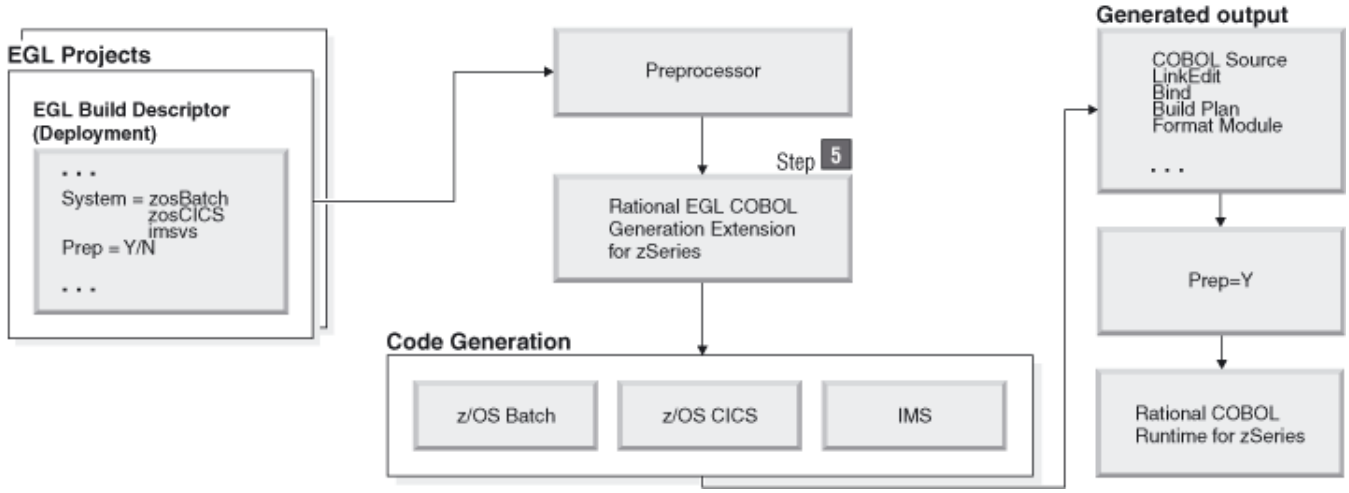


Figure 4. Illustrates an application generation and deployment scenario using EGL for COBOL.

In this application:

1. An EGL developer, working from their Windows or Linux workstation, creates an EGL project(s) with the required EGL components.
2. A Build Descriptor is created (or previously created) to specify the type of COBOL generation to be performed and the specific parameters to use (e.g. system=ZOSCICS, Prep=Y, high-level data set qualifiers etc.)
3. The Generate or Generate with Wizard option is invoked via <cntl> G or via a right mouse click on the object
4. The generation may require a USERID and Password for mainframe access(this can also be imbedded in the BuildDescriptor)
5. Generation invokes the Build Server and performs the COBOL generation using the EGL code and Build Descriptor parameters. The Generation process will communicate back to the workstation upon completion (success or failure) or optionally write result to a data set
6. The COBOL code, LinkEDT, Bind (for DB2) etc. are created on an interim Directory
7. A BuildPlan XML file is created detailing what PROCs to run and what input files from the interim Directory are to be used
8. Load modules are then uploaded to the specified mainframe

Migrations and Enterprise Modernization

This white paper does not intend to address the various driving forces behind Enterprise IT Modernization of legacy applications or the various planning steps involved. However, what is of importance to note in the context of this paper is the role EGL can play in such modernizations.

The new generation of IT systems requires software development capabilities that support new middleware and emerging application architectures. Flexibility is the name of the game. However because of the variety and complexity of activities and artifacts necessary to implement these solutions, it is essential that development organizations establish a systematic and comprehensive approach to developing software with proven methodologies, processes and tools. At the same time, legacy application developers are asking IBM to preserve the levels of productivity and simplicity that they have become accustomed to and which support business-critical applications.

Enter EGL. It is a component of a broad, comprehensive Application Development, Governance and Life Cycle Management solution set from IBM Rational which spans modeling and application development through testing and software configuration management tools. It is highly productive language which provides a familiar development model that business-oriented developers are accustomed to. There is no need to re-staff, but instead rewire and leverage your existing staff with EGL. And knowing how crucial your investment in existing legacy systems is, EGL can call your business-critical applications, leveraging what is best about your existing legacy environment. Or if and when appropriate, you may choose to migrate your existing applications to EGL.

Through code migrations of VAGen, I4GL, Natural, RPG, CA Cool:Gen, CA Telon, CA Ideal, CA Synon and other 4GLs applications, IT can leverage their existing investment and move rapidly onto a modern software development platform. This is made possible through automated conversion utilities designed to migrate the specific legacy environment to EGL in a cost-effective, efficient manner.

Once migrated, you then have the opportunity to step back and analyze if a redesign is required, particularly from a User Interface (UI) perspective. If a redesign is required, EGL can assist you in bringing your applications to the Web or with leveraging a Rich Client Platform (RCP).

To summarize, EGL can play a critical role in Enterprise Application Modernization helping to future-proof IT application organizations from the ever changing world of middleware, databases, languages, platforms and computing environments.

Conclusions

Our purpose here was to introduce Enterprise Generation Language: What it is, why IBM has invested in its development, and how many types of traditional software developers benefit from it. Java, J2EE, and all the modern Web technologies are powerful, yet their complexities make them sometimes challenging to learn. IBM is making advancements in all its product lines to make Java and Web development easier. IBM also understands the importance of leveraging COBOL, as needed, for business-critical applications. EGL is one important technology enabling us to address these critical application development issues.

We hope that this paper has you thinking about the possibilities of using EGL to speed up the adoption of emerging Web technologies, improve productivity, leverage legacy developers, and increase your likelihood of success in building applications. If you feel that EGL can benefit you or your organization, contact your local IBM sales team or visit our Web site at http://www-128.ibm.com/developerworks/rational/products/egl/ to learn more about EGL.

Acknowledgements

This paper is a composition drawn from several documents produced for IBM developerWorks®/Rational by the following EGL subject matter experts: Raul Ortega, Steve Choquette, Joe Pesot, Stephen Hancock, Cliff Meyers, Tim Wilson, Larry England, Rusty Edmister, Stefano Sergi, Daphne Green, Todd Britton, Mark Evans, Jon Sayles, and Reginaldo Barosa and Ed Gondek.

IBM Corporation
Software Group
Route 100
Somers, NY 10589

Produced in the United States of America
04-06

G507-0995-00