

Fwk.Caching

BLOQUE DE CACHING

El sistema de caching que provee el Framework permite a las aplicaciones (Back-End o Front-End) disponer un conjunto de mecanismos que facilitan el almacenamiento en IsolatedStarages customizables de información persistente que sea requerida en las aplicaciones.

El mecanismo de caching que usa la arquitectura de Fwk es básico y en memoria. Internamente aplica el cache de .net framework.

Para implementaciones más avanzada se debería diseñar comportamientos más específicos a otros modelos de caching existentes en mercado:

- ✓ NCache
- ✓ Memcached
- ✓ Velocity

Modelo de componentes

CacheManager

Proporciona una abstracción al manejo de la cache. Esta clase contiene los métodos para manipular la cache y reúne en un punto común la tecnología de caching, de esta manera si se desea dejar de actualizar los App Block de P&P simplemente se cambia la implementación interna de esta clase.

Ejemplo

```
ClienteBE cli = new ClienteBE();
cli.IdCliente = 50999;
cli.Apellido = "Flimming";
cli.Edad = 69;

CacheManager.Add(cli.IdCliente.ToString(),wCli, 1, DateFunctions.TimeMeasuresEnum.FromMinutes);
```

Código 1.0

Métodos

Nombre de método	Descripción
Add(string key, object value, Int32 timeToLive, DateFunctions.TimeMeasuresEnum timeToLiveTimeMesures)	Agrega un nuevo item a la cache por un tiempo determinado por timeToLive y timeToLiveTimeMesures. Este método agrega el item hasta un tiempo determinado. Concluido ese tiempo el elemento es removido del cache Si algún elemento existe con ese mismo nombre será eliminado antes de que el nuevo

	value sea agregado.
Contains(string key)	Determino si el Item Existe en Caché
Add_WithExpirationTiem(string key, object value, Double slidingTime, Fwk.HelperFunctions.DateFunctions.TimeMeasuresEnum slidingTimeTimeMeasures)	Agrega un nuevo item a la cache Si algún elemento existe con ese mismo nombre será eliminado antes de que el nuevo value sea agregado. La expiración del elemento en cache depende del último momento de uso.
Flush ()	Borra todos los ítems de Caché
Remove (String key)	Eliminar un Ítem de Caché pCahcheId = Clave con el que se guardó el objeto
GetData (String key)	Recupera un Ítem de Cache dependiendo del Identificador del mismo. pCahcheId = Clave con el que se guardó el objeto

Tabla 1

Implementación de Caching en los servicios Fwk

Viéndolo desde esta perspectiva integrada en la arquitectura tenemos dos puntos principales para aplicar caching.

Por un lado todos los servicios disponen de una sección llamada **CacheSettings** donde se establecen todos los atributos necesarios para almacenar o leer de la Cache.

Atributos

Nombre de atributo	Descripción
CacheOnServerSide	Bandera que indica si los resultados de la ejecución del servicio serán primero intentados obtener desde la cache del lado del servidor
CacheOnClientSide	Bandera que indica si los resultados de la ejecución del servicio serán primero intentados obtener desde la cache del lado del cliente
ResponseCacheId	Identificador de la cache para el caso de que el servicio este cacheado tanto en el lado del cliente como en el servidor. Puede proporcionarle cualquier identificador de cache. Por ejemplo: 1- El mismo nombre del servicio 2 - Nombre de servicio más fecha 2 - Nombre de servicio más Dominio/ Area donde corra el cliente o servidor Si este valor es = Empty() y alguna de las CacheOnClientSide o CacheOnServerSide están establecidas en true se asume el Id de la cache del servicio con el nombre del servicio.- EJ:

	<pre>BuscarPaisesClienteRequest req = new BuscarPaisesRequest(); req.ResponseCacheId = req.ServiceName + "RRHH"; req.CacheOnServerSide = true;</pre>
ExpirationTime	Cantidad de tiempo acorde a TimeMeasures
TimeMeasures	Métrica (Días, Horas, Min o Seg)

Tabla 2

Los componentes de un servicio que disponen de objetos relacionados a caching integrado son los Request.

Porque esto?

Esto es porque un request es el punto de entrada para la solicitud de un servicio por lo tanto es este quien le informa al servicio si los datos se obtienen de un repositorio de cache o de un servidor de aplicaciones que ejecuta comandos de base de datos u otro origen diferente al cache.

También decide si la cache se va a aplicar del lado del servidor de aplicaciones o del lado del cliente.

Ejemplo:

Supongamos un servicio de búsqueda de localidades que es de muy poca actualización y se decide dejar almacenada en memoria o disco sin necesidad de ir por cada petición al servidor de aplicaciones.

El código se vería como sigue:

```
SearchRelatedDomainsByUserReq req = new SearchRelatedDomainsByUserReq();  
req.CacheSettings.CacheOnClientSide = true;  
req.CacheSettings.ResponseCacheId = string.Concat(req.ServiceName + puserName);  
req.CacheSettings.ExpirationTime = 2;  
req.CacheSettings.TimeMeasures = DateFunctions.TimeMeasuresEnum.FromHours;
```

Código 2.0

Aquí se decidió para este caso cachear las localidades de córdoba en el lado de cliente con un tiempo de expiración de 60 días.

Nota: ResponseCacheId determina el nombre que identifica el ítem almacenado en la cache.

Si no se especifica nada se tomara el nombre del servicio.

En este caso es recomendable establecer un nombre para identificar las búsquedas de Localidades de diferentes provincias.

Escenarios de uso

Ejemplo de buenos escenarios de uso son comúnmente cuando las aplicaciones necesitan obtener listados de Categorías, Clasificaciones de tipos, Países, Localidades etc.

Lo que tienen en común estas entidades es que con muy baja frecuencia son modificadas de su origen de datos.

Con el fin de evitar los round-trips innecesarios para al obtener siempre la misma información se decide persistirlas en un medio de almacenamiento más veloz y que no tenga tanto costo de uso de servidores.

También es posible persistir otro tipo de información con más tasa de modificaciones, pero que lo mismo sigue siendo necesario obtenerlos rápidamente. Ejemplo de estos pueden ser: permisos de usuarios, listado de proveedores, etc.

Si bien estos tipos de entidades pueden ser alterados día a día, las aplicaciones generalmente necesitan consultarlas varias veces en el transcurso del mismo y es muy poco probable que una modificación o agregado de nuevos registros para estas entidades altere la tarea diaria que tiene un usuario en su jornada.

También es posible desde cualquier punto de la aplicación limpiar intencionalmente la cache de modo que el sistema vuelva a consultar los datos desde su origen real.

Ejemplo:

```
CacheManager.Flush();
```

Código 4.0

FwkSimpleStorageBase

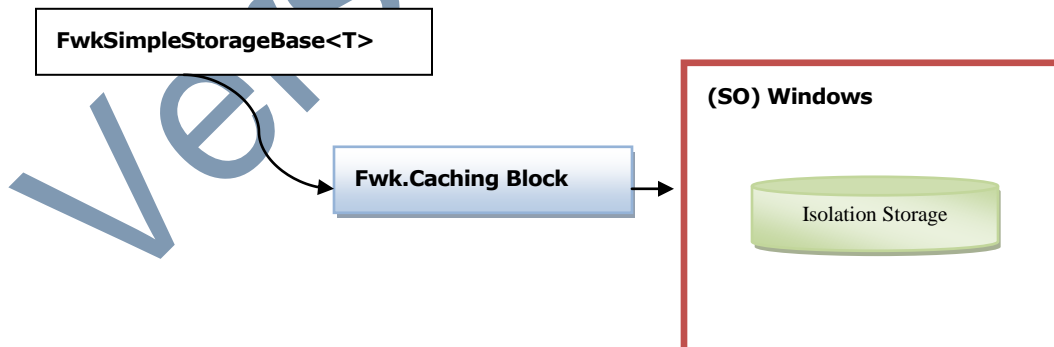
Este componente permite mantener la persistencia de objetos de una manera muy sencilla y sin ningún tipo de configuración.

FwkSimpleStorageBase no utiliza las Enterprise Library para realizar sus tareas de cache.

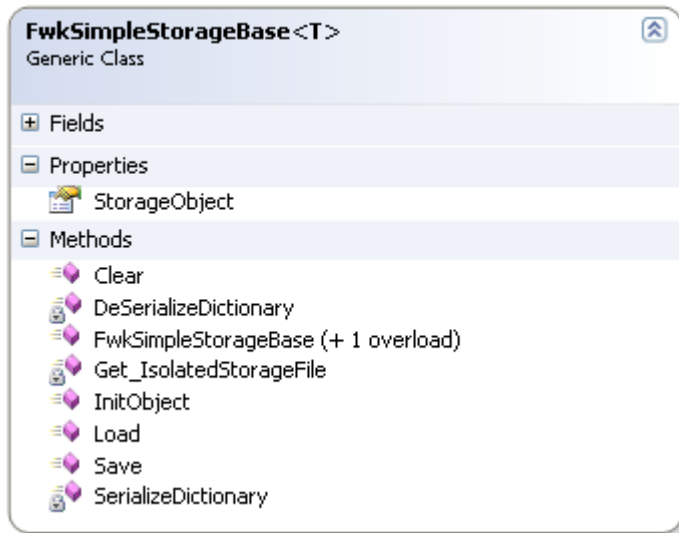
No se configura ningún app.config

Es una clase genérica, por lo tanto permite almacenar la información de cualquier tipo de objeto.

Esta clase es comúnmente utilizada en entornos Win32 cuando se desea mantener la persistencia de inputs del usuario.



Propiedades y métodos:



StorageObject → T	Clase generica que reprecenta el objeto serializable que se almacena en la cache del sistema
InitObject	Si el objeto necesita ser inicializado con algunos valores se debe sobrescribir esta clase donde se inicializa el StorageObject
Load	Permite cargar el almacenamiento del objeto.. Este metodo llama al metodo virtual InitObject. Generalmente se usa desde el Load del formulario
Save.	Almacena en cache el objeto

Como se utiliza?

Este objeto se puede utilizar de dos maneras. La primera es simplemente declarando un objeto de tipo FwkSimpleStorageBase<T> donde T será el tipo de objeto serializable a almacenar en cache. La segunda es utilizar FwkSimpleStorageBase<T> como clase base y así de esta forma poder sobrescribir el método virtual InitObject

Vamos a un ejemplo para ambas situaciones con el supuesto de que nuestra clase a inicializar y cachear representa ciertos valores de campos de texto en un formulario:

- ✓ LastFile (nombre de ultimo archivo accesadoarchivo)
- ✓ LastAccess (Ultimo acceso del usuario)

A) Primera forma

Si tenemos un formulario y deseamos almacenar en cache una clase llamada **FormInit**, primero debemos declarar:

1.

```
FwkSimpleStorageBase<FormInit> _Storage = new FwkSimpleStorageBase<FormInit>();
```

2. En el Load del Formulario podemos autocargar el storage

```
_Storage.Load();
```

En este momento el objeto ya esta inicializado y cargado desde la chache si es que en algun momento se lo almaceno. Si no se encuentra el objeto en cache, el componente automaticamente lo instancia utilizando Fwk.Reflection Components.-

3. Simplemente utilizar:

```
txtLastFile.Text = _Storage.StorageObject.LastFile;
```

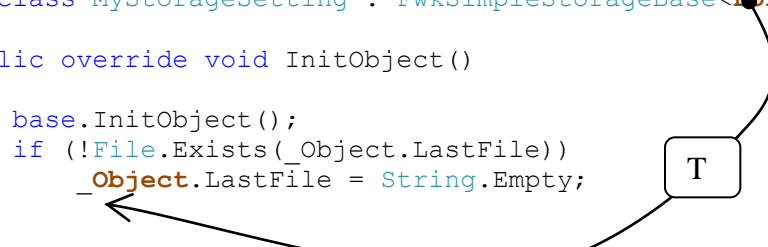
4. Recuerde siempre ejecutar el método **Save** para informarle al componente **FwkSimpleStorageBase** que debe almacenar en cache el objeto. De lo contrario la proxima vez que levante la aplicación los valores que apareceran en el objeto no representaran los ultimos datos que el usuario ingreso en la aplicación. Generalmente el método **Save** se llama en el método **Closing** del formulario

B) Segunda forma

Existen situaciones donde los valores del objeto (ej: formInit) deben ser pasados como parámetros o inicializados de manera especial. En tal caso podemos hacer una clase Custom que herede de **FwkSimpleStorageBase**

1)

```
internal class MyStorageSetting : FwkSimpleStorageBase<FormInit>
{
    public override void InitObject()
    {
        base.InitObject();
        if (!File.Exists(_Object.LastFile))
        {
            _Object.LastFile = String.Empty;
        }
    }
}
```



```
internal class MyStorageSetting : FwkSimpleStorageBase<FormInit>
{
    public override void InitObject()
    {
        base.InitObject();
        _Object = new Object(new DateTime())
    }
}
```

Aquí el desarrollador lo que necesita es utilizar una sobrecarga del constructor para construir el objeto a cachear. Por lo tanto quita la inicialización automática del componente **FwkSimpleStorageBase**

Versión preliminar