EGL For Enterprise Developers

Transforming the Enterprise Using EGL/RAD Technology

IBM-Rational RAD Tools White Paper Series: Volume 1, Number 1
Abstract
This white paper describes the use of WebSphere Studio Enterprise Developer version 5.1.2 and more specifically Enterprise Generation Language (EGL), IBM’s strategic 4GL for rapid application development, and Java Server Faces (JSF) an industry standard, component-based method of web page development for transforming zSeries developers into productive and successful Internet application developers – or into more productive mainframe programmers.

The first section of this paper introduces WebSphere Studio Enterprise Developer (WSED), Java Server Faces (JSF) and Enterprise Generation Language (EGL) from a business-oriented, platform, project, design and development perspective. This is a “view from 10,000 feet” – and will provide background and understanding of the technologies from the broader perspective of contemporary application development, challenges and solutions. Section two of this document describes the step-by-step process of creating a web application page using a combination of EGL, JSF and WebSphere Studio Enterprise Developer.

It should be noted here that, this white paper focuses primarily on an introduction to EGL in the context of building web applications. However, EGL (as part of WebSphere Studio Enterprise Developer) can be used to create z/OS C.I.C.S. and batch applications, including full transaction processing systems. A dedicated and in-depth treatise on this topic is planned for an upcoming white paper.

Table of Contents…

Abstract ........................................................................................................................................................................................2
Section 1 - The Business Intelligence Imperative ....................................................................................................................................3
Here Comes That Next Platform ..........................................................................................................................................................4
What is EGL? ....................................................................................................................................................................................5
Consistent, Productive Programming Model ...............................................................................................................................5
Flexible Programming Model ......................................................................................................................................................5
Leading-Edge Programming Model ........................................................................................................................................5
That Can’t Be All There Is To It! ...................................................................................................................................................6
Un-level the Playing Field – Putting it all together using EGL, JSF and WSED ........................................................................7
Web Page Design Using EGL and Java Server Faces ..................................................................................................................7
Java Server Faces (JSF Components) ........................................................................................................................................8
WebSphere Studio Enterprise Developer ....................................................................................................................................9
The WSED IDE ................................................................................................................................................................................10
Section 2 - Developing Web Applications With EGL, WSED and JSF ..........................................................................................11
The Software Development Process (Pocket Edition) ..................................................................................................................11
The Software Development Process – Using EGL, JSF, WSED .................................................................................................11
Model Your Application’s Data ..................................................................................................................................................13
Develop the Application Functionality ........................................................................................................................................14
Data Access With EGL ..............................................................................................................................................................15
Writing and Testing Business Logic With EGL ........................................................................................................................16
Page Development Using JSF/JSP ................................................................................................................................................17
Generate, and Deploy Your Application .......................................................................................................................................18
Section 3 - Summary and Conclusions ..........................................................................................................................................20
Appendix 1 – EGL QuickStart Application Architecture – Java/Web ...........................................................................................21
Appendix Two – EGL z/OS Similairties and Differences ...............................................................................................................22
Appendix 3 - EGL Productivity Metrics ........................................................................................................................................23

Transforming The Enterprise Using EGL/RAD
Section 1 - The Business Intelligence Imperative

Those of us working in business software development today are aware of the split in development teams between software professionals who have accumulated years (often decades) of business-processing knowledge and others who are primarily oriented towards accumulating technical knowledge and following leading-edge development trends. Business knowledge professionals understand the complex and intimate implementation details of a software application’s rules and semantics – from the deep perspective of operational corporate computing (the engine that runs an enterprise from the data center). Technical software developers understand how to implement new systems using leading-edge technologies. Business knowledge professionals understand what the applications that run your business do.

This split is not new, nor is it absolute, as there is typically overlap, sharing, and cooperation between both camps. However this dichotomy is an important facet of resource utilization, and managing teams successfully on large, complex contemporary software projects.

For the sake of argument, let us cast legacy and z/OS developers in with business knowledge professionals. As far back as the 1970’s, legacy programmers have been developing, maintaining and enhancing the large, mission-critical mainframe applications that – to this day – support the vast majority of corporate transaction processing, production data updates, and business reporting across the Global 2000. These business knowledge professionals – for all intents and purposes own business processing intelligence, in most software shops, at least insofar as understanding the core business rules and semantics embedded in the code. This is a fact all too frequently lost on those who stand to gain by focusing on, and promoting “what’s new” at the expense of “what works”, “what operates in the data center every night” and “what makes – as opposed to spends money” for an organization.

Business knowledge professionals are as technical as any other software professionals, as evidenced by the fact that they have developed, enhanced and continue to support some of – if not the most complex logic, algorithms and technology in operation at any software site. In fact, it is just this wealth of both business knowledge and mainframe-technology expertise that is at the heart of this white paper. Because to “transform” any enterprise will require the deep and ongoing involvement of business knowledge professionals, who bring their accumulated decades of understanding how to analyze, design and implement complex business rules and semantics to the table – in each new project and into the next platform introduced.
Here Comes That Next Platform

So what is the next leading-edge platform? There are so many to choose from, but clearly a case could be made for the J2EE/Java computing platform; a deep and complex blend of languages and technologies that includes: Enterprise Java Beans, Web Services, the J2EE API, XML messaging, MVC/Struts, JSF, and much more. If as a legacy business professional you are not deeply familiar with these terms you are in good company. Many who have been doing 24X7 support and work over-time, week-ends and holidays on mainframe mission critical systems have not been free to spend the protracted length of time (twelve months, or more, by many estimates – hardly a cost-effective proposal) it takes to gear up for production-level work, natively coding to the language APIs and technology specifications of the J2EE platform.

So the business intelligence imperative becomes – How can we integrate and leverage the critical required knowledge base of business professional software developers in leading edge application development work? Especially given the extraordinary learning curve associated with these new platforms, and the accelerated development schedules common in today’s enterprise software environment? The answer – in a word: EGL.
**What is EGL?**

EGL is IBM’s strategic 4GL for rapid application development of:

- **CICS COBOL transactions** – running under z/OS
- **Batch COBOL applications** – running under z/OS
- **Java/Web Applications** – running under WebSphere on a multitude of platforms

As a 4GL (from IBM) EGL does not represent the quantum leap associated with learning J2EE/Java-related technologies (J2EE, WML, XML, etc.). EGL is based on a simple, transparent 4GL programming model, one that deals with concepts that are familiar to z/OS professionals: Records instead of objects, encapsulation and inheritance, Keywords instead of APIs, and function calls instead of polymorphism. A large part of EGL’s strategic value to organizations in fact is its ease of learning and its simple programming model. As long as you have programmed in COBOL, RPG, Assembler, PL/I or any Client/Server language for some time, you will find EGL easy to learn – even to the demanding level of programming production-quality large, complex business rules and logic.

However, if ease-of-use were the only standard of measurement applied there would be many choices for leading edge development. The truth is that leading-edge application functionality requires a procedural language that supports both a deep, consistent and productive vocabulary, for specifying production-level business logic, coupled with keywords that provide a broad and elastic set of interfaces to leading-edge technologies.

**Consistent, Productive Programming Model**

For example, using EGL you issue a “get” statement to access a relational database record – or to access the entire result set from an SQL SELECT that returns multiple rows into an array. You use the same (get) keyword to return a WebSphereMQ (middle-ware) message, or to access a COBOL sequential or indexed file. Based on the application context and data definition in which it is coded, an EGL “get” statement constructs into the equivalent native coding (Java or COBOL) routine to access the data. This could be anywhere from 1 to 100 lines of source. Consider the development, testing and maintenance benefits of writing a one statement ... versus dozens of lines of intricate, even convoluted Java/JDBC code.

**Flexible Programming Model**

But, you might ask “what if I need additional flexibility – perhaps even total control over the entire database access routine?” A fair question, and one that EGL answers by offering you multiple levels of specification flexibility. Let’s take as an example, an SQL cursor processing routine – consisting of Declare/Open/Fetch-until/Close secondary routines. Using EGL you can use the implicit SQL statement: get – as discussed above, with default conditions to return data with minimal coding effort. At the next level down, you can over-ride the default WHERE clause of an EGL get statement with a keyword-modifier “defaultSelectCondition” – which allows you to specify custom data access filter logic. At a third level, during development you can use the EGL get statement to generate an initial, syntactically correct select/from/into statement, and further customize any part of it – allowing the EGL compiler to process the result rows with framework code. And finally, for situations where you need complete control over the entire statement, you can code all elements of the SQL Select routine – (declare, open, fetch, close). One language, one toolset, one coding paradigm...total, unrestricted programming flexibility.
**Leading-Edge Programming Model**

But there is much more to today’s Internet application development requirements than procedural logic and database access. And the EGL programming model accommodates contemporary Internet development requirements, insulating you from having to learn (and develop in) multiple native APIs. Let’s take an example of your program conversing directly with the WebSphere application server. In this capacity, an Internet you write might have to do two things:

1. Access the variable data sent to and received from a web page - known as HTTP request variables)
2. Access application server variables used to define and maintain the state of your transaction – known as HTTP session variables.

1. Accessing request variable data using EGL is simply a no-brainer. During development you bind (assign) a control field on a web page with an EGL variable. This binding process automatically forces the EGL compiler to create “presentation-logic” in the Java generation of your program – that sends and receives web page data for your program automatically. Needless to say, these routines are non-trivial and time-consuming to hand-code. Having EGL generate them allows you more time to focus on procedural business logic, rather than dealing with low-level API calls, Java casting (redefine) operations and what not.

2. To access application server (or “session variables” – which are variables that persist data for as long as a user is logged on to their session with your application over the internet) you issue an EGL: `getSessionAttr` call. To update or insert session data in EGL you code: `putSessionAttr`. Not rocket science.

We go into this level of detail here only to demonstrate that EGL insulates developers from the complex and often vendor-specific APIs native to the coding paradigms of J2EE, Web Services, synchronous and asynchronous messaging, etc. It generalizes and raises your technical coding effort to a higher level of abstraction. Which brings benefits to the table such as:

- Increased productivity
- Fewer code defects and lowered support and enhancement costs
- Accessibility to and by the audience of legacy business developers

**That Can't Be All There Is To It!**

If you are wondering; “is EGL some sort of: panacea-language” that solves the challenge of leading edge application development all by itself, you’d be right to think not. Like virtually everything in the software world that solves complex demanding real-world problems, there are many parts to a bona fide solution. And in the EGL world, those parts include the following:

- **EGL** – for specifying business logic. Think of EGL as the equivalent (or perhaps successor) to COBOL.
- **Java Server Faces (JSF)** – for developing a state-of-the-art User Interface. JSF is the equivalent of BMS (for you C.I.C.S. programmers) or MFS (for you IMS TM folks). Except that comparing JSF/EGL rapid development using state-of-the-art visual tools to BMS or MFS macro hand-coding is like comparing state of the art word processing software running on a computer to a 1960’s manual typewriter (that is, it’s simply not a fair fight).
EGL For Enterprise Developers

- **WebSphere Studio Enterprise Developer** – for organizing your projects. WebSphere Studio Enterprise Developer is the packaged product offering from IBM that contains the software bits for EGL, JSF and includes a test version of **WebSphere Application Server** which is used running and debugging your EGL/JSF code. You can consider WebSphere Studio analogous to TSO/ISPF – for project management and job submission. Note that by “projects” here, we refer to an application as a project – not project as a task.

### Un-level the Playing Field – Putting it all together using EGL, JSF and WSED

Software market analysts consider each of the above technologies “industry-leading” – in that they are recommended “best practice” solutions in their respective software category. By investing substantially in EGL, JSF and WSED internal research and development, and by integrating and delivering them in one state-of-the-art packaged product IBM has created an “un-level playing field” when compared with any combination of tools from other vendors.

For example, to provide the equivalent development functionality of EGL, JSF and WebSphere Studio together, you would have to use J2EE/Java (from Sun Microsystems), purchase an IDE to develop the J2EE code, purchase a 3rd party library of JSF controls (unless you wished to code at the native J2EE API level) and purchase an application server for testing and deployment – assuming you wish to develop and test in a true J2EE/EJB platform. None of these tools and languages supplied by myriad vendors would be integrated. Each would require separate training, installation, configuration and technical support.

Contrast this with WebSphere Studio Enterprise Developer – which contains EGL, JSF (including powerful IBM graphical component enhancements) and WebSphere Application Server in the box. Not only are all the tools organized, viewed and used under one consistent programming shell (eclipse) but the implementation integrates and leverages each tool’s unique advantages and features – a situation where return-on-investment equates to: \(1 + 1 + 1 = 5\) (possibly more).

We’ve been discussing EGL exclusively up to this point; so let’s focus next on Java Server Faces technology so that you can understand its importance and the role it plays in doing leading edge development work technology. We will end this section with a short tour of WebSphere Studio Enterprise Developer.

### Web Page Design Using EGL and Java Server Faces

If you have done any kind of online programming using C.I.C.S. or IMS/TM (D.C.) you know that an online screen:

- Consists of text labels and data fields
- Responds to Enter, Clear, PA (Program Attention), and PF (Program Function) keys
- Interacts with your application program (COBOL, PL/I, etc.) by sending or receiving a formatted data structure, to and from CICS or IMS, via a BMS or MFS control block – which is defined to handle the fields (formatting, etc.) in a macro language
- Additionally, your program may be coded to handle different Enter, PA, or PF keys pressed by the user

Similarly, in an Internet web application, a web page:

- Consists of text label controls, data field controls, Command controls and tags
- Command controls are **bound to** (assigned to) individual named EGL functions (COBOL paragraphs) inside your program. The functions are automatically invoked, when the user
clicks the Command control with their mouse (or presses a keystroke shortcut combination defined for that control)

- Interacts with your application program (EGL) via records and individual data fields bound to JSF controls
  - Note: To bind a record or field to a control (or to bind a Command button to a function) you simply drag and drop the field onto an area inside of WebSphere Studio called the Content Area – during page design.

So, while the names and some of the development activities change, there are many more parallels and similarities than differences – and little that is conceptually new in the EGL Web page development process – although there is certainly a new – and better – way to do page development, using graphical tools and pre-built components through WebSphere Studio Page Designer and JSF.

**Java Server Faces (JSF Components)**

JSF is a non-proprietary (industry open standard) specification for designing web page controls (text fields, labels, Command buttons, combo-boxes, radio buttons, etc.). IBM has implemented the JSF standard in WebSphere Studio, providing page developers with a rich palette of such controls (see Figure 1), and making the rendering of sophisticated user interface designs a drag and drop development process requiring little to no coding or scripting in J2EE page-development languages such as JSP (Java Server Page) coding.

It should be noted that the power of JSF lies in the constructed code, that, behind the scenes works in concert with EGL to send and receive data dynamically to web pages, and during development provides access to a set of properties for visual customization, as well as high-level data validation rules and an event/response model that allows you to code for all of the standard Windows/Browser events. As before, non-trivial hand-coding involving deep Java API calls is simplified in the extreme using WSED and JSF drag and drop development.

Web page developers can also take advantage of style templates (page frameworks with pre-assigned graphics and text), which enforce consistency and raise productivity) and of course EGL interacts with JSF to send and receive data automatically. IBM provides a robust set of these (available to use as-is, or as the basis for you to create your own custom designs).
WebSphere Studio Enterprise Developer

EGL and JSF technologies are actually shipped with, and integrated into IBM’s award-winning tool-suite known collectively as: **WebSphere Studio Enterprise Developer**.  WebSphere Studio Enterprise Developer – pictured in figure 2 below – contains features and facilities for:

- **Creating and managing your applications** – including interfacing with source code management systems like ClearCase – this would be similar to ISPF/PDF and code library management system on the mainframe for developing COBOL, JCL, etc.

- **Establishing personal development preferences** – similar to **ISPF option 0** where you specify over-rides to the base TSO product installation defaults.  Except that with WSED you establish and customize your own combination of preferences, views, editors, build (compile), run-time options and more.

- **Running and testing your application** – Similar to a combination of TESTCOB/COBTEST, **ISPF Option 6**, Clists and **REXX execs**, and **IBM Debug Tool**.  WSED contains an integrated run-time and test environment using the WebSphere Application Server, that allow you to see both your executing application AND the source code line-by-line in debug mode.
The WSED IDE

As shown in figure 2 below, WSED is a state-of-the-art integrated development environment that organizes your projects and project artifacts (such as source code, web pages, libraries, folders and sub-folders, etc.), and organizes the tools you would use to work on your project for you.

WSED presents your project and application elements through a customize-able set of **views**, which allow you access to a graphical set of properties for something within **context**. **Context** here refers to something you are currently working on, as shown below in figure 2, the attributes of a table grid control that you’ve clicked on. There are also native coding-level views, for modifying most source elements (See the **Source** tab, for the page in the Content Area below). This dual-mode approach offers the best of both the rapid application development world and the ultimate flexibility and control of hand coding for your application business and technical requirements.

So now that we’ve opened the hood and inspected the individual parts (EGL, JSF and WSED), let’s take this approach to web development for a test drive – and look more closely at the process of creating and testing web pages and the logic that drives them.
Section 2 - Developing Web Applications With EGL, WSED and JSF

The Software Development Process (Pocket Edition)
While every shop – and often divisions within shops – will have their own unique process for creating business software, let us assume that majority of the application development efforts you participate in adhere to the following lifecycle (we’re assuming here that the business-side of the equation is complete, and that requirements are known, published, and signed-off):

1. **Analyze and design the solution** - based on the business requirements (both data and logic) – create (or use existing) database tables, and record definitions
2. **Develop** – write the COBOL code, the screens/maps, test harness scripts, JCL, etc.
3. **Compile the artifacts** – COBOL to load modules, Gen the BMS maps, MFS blocks, DBD/PSB/ACBGEN, etc.
4. **Debug** – test your work
5. **Deploy** – promote the code to the production libraries for release

I realize this is a reduction and simplification of project lifecycles you may have participated in (especially larger ones), but for the sake of comparing what’s new versus what’s the same across these platforms let’s accept that generally this is a representation of what we do in building software applications.

The Software Development Process – Using EGL, JSF, WSED
Even though the tools and languages will change (for the better, once you’ve adapted to the new platform) the same process you’ve used as described above are still the order of business, in creating leading-edge web applications, at least when using WSED, EGL and JSF. Do not be concerned that the names of the steps in the process are different, read the details in the list below and you will see that the decades spent refining a best-practice approach to software implementation and the hard-core lessons-learned are not being thrown out and replaced with new-age, untried/untested approaches.

1. **Model (Analyze and Design the solution).** Just as on the mainframe, in this first phase, from the business requirements you would analyze and design your software solution. When designing data source(s), you might consider using one of the software modeling packages from IBM/Rational (IBM Rational Software Architect, IBM Rational Software Modeler, IBM Rational Rose Data Modeler, etc.) to analyze your business/data application requirements, and specify your application’s classes – which become relational tables and EGL Records and data definitions. Transforming relational tables to EGL records can be accomplished one of several ways:
   a. Using an EGL utility package (UML2EGL) you can import the model directly into your EGL libraries and source files for immediate development use.
   b. If you have existing databases EGL supports a process of importing database schema (table, columns and key definitions) directly into an EGL library.
   c. Of course you can always code or EGL record and data definitions from-scratch, but in the world of rapid application development that is usually considered a last resort.
2. **Develop (code/create the program and page source).** Using WSED tools and visual development facilities such as Page Designer – to create sophisticated web pages using drag and drop development techniques, Content Assist (an intelligent and extensible source editor)
and **Code Snippets** (a means of establishing libraries of parameter-driven reusable source code) you write or **create**:

a. **EGL programs** - business logic for back-end processing. **New Concept:** You might consider thinking of EGL programs as software “services” – as opposed to simply modules, in that they contain functionality that transforms (sends and receives data), and provides this service to some calling or requesting program(s)

b. **JSP pages** – Using Page Designer, you drag and drop JSF components (recall that these become web page controls) onto the Content Area of WebSphere Studio. You assign and customize the JSF component properties and attributes.

c. **EGL Page/transaction control logic.** Using the intelligent editing facilities of WebSphere Studio, you code/create logic that controls the flow of pages through a transaction and fills JSF data components with backend data values.

3. **Generate (construct and compile the programs into Java or COBOL).** After you have created an EGL program, or Web Page and EGL control logic you would generate the Java or COBOL that will be tested and run in WebSphere (or if you are generating COBOL, tested and run on a zSeries machine. Note that Generate would seem to be the only step in this process different from the mainframe application development steps, however, looks can be deceiving. In COBOL, the Compile process first generates Assembly language statements from your COBOL, then creates the object decks and load modules from the Assembly source. This is little different from EGL-Generate, where statements – written in this higher-level 4GL are constructed into Java or COBOL – before being compiled into executable and test-ready code. In this analogy think of Java and COBOL as sort of modern-day Assembly languages, and EGL as the language for specifying business logic at a higher level of abstraction.

4. **Debug (test your solution)** – Using the WebSphere Application Server that comes with WebSphere Studio, you would run your web pages in a test environment to evaluate the functionality and user interface design work, or step through your business logic line-by-line in debug mode.

5. **Deploy and Run** – in this final step, you package up the Java beans into WAR/EAR files, or compile and link the COBOL modules into a production load library for inclusion into your production system.

What is important to understand from the above is that, the entire process is consistent with mainframe best practice solutions, rapid/productive, uses integrated state-of-the-art development tools and technologies, and yet does not require that developers who use them learn or program in Java/J2EE. Let’s take a short pictorial junket through these steps. It should help to see what it’s like to use WSED, EGL and JSF to build Internet software.
Model Your Application’s Data

As mentioned, there are two options for modeling your data for an EGL application: 1. Translate a UML model, and 2. Import database schema directly from the metadata in the DBMS catalog, through either the Data Parts Import wizard, or by coding EGL import statements.

To translate an existing Rational Rose Class model (Class diagram) into database schema statements (DDL) and EGL Services (libraries with records and function calls) you use a utility program known as the UML2EGL Translator (shown below in figure 3);

Figure 3 – UML2EGL Translator – used to generate database schema and EGL code
Alternatively you could Import the EGL statements (specifically Record and DataItem parts) from relational tables using a Wizard driven process – that even allows you to prototype C.R.U.D. and administrative pages.

Note that this EGL wizard is available for version 6.x and later of WebSphere Studio.

**Figure 4 – Import Wizard** - used to generate EGL Records and Data Definitions

---

**Develop the Application Functionality**

Using the imported data models and WSED tools (such as Content Assist, Code Snippets, etc.) you write or create EGL business logic for back-end processing and page/transaction control. You use and highly visual and declarative development facilities to create JSP pages for dynamic web content. You test and debug interactively using WebSphere Application Server. If you are doing mainframe development (EGL COBOL Generation) you test and run the generated code on the host.

So this is basically an iterative development process consisting of application, prototyping, creating, testing and refining three elements:

1. Data access logic
2. Business logic and debugging
3. User interface (page) design work and logic

The following are a number of annotated screen captures which depict development activity in these three areas.
Data Access With EGL

Using simple EGL language abstractions (Get, Delete, Add, Replace keywords), you can create sophisticated and syntactically correct SQL data access statements easily and quickly. Recall from the beginning of this white paper that there are multiple levels of code automation. Shown below (figure 5) is the most automated (RAD) approach – of coding an EGL get statement, and then seeing the resultant SQL code generated for it. Recall that there are various ways of enhancing what is generated, over-riding defaults and if necessary hand code any and all elements of the data access routine.

Figure 5 – Data Access Logic in EGL – a get statement, and the SQL it generates by default
**Writing and Testing Business Logic With EGL**

EGL is both a rich 4GL – with powerful constructs that scale to the complex requirements of production business logic, and a simple language to develop in – without the inherent complexities and lengthy learning curve associated with O-O technologies like J2EE.

Additionally the combination of EGL and the facilities in the box with WSED allow you to leverage IBM’s considerable investment in advanced IDE software. Shown below in figure 6 is an EGL program call statement, run in debug mode (shown line-by-line) at the EGL source-language level. Note that you can set breakpoints; view variables see an Outline of your application and have multiple windows open for the Browser (to view your web page results) and the source code itself.

*Figure 6 – Business Logic - Debugging EGL statements using EGL/WSED debugging facility*
**Page Development Using JSF/JSP**

EGL Internet applications may either use Java/Struts technology to create web pages, or JSF components. Either way, developers again are leveraging IBM’s extended investment in technology beyond the EGL language, and benefiting from it.

Below (figure 7) is a page being developed with JSF components, which are dragged-and dropped into the Canvas Area – and then using declarative (RAD) development refined by setting properties, as opposed to manually coding in the respective JSF API, JSP and J2EE languages.

---

*Figure 7 – JSP Page development – Drag & drop declarative development process*
Generate, and Deploy Your Application

EGL generates either COBOL programs (figure 8) or Java programs (figure 9) for the run-time platform(s) specified in your project. In fact it is a simple, one-step operation to generate in either COBOL or Java – allowing you to write business rules (services, as described earlier) in EGL, and generate them to COBOL for running on the host, then later generate the same EGL to Java – if/when it comes time to run certain business functions from an application server like WebSphere.

![Figure 8 – COBOL Generated from the EGL statements – note that this program would be uploaded to the host and run on the iSeries or zSeries – in batch or under C.I.C.S.](image-url)
Figure 9 – Java Program Generation – the same EGL Library was used to generate Java classes or COBOL programs. By rebuilding the Project you generate Java or COBOL code, which can be deployed.
Section 3 - Summary and Conclusions

EGL provides a 4GL-programming model for building leading edge Internet applications and generating COBOL business logic that is deep yet easy to learn. Using the combination of EGL for application logic, JSF for user interface and page design and WebSphere Studio Enterprise Developer for project infrastructure and testing, mainframe programmers can become successful and productive contributors to current web development projects, bringing with them their decades of business intelligence as well as technical skills and the knowledge of the existing code-base in production in your corporate data center. Including highly experienced, business-intelligence professionals is a key element in projects that successfully navigate the complex and challenging waters of hooking web applications up with mission-critical applications and data, as well as re-engineering production business rules into a distributed, web-based architecture.

EGL/JSF/WebSphere provides a unique, proven approach to:

- Skills transformation and the adoption of new technologies
- Architectural design and flexibility

By architecture design and flexibility, we refer to creating a Service Oriented Architecture using EGL.

Service Oriented Architecture

Service Oriented Architecture is fundamentally, a collection of services that communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. On the mainframe, inter-services communication has been achieved via COBOL CALL/CALLING conventions. However, in contemporary Internet application development, new technologies and platforms are being used to link applications, including: Messaging (synchronous and asynchronous), Web Services and distributed component-based business rules and processing logic.

Through its productive, flexible and leading-edge programming model, EGL simplifies the process of building all these types of services:

- Creating traditional call/calling programs using the higher level of specification abstraction, available in the 4GL and WebSphere Studio tools
- Calling EGL and non-EGL load modules on a standalone or in a distributed systems architecture – which requires only simple changes to build file parameters
- Calling Messaging platforms like WebSphereMQ by simply coding the EGL add and get verbs
- Publishing and consuming (create and call) Web Services in the same easy, syntactical manner (note that Web Service calling in native EGL will be available by mid-2005)

So, whether you are building on to traditional zSeries applications, coupling them with the Internet, or creating leading edge web applications from scratch – take a closer look at EGL, before assuming there is no other way to enable mainframe, business knowledge professionals other than to drop them off at some Java boot-camp for a month, and wait a year to see if they become productive members of the O-O/Java society. Why not let them learn EGL and become productive in a little over a week?

To learn more about EGL, visit the EGL Zone at:
Appendix 1 – EGL QuickStart Application Architecture – Java/Web

An EGL application follows the traditional MVC design pattern approach by separating different aspects of application functionality. This (MVC design pattern) is encouraged by the nature of development within the EGL/WebSphere Studio Application Developer platform.

Technical notes:

- **JSP Pages and PageHandlers**
  - Each .JSP page will have a one-to-one correspondence with a pagehandler.egl file of the same name. The JSP page will be generated to standard Java .JSP file. The pagehandler.egl file is generated to a Java program (a Java bean class ... not an EJB).

- **EGL Libraries**
  - Each EGL Library file is generated to a Java program. Individual functions coded within the library may be called directly from EGL statements inside PageHandlers.
  - The EGL Library file must reside on the same machine (local) as the EGL pagehandler that calls its functions.

- **EGL Programs**
  - Each EGL Program file is generated to a: Java program, Java wrapper, or EJB (Session Bean). You pass parameters to an EGL program from any calling source (there is a single point of entry).
  - The EGL program can be located and called either locally or remotely. It is important to point out that EGL programs can also be generated to iSeries and z/OS environments in COBOL.

Comment: It is important to note that EGL also generates COBOL programs that run under iSeries/zOS and other legacy environments.

Comment: The model-view-controller (MVC) design pattern separates the parts of an application into the model (the model is the core of the application’s business and technical functionality), the controller (the controller manages the execution flow of the application) and the view (the view presents, gathers, and submits information).
Appendix Two – EGL z/OS Similarities and Differences

We end this white paper with a chart that clearly shows the similarities between mainframe and EGL development constructs.

<table>
<thead>
<tr>
<th>Mainframe COBOL/PL1</th>
<th>I-Net/EGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Dictionary</td>
<td>EGL DataItem declaration</td>
</tr>
<tr>
<td>Program field or variable</td>
<td>EGL primitive or data/record declaration</td>
</tr>
<tr>
<td>Program paragraph</td>
<td>EGL function</td>
</tr>
<tr>
<td>Program (source)</td>
<td>EGL Program (.eg1 file)</td>
</tr>
<tr>
<td>Program (Load Module)</td>
<td>Compiled Java . or . COBOL executable</td>
</tr>
<tr>
<td>//Steplib</td>
<td>CLASSPATH=... (environment variable)</td>
</tr>
<tr>
<td>Program passed/returned parameters</td>
<td>Function passed/returned parameters</td>
</tr>
<tr>
<td>JCL</td>
<td>.cmd/ shell script (UNIX/Linux)</td>
</tr>
<tr>
<td>Typecast/REDEFINES clause</td>
<td>EGL REDEFINES</td>
</tr>
<tr>
<td>Procedure/Program/sub-routine calls</td>
<td>EGL function or program calls</td>
</tr>
<tr>
<td>Call to System/Utility programs</td>
<td>EGL Built-in Functions</td>
</tr>
<tr>
<td>COBOL Identification/Environment divisions</td>
<td>EGL program identifiers and Linkage part properties</td>
</tr>
<tr>
<td>COBOL Data Division</td>
<td>EGL Page Data and Record/DataItem and Primitive Data Declarations</td>
</tr>
<tr>
<td>COBOL Procedure Division</td>
<td>EGL functions</td>
</tr>
<tr>
<td>COBOL (Working-Storage) Table</td>
<td>EGL DataTable</td>
</tr>
<tr>
<td>Database/Table</td>
<td>RDBMS table</td>
</tr>
<tr>
<td>SQL statement/Stored Procedure</td>
<td>SQL statement/Stored Procedure</td>
</tr>
<tr>
<td>CICS, IMS TM, z/OS Batch execution</td>
<td>WebSphere Application Server</td>
</tr>
<tr>
<td>BMS Map <em>and/or</em> IMS MFS</td>
<td>JSP/JSF Page</td>
</tr>
<tr>
<td>CICS Queue/IMS TM Scratch Pad</td>
<td>EGL Session variable</td>
</tr>
<tr>
<td>CICS (BMS) In/Out map fields <em>or</em> IMS MFS MID/MOD</td>
<td>EGL Request/Response variables</td>
</tr>
<tr>
<td>TSO/ISPF</td>
<td>WebSphere Studio Enterprise Developer</td>
</tr>
</tbody>
</table>

*Figure 24 – Equivalences in Mainframe Concepts and Technology with EGL/Internet Terms and Concepts*
Appendix 3 - EGL Productivity Metrics

As demonstrated in an internal IBM study, where a control group created the Sun PetStore application from scratch, using a leading-edge, competitive IDE, then a development group in IBM re-created the exact same functionality using WSED/JSF/EGL – the IBM tooling provides a rapid, productive, and iterative development approach to building enterprise IT applications. This is also illustrated by a productivity study in which the J2EE industry-standard, SUN pet store application was utilized.

Using traditional methods of IDE development with savvy J2EE Java developers, you could hand-code the pet store application by writing more than 10,000 lines of code in about 500 hours or so. Or you may leverage your IT development staff by using the visual modeling and automated construction capabilities native to EGL. You visually design the pet store application; specify a little more than 600 lines of business logic and a few mouse clicks to get the same high quality, high performance application in approx 55 hours. Therefore, you focus on the business aspect of your application, not the infrastructure or platform plumbing (highly technical, complex and verbose J2EE code).

What is unique about the application on the right, created with EGL, is it is completely agile. It can be automatically re-constructed and targeted to a new combination of deployment technologies (for app server, databases, messaging transport), and using all new construction patterns and partitioning. And all these technology decisions and changes can be made AFTER the application is created functionally. The hand-coded application on the left is static; hardwired to specific deployment technologies, design, coding and construction patterns.