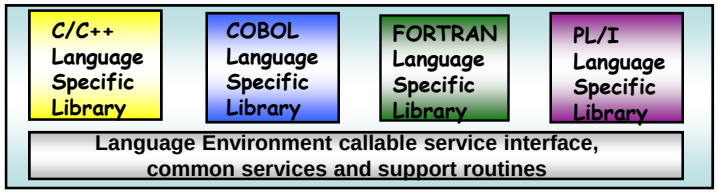
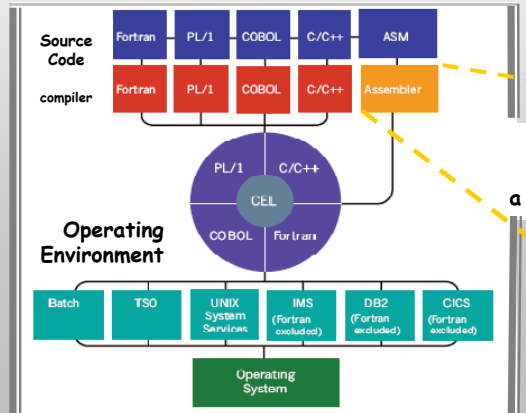
# CheatSheet #37 zTidBits zOS' Language Environment (LE)

- Today, enterprises need efficient, consistent, and less complex ways to develop quality applications and to maintain their existing inventory of application code.
  - The trend in application development is to modularize and share code, and to develop applications on a workstation-based front end.
- Language Environment (LE) provides customers a common environment for all LE-conforming high-level language (HLL) products.
  - An HLL is a programming language above the level of assembler language and below that of program generators and query languages.
  - In the past, programming languages also have had limited ability to call each other and behave consistently across different operating systems.
  - This had constrained those who wanted to use several languages in an application.
  - Programming languages have had different rules for implementing data structures and condition handling, and for interfacing with system services and library routines.
- Language Environment establishes a common run-time environment for all participating HLLs.
  - It combines essential run-time services, such as routines for run-time message handling, condition handling, and storage management.
  - All of these services are available through a set of interfaces that are consistent across programming languages.
  - Customers may either call these interfaces themselves, or use language-specific services that call the interfaces.
  - With Language Environment, customers use one run-time environment for their applications, regardless of the application's programming language or system resource needs.
- **Language Environment consists of:**
  - ***Basic routines*** that support starting and stopping programs, allocating storage, communicating with programs written in different languages, and indicating and handling conditions.
  - ***Common library services***, such as math services and date and time services, that are commonly needed by programs running on the system. These functions are supported through a library of callable services.
  - ***Language-specific portions of the run-time library.***
    > Because many language-specific routines call Language Environment services, behavior is consistent across languages.




**NOTE1:** Here displays the scope of compilers even though not all listed are currently supported, although Language Environment supports the compiled objects that they created.

- ✓ z/OS XL C/C++
- ✓ OS/390 C/C++
- ✓ C/C++ Compiler for MVS/ESA
- ✓ AD/Cycle C/370 Compiler
- ✓ VisualAge for Java, EE
- ✓ Enterprise COBOL for z/OS
- ✓ Enterprise COBOL OS/390
- ✓ COBOL for OS/390 & VM
- ✓ COBOL for MVS & VM (formerly COBOL/370)
- ✓ Enterprise PL/I for z/OS
- ✓ Enterprise PL/I for z/OS and OS/390
- ✓ VisualAge PL/I for OS/390
- ✓ PL/I for MVS & VM
- ✓ AD/Cycle PL/I for MVS & VM
- ✓ VS FORTRAN and FORTRAN IV (compatibility mode)




**NOTE2:** You might be surprised that ASSEMBLER is not listed, but there is a set of assembler macros provided to allow assembler routines to run with LE.

**Assembler does not require a RUNTIME Library**

- The z/OS Operating System has trimodal addressing: 24bits / 31bits / 64bits
  - Language Environment can exploit 64-bit addressing for applications written in C, C++, or *Language Environment-conforming* Assembler.
  - In the 64-bit addressing mode (AMODE 64) supported by Language Environment, addresses are 64 bits in length allowing access to virtual storage up to 16 exabytes.

☺ **What customers can do with Language Environment** ☺

- Language Environment helps them create **mixed-language applications** and gives them a consistent method of accessing common, frequently used services.
  - Building mixed-language applications is easier with Language Environment-conforming routines because LE establishes a consistent environment for all languages running within the application.
  - **Common use of system resources gives them greater control** provides the base for future IBM language library enhancements in the z/OS environment.
  - Many system dependencies are removed from LE-conforming language products.
  - Because Language Environment provides a common library, with services that customers can call through a common callable interface, the behavior of their application will be easier to predict.
  - Language Environment's common library includes common services such as messages, date and time functions, math functions, application utilities, system services, and subsystem support.
  - The language-specific portions of LE provides language interfaces and specific services that are supported for each individual language.
- LE provides consistent condition handling **simplifying error recovery**
  - Language Environment establishes consistent condition handling for HLLs, debug tools and assembler language routines.
    > For languages with little or no condition handling function, like COBOL, LE provides a user-controlled method that was not available before robust error recovery.
    >Language Environment condition handling honors single- and mixed-language semantics and is integrated with message handling services to provide customers with specific information about each condition.
- Language Environment **protects customer programming Investment**
  - Language Environment provides compatible support for existing HLL applications.
  - Applications linked with the migration tools provided with libraries that predate LE do not need to be linked with the LE library routines.
    > For mixed-language applications, however, relinking with LE may be required if the application was not previously relinked using migration tools available with pre-Language Environment libraries.
  - Routines compiled with the new LE-conforming compilers can be mixed with old routines in an application.
    > Thus, applications can be enhanced or maintained selectively, without recompiling the whole application when a change is made to a single routine.
- Inter Language Communication (ILC) offers **greater efficiency and flexibility**
  - Language Environment eliminates incompatibilities among language-specific run-time environments.
  - Routines call one another within one common run-time environment, eliminating the need for initialization and termination of a language-specific run-time environment with each call.
    > This makes interlanguage communication (ILC) in mixed-language appls easier, more efficient, and more consistent.
  - This ILC capability also means that customers can share and reuse code easily.
    > They can write a service routine in the language of their choice—C/C++, COBOL, PL/I, or assembler—and allow that routine to be called from C/C++, COBOL, PL/I, or assembler applications.
    > Similarly, vendors can write one application package in the language of their choice, and allow the application package to be called from C/C++, PL/I, and assembler routines or from Fortran or COBOL programs.
- BASICALLY, Language Environment lets customers **use the best language for a task**.
- Language Environment provides **a common dump for all conforming languages**.
  - The dump includes, in an easy-to-read format, a description of any relevant conditions and information on error location, variables, and storage.
- POSIX-Conforming Application support **enhanced code portability**
  - The IEEE *Portable Operating System Interface* (POSIX) standard is a series of industry standards for code and user interface portability.
  - POSIX support allows applications written for a UNIX-like operating system to be run on z/OS.
  - C language programmers can access operating system services through a set of standard language bindings.
  - C language programmers who uses z/OS UNIX System Services (z/OS UNIX) and z/OS Language Environment can call C language functions defined in the POSIX standard from their C applications, (***see #08 zTidBits zOS + UNIX = Winner).***
- Locale callable services enhances the **development of internationalized applications** increasing global markets for software products.