.NET BASICS ASSIGNMENT

- Submitted by (Kavita Goodwani INT 438)

1. Demonstrate the process of conversion of Source code into the native machine code in .Net framework with the help of a flowchart.

Answer:

The Code Execution Process involves the following two stages:

1. Compiler time process.
2. Runtime Process

Compiler time process

1. The .Net framework has one or more language compliers, such as Visual Basic, C#, Visual C++, JScript, or one of many third-party compilers .
2. Any one of the compilers translate your source code into Microsoft Intermediate Language (MSIL) code.
3. For example, if you are using the C# programming language to develop an application, when you compile the application, the C# language compiler will convert your source code into Microsoft Intermediate Language (MSIL) code.
4. In short, VB.NET, C# and other language compilers generate MSIL code. (In other words, compiling translates your source code into MSIL and generates the required metadata.)
5. Currently "Microsoft Intermediate Language" (MSIL) code is also known as "Intermediate Language" (IL) Code **or** "Common Intermediate Language" (CIL) Code.

SOURCE CODE -----.NET COMLIPER------> BYTE CODE (MSIL + META DATA)

2. Runtime process.

1. The Common Language Runtime (CLR) includes a JIT compiler for converting MSIL to native code.
2. The JIT Compiler in CLR converts the MSIL code into native machine code that is then executed by the OS.
3. During the runtime of a program the "Just in Time" (JIT) compiler of the Common Language Runtime (CLR) uses the Metadata and converts Microsoft Intermediate Language (MSIL) into native code.

BYTE CODE (MSIL + META DATA) ----- Just-In-Time (JIT) compiler------> NATIVE CODE

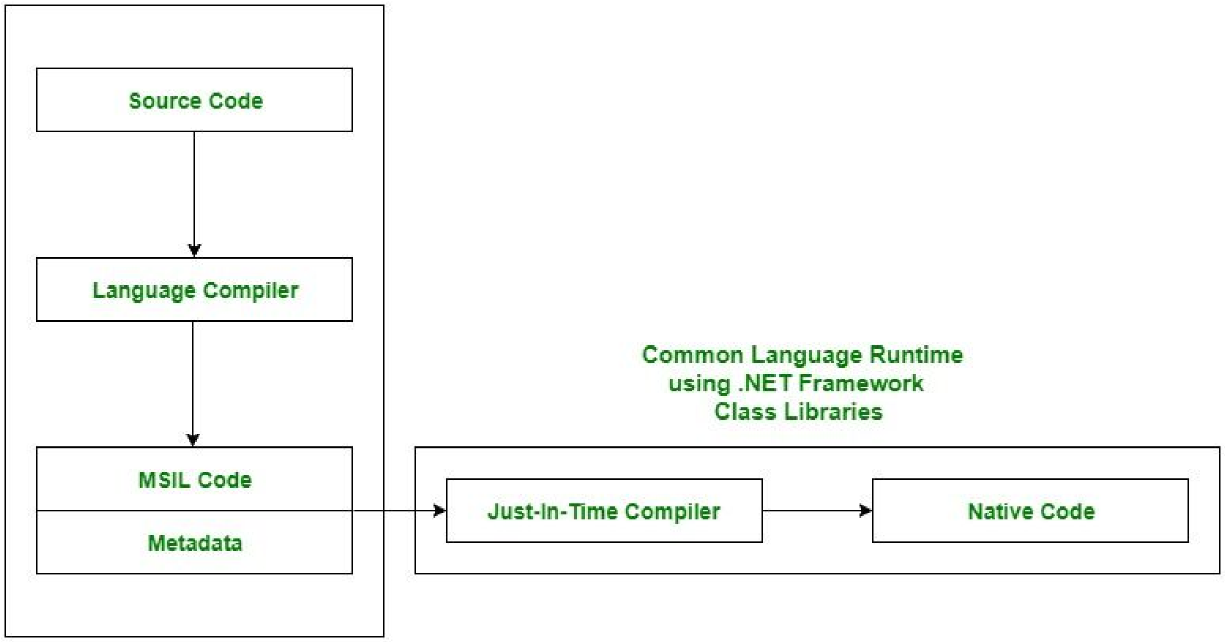

Image Reference : Geeks for Geeks

2. Explain CTS and how the .net framework implements CTS.

Answer:

CTS describes a set of data types that can be used in different .Net languages in common. (i.e), CTS ensures that objects written in different .Net languages can interact with each other.

For Communicating between programs written in any .NET complaint language, the types have to be compatible on the basic level.

The common type system defines how types are declared, used, and managed in the common language runtime, and is also an important part of the runtime's support for cross-language integration. The common type system performs the following functions:

- Establishes a framework that helps enable cross-language integration, type safety, and high-performance code execution.
- Provides an object-oriented model that supports the complete implementation of many programming languages.
- Defines rules that languages must follow, which helps ensure that objects written in different languages can interact with each other.

- Provides a library that contains the primitive data types used in application development.

The common type system supports two general categories of types:

Value types:

Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

Reference types:

Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and delegates.

3. Name at least 3 runtime services provided by CLR and explain their role in .net framework

Common Language Runtime (CLR) manages the execution of .NET programs. The just-in-time compiler converts the compiled code into machine instructions. This is what the computer executes.

The services provided by CLR include memory management, exception handling, type safety.

1. Memory Management

The Garbage Collector (GC) is the part of the .NET Framework that allocates and releases memory for your .NET applications. The Common Language Runtime (CLR) manages allocation and deallocation of a managed object in memory. C# programmers never do this directly, there is no delete keyword in the C# language. It relies on the garbage collector. The .NET objects are allocated to a region of memory termed the managed heap. They will be automatically destroyed by the garbage collector. Heap allocation only occurs when you are creating instances of classes. It eliminates the need for the programmer to manually delete objects that are no longer required for program execution. This reuse of memory helps reduce the amount of total memory that a program needs to run. Objects are allocated in the heap continuously, one after another. It is a very fast process, since it is just adding a value to a pointer. The process of releasing memory is called garbage collection. It releases only objects that are no longer being used in the application. A root is a storage location containing a reference to an object on the managed heap. The runtime will check objects on the managed heap to determine whether they are still reachable (in other words, rooted) by the application. The CLR builds an object graph, that represents each reachable object on the heap. Object graphs are used to document all reachable objects.

2. Exception Handling

An exception is a problem that arises during the execution of a program. An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero. In the .Net Framework, exceptions are represented by classes. The exception classes in .Net Framework are mainly directly or indirectly derived from the System.Exception class. Some of the exception classes derived from the System.Exception class are the System.ApplicationException and System.SystemException classes. The System.ApplicationException class supports exceptions generated by application programs. So, the exceptions defined by the programmers should derive from this class. The System.SystemException class is the base class for all predefined system exception. .Net provides a structured solution to the exception handling problems in the form of try and catch blocks. Using these blocks the core program statements are separated from the error-handling statements. These error handling blocks are implemented using the Try, Catch and Finally keywords.

3. Type Safety

Type safety in .NET has been introduced to prevent the objects of one type from peeking into the memory assigned for the other object. Writing safe code also means to prevent data loss during conversion of one type to another. Type-safe code accesses only the memory locations it is authorized to access. For example, type-safe code cannot directly read values from another object's private fields or code areas. It accesses types only in well-defined, allowable ways, thereby preventing overrun security breaches. Type safety helps isolate objects from each other and therefore helps protect them from inadvertent or malicious corruption. It also provides assurance that security restrictions on code can be reliably enforced. Type-safe code accesses only the memory locations it is authorized to access. (type-safety specifically refers to memory type safety.) For example, type-safe code cannot read values from another object's private fields. It accesses types only in well-defined, allowable ways. Type safety means that the compiler will validate types while compiling, and throw an error if you try to assign the wrong type to a variable.

4. What are the differences between Library vs DLL vs .Exe? Explain.

Answer
DLL

- A Dll is a Dynamic Link Library can not run itself, used as a supportive file to other application.
- The Library Functions are Linked to the Application at Run Time (Dynamically) So the name is Dll.

- A Dll does not contain an entry point (main function) so can not run individually.

EXE

- An Exe is executable file and is not a supportive file rather itself an application.
- An Exe will contain an entry point (main function) so runs individually.

1	Can not run Individually	Runs Individually
2	Used as supportive file for other Application	Itself an Application
3	Does not contain an entry point (no main function) so can not run individually	Contains an entry point (Main function) so can run individually
4	A Program /Application with out main creates a DLL after compilation.	A Program /Application With main creates an EXE after compilation.
5	OS does not create a separate process for any DLL rather DLL will run in the same process created for an EXE	OS Creates a separate process for each EXE it executes.

5. How does CLR in .net ensure security and type safety? Explain.

Answer
Ensuring Type Safety by CLR

Type safety in .NET has been introduced to prevent the objects of one type from peeking into the memory assigned for the other object. Writing safe code also means to prevent data loss during conversion of one type to another. Type-safe code accesses only the memory locations it is authorized to access.

For example, type-safe code cannot directly read values from another object's private fields or code areas. It accesses types only in well-defined, allowable ways, thereby preventing overrun security breaches.

Type safety helps isolate objects from each other and therefore helps protect them from inadvertent or malicious corruption. It also provides assurance that security restrictions on code can be reliably enforced. Type-safe code accesses only the memory locations it is authorized to access. (type-safety specifically refers to memory type safety.)

For example, type-safe code cannot read values from another object's private fields. It accesses types only in well-defined, allowable ways. Type safety means that the compiler will validate types while compiling, and throw an error if you try to assign the wrong type to a variable.

In .Net type safety check happens both at Compile time and at runtime. The compiler will do the compile-time type safety check and CLR will do the runtime safety check.

At compile time, we get an error when a type instance is being assigned to an incompatible type; hence preventing an error at runtime. So, at compilation time itself, developers come to know such errors and code will be modified to correct the mistake.

Run time type safety ensures we don't get strange memory exceptions and inconsistent behaviour in the application. Type safety is provided by using the Common Type System (CTS) and the Common Language Specification (CLS) that are provided in the CLR to verify the types that are used in an application.

Type safety in .NET had been introduced to prevent the objects of one type from peeking into the memory assigned for the other object.