Testing unitario

Fecha de confección:	29/09/2008
Título:	Testing unitario para servicios svc del fwk
Temática:	Aplicativos .NET
Confeccionado por:	Marcelo Oviedo

Objetivos

El propósito del presente documento es describir la normativa de codificación de los componentes de testing unitario de los servicios desarrollados bajo la arquitectra estándar de aplicaciones.

Alcance

Esta normativa es aplicable a todo desarrollo de aplicaciones en .NET, ya sea internas de ALLUS o Tercerizadas.

Elaboración / Revisión / Aprobación

El presente documento es elaborado por el **Arquitecto Líder de la División QA**, Revisado por el **Responsable de División QA** y aprobado por la **Gerencia de Desarrollo.**

Roles y Responsabilidades

Rol	Responsabilidad
Gerencia de desarrollo	Aprobar el estándar de Desarrollo.
Arquitecto Líder	Definir Estándares y auditar el estándar.
Responsable División QA	Revisión general del estándar.

Introducción

Los test unitarios son las únicas pruebas que existen en programación de tal forma que hay una integración continua entre la codificación y las pruebas, estas pruebas son las que dan lugar a una re-fabricación constante del código.
Como se menciona, son las que atañen más directamente al programador. Una prueba de test unitario es una porción de código que prueba un área de funcionalidad, generalmente pequeña, del código que se desea testear. Por ejemplo podemos pensar en un test unitario que se encargue de validar si un método obtiene los campos esperados de una tabla y su macheo correspondiente de nombres. Esto permitirá en un futuro determinar alguna falla por modificación de esquemas de BD.

Requerimientos Generales

Este documento no tiene por objetivo enceñar a realizar test unitarios ni explicar el framework de unit test. Solo va a mostrar siertos patrones de testing que se deberán respetar en todos los desarrollos de los componentes de servicios.

Los diferentes componentes que vamos a testear seguirán un patrón común y los componentes de testing deberán contener "todos" los mismos objetos declarados a fin de no tener multiples unidades de test heterogeneas y fasilitarle al desarrollador desplasarce de un proyecto a otro y poder comprender la lógica de test.

Componentes que aplican:

- ✓ Servicios SVC
- ✓ Componentes de negocio BC
- ✓ Componentes de acceso a datos DAC

Por el momento se exige únicamente las pruebas unitarias de cada servicio desarrollado. Los demás componentes quedan a criterio del desarrollador y serán desarrollados dentro del mismo proyecto de test pero en clases diferentes de donde se ubiquen los test de servicios.

Tecnologia:

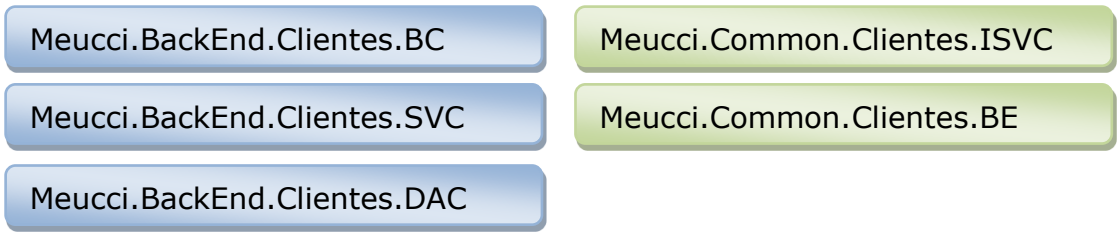
La tecnología para efectuar test unitarios será la que esta integrada con visual studio 2008. Es una potente herramienta de test y respeta los estándares de NUnit. Por lo tanto aquel desarrollador que ah echo pruebas con NUnit no tendrá que aprender mucho para adaptarse a esta tecnlogia.

Proyecto de testing

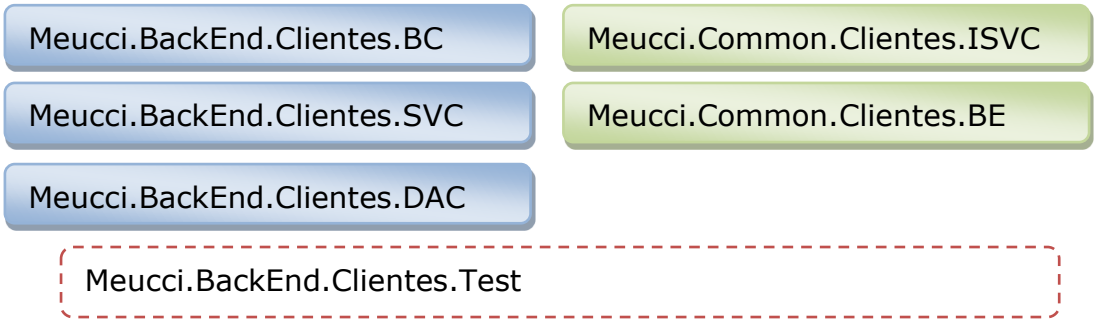
Por cada conjunto de proyectos de back end se generara un proyecto de test asociado.

Ejemplo:

Supongamos el back end de clientes coin sus respecticvos componentes.



A este proyecto le debemos agregar otro de test como muestra la figura siguiente



Contenido del proyecto de test unitario

- ✓ Dentro de este proyecto existirán todas las pruebas unitqarias del dominio de back end de Clientes.

- ✓ Referencias a todos los componentes del dominio
- ✓ Referencia a componentes de otros dominios en caso de que sea necesario *ej:Meucci.BackEnd.Stock.DAC*
- ✓ Referencia a librerías del frameworck (link)
- ✓ Referencias a los componentes de application blocks (link)
- ✓ Archivos xml con información de entrada para los servicios donde los request sean de gran tamaño como para ser pasados por código ej: servicios de creación u modificación.

Clases contenedores de pruebas unitarias

Esta es una clase de .net que implementa la lógica de test unitario por lo tanto deberá estar marcada por el atributo `TestClass`.

```
Ej;
[TestClass()]
    public class ProductsTest
    {
            public TestContext TestContext
        {
            get
            {
                return testContextInstance;
            }
            set
            {
                testContextInstance = value;
            }
        }
}
```

Se exige solo una clase, que es la que contiene el nombre del servicio:

Dominio[ServiceTest]

Ej

ClientesServiceTest
ClientesAltasServiceTest
ClientesCobroanzaServiceTest

El desarrollador es libre de subdividir el test unitario en las partes que crea conveniente. Ademas si el desarrollador nececita hacer test particulares sobre un componente diferente al de servicios (DAC, validador de formulas, etc) es totalmente libre de hacerlo y de echo es conveniente y aconsejable.
Por el momento para hacer una primera adaptación a esta metodología comenzaremos con el test desoló los servicios o SVC.

Trabajaremos con la capa de servicio ya que es una de las piezas mas importantes de la arquietectura y la ejecución de una de ellas requiere la puesta en marcha de todos los demás componentes que están por debajo de ella. (BC DAC ISVC SPs etc.)-

Tipos de test de servicios.

Los test que realizaremos podemos dividirlos en dos:

- ✓ Conectados: Las pruebas unitarias se realizan atravez del despachador de servicio.
- ✓ Locales: Las pruebas unitarias corren directamente en el proyecto.

De estas dos la exigida es la local ya que la quie corre por medio del dispatcher es la misma que la local pero en contexto orientado a servicio donde se exige que el servicio este registrado en la metadata y el servicio (si es remoto) cuente con las actualizaciones del compilado del dominio.

Nota: una muy bena obcion es hacer el test unitario de servicios Conectado pero de manera local. De esta manera el test corre en un dispatcher LOCAL y no requiere el movide de assemblies a un servidor de aplicaciones.

Ejemplo de test unitarios de servicios desconectados o locales:

Patron de nombrado:

[NombreServicio]_NoService()

Servicio no transaccional:

```
[TestMethod()]
public void SearchProductsService_NoService()
  {
        String strErrorResut = String.Empty;
        SearchProductsByParamService svc = new SearchProductsByParamService();
        SearchProductsByParamRequest req = new SearchProductsByParamRequest();


        req.BusinessData.Name = "TORNI";
        req.BusinessData.TipoBusquedaName = Fwk.Common.TipoBusquedaEnum.Comienza;
        req.BusinessData.UserName = "moviedo";

 try
  {
        SearchProductsByParamResponse res = svc.Execute(req);
  }
 catch (Exception ex)
    {
        strErrorResut = Fwk.Exceptions.ExceptionHelper.GetAllMessageException(ex);
    }
 Assert.AreEqual<String>(strErrorResut, String.Empty, strErrorResut);
}
```

Servicio Transaccional:

Los servicios transaccionales a fin de que no ensucien la bace de datos en la ejecucion de pruebas unitarias deberan estan acotadas por un aislamiento de transaccion y deberia ser abortada.

```
[TestMethod()]
public void CreateProductsService_NoService()
  {
  String strErrorResut = String.Empty;

  _Tx = new TransactionScopeHandler(TransactionalBehaviour.RequiresNew,
  IsolationLevel.ReadCommitted, new TimeSpan(0, 0, 15));

  CreateProductsService svc = new CreateProductsService();
  CreateProductsRequest  req = new CreateProductsRequest();

   try
        {
          _Tx.InitScope();

          req.BusinessData = Products.GetFromXml(xmlSurveyBEFile);
```

```
            req.BusinessData.CreatedUserName = "aaguirre";
             CreateProductsResponse res = svc.Execute(req);
            _Tx.Abort();
           }
        catch (Exception ex)
        {
             strErrorResut = Fwk.Exceptions.ExceptionHelper.GetAllMessageException(ex);
        }
}
```

Ejemplo de test unitarios de servicios conectados:

Se deben cumplir ciertos requisitos antes de realizar los test de este tipo:

1. Configurar el app.config del proyecto test los siguientes componentes:

   - Loging
   - Catching
   - Wrapper
   - Metadata de servicios (Dispatcher)

   Nota:Existen muchos archivos de ejemplos solo basta con copiarlos y cambiarles unos pocos parámetros como cadena de coneccion y ruta de acceso ar metadatos.

2. La clase contenedora del testing debe incluir las siguientes declaraciones;

```
TransactionScopeHandler _Tx;
ClientServiceBase _ClientServiceBase = null;
FwkCacheCollectionMannager _FwkCacheCollectionMannager;
```

3. El constructor de la clase de test debe inicializar los components anteriores:

```
public ProductsTest()
     {
         _ClientServiceBase = new ClientServiceBase();
         _FwkCacheCollectionMannager = new FwkCacheCollectionMannager();
     }
```

Servicio Transaccional / Consulta: todos se desarrollan de la misma manera:

```
/// <summary>
///A test for CreateProductsService Constructor
///</summary>
[TestMethod()]
public void CreateProductsServiceTest()
  {
   String strErrorResut = String.Empty;

   CreateProductsRequest   req = new CreateProductsRequest();

   CreateProductsResponse res = _ClientServiceBase.ExecuteService<CreateProductsRequest,
               CreateProductsResponse>(req);

   if (res.Error != null)
    {
      strErrorResut = Fwk.Common.Helpers.ProcessException(res.Error).Message;
    }
```

```
    Assert.AreEqual<Fwk.Exceptions.ServiceError>(res.Error, null, strErrorResut);
}
```