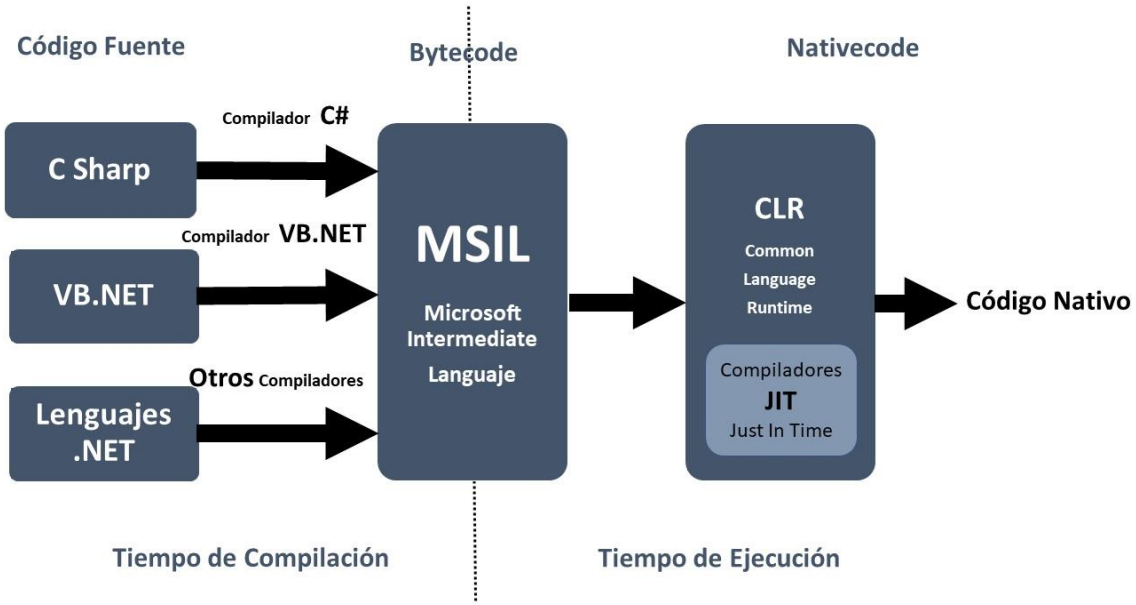
Introducción a .Net Framework

.Net Framework es una plataforma que proporciona las herramientas y tecnologías que necesita para crear aplicaciones en red, así como servicios web distribuidos y aplicaciones web.

.Net Framework proporciona el tiempo de compilación y la base de tiempo de ejecución necesarios para compilar y ejecutar cualquier lenguaje de programación que cumpla con la Especificación de lenguaje común (CLS). Los dos componentes principales de .Net Framework son el Common Language Runtime (CLR) y la Biblioteca de clases .Net Framework (FCL).

El **Common Language Runtime (CLR)** es un entorno de ejecución. Funciona como una capa entre los sistemas operativos y las aplicaciones escritas en lenguajes .Net que cumplen con la especificación de lenguaje común (CLS). La función principal de Common Language Runtime es convertir el código administrado en código nativo y luego ejecutar el programa. El Código Administrado solamente se compila cuando es necesario, es decir, convierte las instrucciones apropiadas cuando se llama a cada función. La compilación Just In Time (JIT) de Common Language Runtime convierte el Intermediate Language (MSIL) en código nativo a solicitud del tiempo de ejecución de la aplicación.



Durante la ejecución del programa, el Common Language Runtime administra la memoria, la ejecución de subprocesos, la recolección de basura, el manejo de excepciones, el **sistema de tipos comunes (CTS)**, las verificaciones de seguridad de códigos y otros servicios del sistema. El Common Language Runtime define el sistema de tipo común (CTS), que es un sistema de tipo estándar utilizado por todos los

lenguajes .Net. Eso significa que todos los lenguajes de programación .NET utilizan la misma representación para los tipos de datos comunes, por lo que Common Language Runtime es un entorno de tiempo de ejecución independiente del lenguaje de desarrollo. Durante la ejecución de un programa el Common Language Runtime también controla la interacción con el sistema operativo.

La biblioteca de clases de .Net Framework (FCL) proporciona la funcionalidad principal de la arquitectura de .Net Framework. La biblioteca de clases de .Net Framework incluye una gran colección de clases reutilizables, interfaces y tipos de valor que agilizan y optimizan el proceso de desarrollo y brindan acceso a la funcionalidad del sistema.

La biblioteca de clases de .Net Framework se organiza en una estructura de árbol jerárquica y se divide en **Espacios de Nombres (namespaces)**. Los espacios de nombres son una agrupación lógica de tipos con el propósito de identificación. La biblioteca de clases de Framework proporciona los tipos base que se utilizan en todos los lenguajes de desarrollo habilitados para .NET. Se accede a las Clases mediante los espacios de nombres, que residen dentro de los Ensamblados. El espacio de nombres System es la raíz de los tipos en .NET Framework.

Las clases de la biblioteca de clases de .Net Framework son clases administradas que proporcionan acceso a los Servicios del sistema. Las clases de la biblioteca de clases de .Net Framework están orientadas a objetos y son fáciles de usar en desarrollos de programas.

El **Common Language Specification (CLS)** es un conjunto de características básicas que los lenguajes .Net necesitan para desarrollar aplicaciones y servicios que sean compatibles con .Net Framework. Cuando hay una situación para comunicar objetos escritos en diferentes lenguajes de desarrollo de .Net, esos objetos deben exponer las características que son comunes a todos los lenguajes. La especificación del lenguaje común (CLS) garantiza una interoperabilidad completa entre las aplicaciones, independientemente del lenguaje utilizado para crear la aplicación.

La especificación del lenguaje común define un subconjunto de tipos comunes del sistema. El **Common Type System (CTS)** describe un conjunto de tipos que pueden usar diferentes lenguajes que tienen .Net en común, que aseguran que los objetos escritos en diferentes lenguajes puedan interactuar entre sí. La mayoría de los miembros definidos por tipos en la Biblioteca de clases de .NET Framework son tipos compatibles con la especificación de lenguaje común. Además, la especificación de lenguaje común está estandarizada por ECMA.

Estos tipos pueden ser tipos de valor o tipos de referencia. Los tipos de valor se pasan por valores y se almacenan en la pila. Los tipos de referencia se pasan por referencias y se almacenan en el heap. El Common Type System proporciona un conjunto básico de tipos de datos, que es responsable de la integración entre lenguajes.

El Common Language Runtime (CLR) puede cargar y ejecutar el código fuente escrito en cualquier lenguaje .NET, sólo si el tipo se describe en el Sistema de tipos común (CTS). La mayoría de los miembros definidos por los tipos en el .NET Framework Class Library (FCL) son tipos compatibles con la especificación de lenguaje común (CLS).

MSIL significa **Microsoft Intermediate Language**, también es conocido como **Lenguaje Intermedio (IL)** o **Lenguaje Intermedio Común (CIL)**. Durante el tiempo de compilación, el compilador convierte el código fuente a Microsoft Intermediate Language. Microsoft Intermediate Language es un conjunto de instrucciones independientes de la CPU que se pueden convertir de manera eficiente al código nativo. Durante el tiempo de ejecución, el compilador **Just In Time (JIT)** del Common Language Runtime (CLR) convierte el código de Microsoft Intermediate Language en código nativo para el sistema operativo. Cuando el compilador produce Microsoft Intermediate Language, también produce Metadatos. Estos metadatos junto con el lenguaje intermedio de Microsoft están contenidos en un archivo ejecutable portátil (PE).

Microsoft Intermediate Language contiene las instrucciones para cargar, almacenar, inicializar y llamar a métodos en objetos, así como instrucciones para operaciones aritméticas y lógicas, flujo de control, acceso directo a la memoria, manejo de excepciones y otras operaciones.

El **formato Portable Executable (PE)** es un formato para archivos ejecutables, código de objetos y DLLs, que se usa en las versiones de 32 y 64 bits de los sistemas operativos Windows. El formato de archivo PE nos proporciona la mejor manera para que el sistema operativo Windows ejecute el código y también para almacenar los datos esenciales que se necesitan para ejecutar un programa.

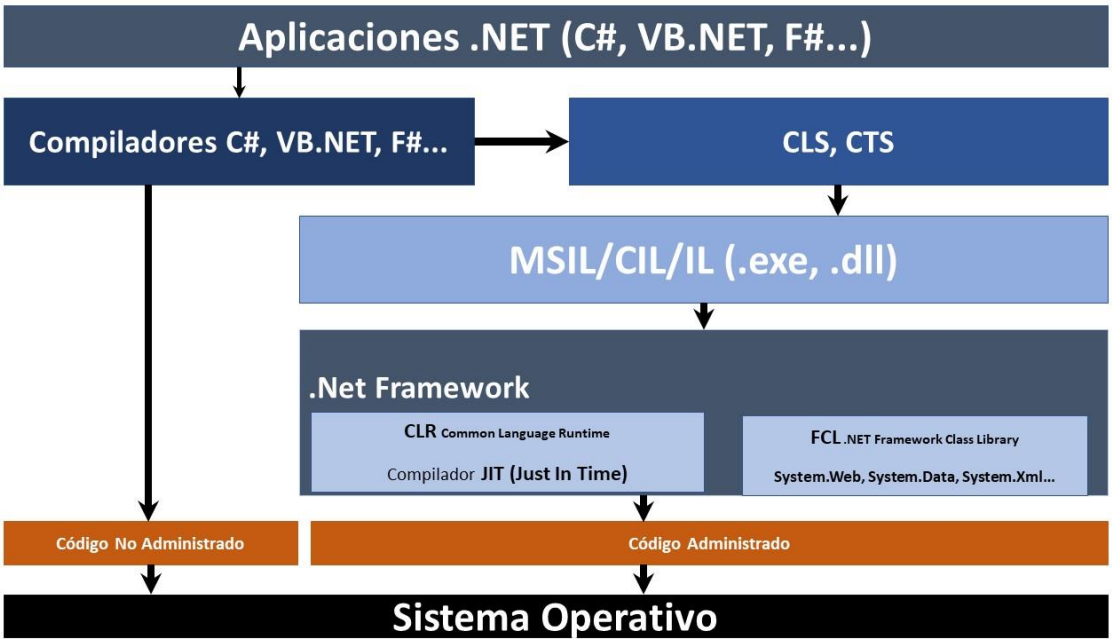
Los lenguajes .Net utilizan su tiempo de ejecución correspondiente para ejecutar la aplicación en diferentes Sistemas Operativos. Durante el tiempo de ejecución del código, el Código administrado se compila solo cuando es necesario, es decir, convierte las instrucciones apropiadas al código nativo para su ejecución justo antes de que se llame a cada función. Este proceso se denomina compilación Just In Time (JIT). Con la ayuda de la compilación Just In Time, el Common Language Runtime (CLR) puede realizar realiza sus tareas.

El Common Language Runtime (CLR) proporciona varios compiladores Just In Time y cada uno funciona en una arquitectura diferente según el sistema operativo. Es por eso que el mismo lenguaje intermedio de Microsoft (MSIL) se puede ejecutar en diferentes sistemas operativos sin tener que volver a escribir el código fuente. La compilación Just In Time conserva la memoria y ahorra tiempo durante la inicialización de la aplicación.

El código administrado en .Net Framework, es el código que ha ejecutado el entorno de Common Language Runtime (CLR). Por otro lado, el código no administrado es ejecutado directamente por la CPU de la computadora. Los tipos de datos, los

mecanismos de manejo de errores, las reglas de creación y destrucción y las pautas de diseño varían entre los modelos de objetos administrados y no administrados.

Los beneficios del código administrado incluyen la conveniencia de los programadores y la seguridad mejorada. El código administrado está diseñado para ser más confiable y robusto que el código no administrado, los ejemplos son la recolección de basura, la seguridad de tipos, etc. El código administrado que se ejecuta en un Common Language Runtime (CLR) no se puede acceder desde fuera del entorno de ejecución y no se puede llamar directamente desde el exterior del entorno de ejecución. Esto hace que los programas estén más aislados y, al mismo tiempo, las computadoras son más seguras. El código no administrado puede omitir .NET Framework y hacer llamadas directas al sistema operativo. Llamar al código no administrado representa un riesgo de seguridad importante.



Los metadatos en .Net son información binaria que describe las características de un recurso. Esta información incluye la descripción del ensamblaje, Tipos de datos y miembros con sus declaraciones e implementaciones, referencias a otros tipos y miembros, permisos de seguridad, etc. Los metadatos de un módulo contienen todo lo necesario para interactuar con otro módulo.

Durante el tiempo de compilación, los metadatos se crean con el Microsoft Intermediate Language (MSIL) y se almacenaron en un archivo llamado Manifest. Tanto los metadatos como el lenguaje intermedio de Microsoft (MSIL) se envuelven juntos en un archivo ejecutable portátil (PE). Durante el tiempo de ejecución de un programa Just In Time (JIT), el compilador Common Language Runtime (CLR) utiliza los metadatos y convierte el lenguaje intermedio de Microsoft (MSIL) en código nativo. Cuando se ejecuta el código, el tiempo de ejecución carga metadatos en la memoria y lo remite para descubrir información sobre las clases, los miembros, la herencia de su código, etc.

.Net Assembly es una unidad lógica de código que contiene el código que ejecuta Common Language Runtime (CLR). Es la unidad más pequeña de implementación de una aplicación .net y puede ser una .dll o un exe. El ensamblaje es en realidad una colección de tipos e información de los recursos que se crean para trabajar juntos y formar una unidad lógica de funcionalidad. Contiene tanto el archivo ejecutable de la aplicación que puede ejecutar directamente desde Windows sin la necesidad de otros programas (archivos .exe) como bibliotecas (archivos .dll) para uso de otras aplicaciones.

Los ensamblajes son los componentes básicos de las aplicaciones de .NET Framework. Durante el tiempo de compilación, los metadatos se crean con Microsoft Intermediate Language (MSIL) y se almacenan en un archivo llamado Assembly Manifest. Tanto los metadatos como el lenguaje intermedio de Microsoft (MSIL) se envuelven juntos en un archivo ejecutable portátil (PE). El manifiesto de la asamblea contiene información sobre sí mismo. Esta información se llama Manifiesto de ensamblaje y contiene información sobre los miembros, los tipos, las referencias y todos los demás datos que el tiempo de ejecución necesita para la ejecución.

Cada ensamblaje que cree contiene uno o más archivos de programa y un manifiesto. Hay dos tipos de archivos de programa: Ensamblajes de proceso (EXE) y Conjuntos de biblioteca (DLL). Cada ensamblado solo puede tener un punto de entrada, es decir, DllMain, WinMain o Main.

Podemos crear dos tipos de Ensamblados: privados o compartidos.

Un **ensamblado privado** solo puede ser utilizado por una sola aplicación, y generalmente se almacena en el directorio de instalación de esa aplicación. Un **ensamblado compartido** es aquel al que se puede hacer referencia mediante más de una aplicación. Si varias aplicaciones necesitan acceder a un Assembly, deberíamos agregarla a la **Caché Assembly Global (GAC)**. También hay un tercer tipo de ensamblado: Ensamblado satélite. Un **ensamblado satélite** solo contiene objetos estáticos como imágenes y otros archivos no ejecutables requeridos por la aplicación.

.Net Assembly Manifest es un archivo que contiene Metadatos sobre ensamblados .NET. El Manifiesto de ensamblado contiene una colección de datos que describe cómo los elementos del ensamblado se relacionan entre sí. Describe la relación y las dependencias de los componentes en el ensamblado, la información de las versiones, la información del alcance y los permisos de seguridad requeridos por el ensamblado.

El manifiesto de ensamblado se puede almacenar en un archivo ejecutable portátil (PE) con el código de lenguaje intermedio de Microsoft (MSIL). Puede agregar o cambiar alguna información en el manifiesto de ensamblado utilizando los atributos de ensamblado en su código.

Cada computadora en la que está instalado Common Language Runtime tiene un caché de código para toda la máquina llamado Global Assembly Cache. GAC es una carpeta

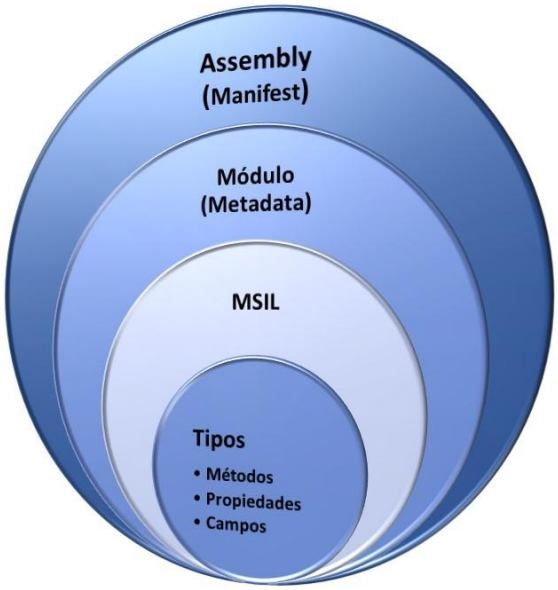
en el directorio de Windows para almacenar los ensamblados .NET que están específicamente diseñados para ser compartidos por todas las aplicaciones ejecutadas en un sistema operativo. Los ensamblados se pueden compartir entre varias aplicaciones en la máquina registrándolos en la caché de ensamblados global (GAC).

El GAC se instala automáticamente con el tiempo de ejecución .NET. El caché de ensamblado global se encuentra en el directorio Windows/WinNT y hereda la lista de control de acceso del directorio que los administradores han usado para proteger la carpeta.

El enfoque de tener un repositorio central especialmente controlado aborda el concepto de biblioteca compartida y ayuda a evitar los inconvenientes de otras soluciones que conducen a inconvenientes como el infierno de DLL.

Un ensamblado.NET puede constar de los siguientes elementos:

- Manifiesto de ensamblado: los metadatos que describen el ensamblado y su contenido.
- Metadatos de tipo: define todos los tipos, sus propiedades y métodos.
- MSIL - lenguaje intermedio de Microsoft
- Un conjunto de recursos: todos los demás recursos como iconos, imágenes, etc.



Los **espacios de nombres** son la forma de organizar la .NET Framework Class Library en una agrupación lógica de acuerdo con su funcionalidad, facilidad de uso y la categoría a la que deben pertenecer, o podemos decir que los espacios de nombres son agrupaciones lógicas de tipos con el propósito de identificación.

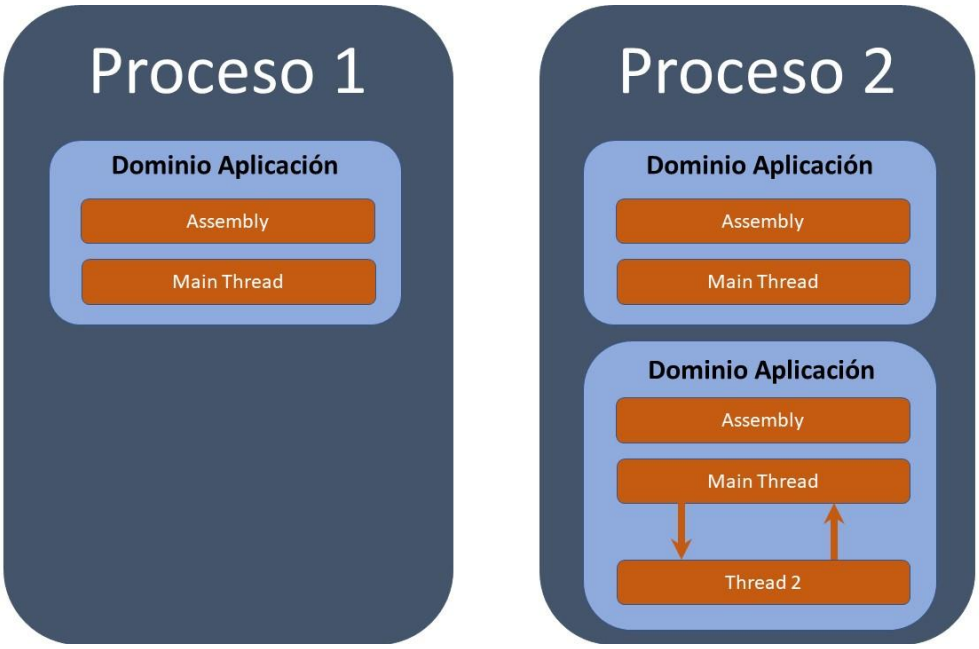
La biblioteca de clases de .NET Framework (FCL) es una gran colección de miles de clases. Estas clases están organizadas en un árbol jerárquico. Los Espacios de nombres del sistema son la raíz de los tipos en el .NET Framework. Podemos identificar de

forma única cualquier clase en la biblioteca de clases de .NET Framework (FCL) utilizando los espacios de nombres completos de la clase. En los lenguajes .Net, todos los programas se crean con espacios de nombres predeterminados. Los programadores también pueden crear sus propios espacios de nombres en lenguajes .Net.

Un **dominio de aplicación** es un proceso virtual y se utiliza para aislar aplicaciones entre sí. Cada dominio de aplicación tiene su propio espacio de direcciones virtuales que abarca los recursos para el dominio de la aplicación que utiliza ese espacio de direcciones.

Cada aplicación se ejecuta dentro de sus límites de proceso principales y sus límites de dominio de aplicación. Todos los objetos creados dentro del mismo ámbito de aplicación se crean dentro del mismo dominio de aplicación. Pueden existir múltiples dominios de aplicación en un solo proceso de sistema operativo. Una aplicación que se ejecuta dentro de un dominio de aplicación no puede acceder directamente al código que se ejecuta dentro de otro dominio de aplicación.

La clase principal que puede usar para tratar con dominios de aplicaciones es System.AppDomain.



El .Net Framework proporciona un mecanismo para la liberación de los objetos sin referencias en la memoria, este proceso se llama **Recolección de basura**. Cuando un programa crea un objeto, el objeto ocupa la memoria. Más tarde, cuando el programa no tiene más referencias a ese objeto, la memoria del objeto se vuelve inalcanzable, pero no se libera de inmediato. La recolección de basura comprueba si hay objetos en la memoria heap que la aplicación ya no utiliza. Si esos objetos existen, entonces la memoria utilizada por estos objetos puede ser reclamada. Por lo tanto, estos objetos no referenciados deben eliminarse de la memoria, de esta manera los nuevos objetos que se van a ir creando podrán encontrar un lugar en la memoria heap libre.

La finalización permite que un recurso se limpie después de ser recolectado. Esta liberación de objetos no referenciados ocurre automáticamente en los lenguajes .Net gracias al recolector de basura. Podemos llamar explícitamente al recolector de basura (Garbage Collection) en el programa llamando a System.GC.Collect.

Assembly representa un bloque de código reutilizable, versionable y autodescriptivo de una aplicación Common Language Runtime. El espacio de nombres de System.Reflection contiene tipos que recuperan información sobre conjuntos, módulos, miembros, parámetros y otras entidades en código administrado mediante el examen de sus metadatos.

Dentro del espacio de nombres System.Reflection.Assembly podemos usar el método AppDomain.GetAssemblies para recuperar una matriz de objetos de ensamblado que representan los ensamblados actualmente cargados en un dominio de aplicación.

```
AppDomain _appDomain = null;

System.Reflection.Assembly[] myAssemblies = null;

System.Reflection.Assembly myAssembly = null;

_appDomain = AppDomain.CurrentDomain;

myAssemblies = _appDomain.GetAssemblies();
```

Podemos usar la clase Assembly para cargar ensamblados, explorar los metadatos y las partes constitutivas de los ensamblados, descubrir los tipos contenidos en los ensamblados y crear instancias de esos tipos.

El **Registro** es una ubicación óptima para guardar información sobre una aplicación, así como la configuración individual del usuario.

El Registro comprende una serie de secciones lógicas llamadas hives (colmenas). Las siguientes son las claves predefinidas que utiliza el sistema.

HKEY_CURRENT_USER

HKEY_USERS

HKEY_LOCAL_MACHINE

HKEY_CLASSES_ROOT

HKEY_CURRENT_CONFIG

Cada clave tiene muchas subclaves y pueden tener un valor.

Al programar en C #, podemos elegir acceder al registro a través de las funciones proporcionadas por C # o las clases de registro de .NET Framework. Las operaciones en el registro en .NET se pueden realizar usando dos clases del espacio de nombres

Microsoft.Win32: clase de Registry y la clase RegistryKey. La clase de Registry proporciona claves de registro base como métodos públicos compartidos:

```
using Microsoft.Win32;

// creación de una entrada de registro

claveRegistro = Registry.LocalMachine.OpenSubKey("SOFTWARE", true);

// cómo crear una subclave bajo HKLM\Software llamada AppReg

claveRegistro.CreateSubKey("AppReg");

// Eliminar una subclave

claveRegistro = Registry.LocalMachine.OpenSubKey("Software", true);

claveRegistro.DeleteSubKey("AppReg", true);
```

FIN