DotNetNuke 3.0 Guía para el Desarrollador de Módulos

Revisión: 1.0.2

Traducido por el equipo de Dnnhipano.com

Tabla de Contenidos

DotNetNuke 3.0 Guía para el Desarrollador de Módulos 1
Tabla de Contenidos 2
Introducción 3
¿Qué son los Módulos? 3
Contenedores del Módulo 5
Interconectar un Módulo con el DotNetNuke 9
Arquitectura del DotNetNuke 9
Modelo de Proveedor 10
Estructura de la Carpeta 13
Empieza el Desarrollo del Módulo 14
Configurar tu Proyecto de Visual Studio.NET 14
Configurar el DNN para que se Interconecte con tus Controles 18
Arquitectura del Módulo 19
Arquitectura de Tres Capas 20
Una Página, Muchos Controles 21
Capa de Datos 22
Capa Lógica de Negocio (BLL) 24
Capa del Usuario 27
Proveedores de Base de Datos Física 34
Interconectar con el DNN 38
Personalización 38
Planificador 39
Proveedor de Registro 41
Localización. Selección del lenguaje y configuraciones locales. 42
Importación y Exportación 42
Búsqueda 44
Editor de Texto 45
Controles para el Usuario del DotNetNuke 46
Captura de Errores 46
Comunicación Inter-Módulo 47
Distribuir Tu Módulo 48
Crear Tu Base de Datos para la Distribución 49
Crear Tu Definición para la Distribución del DNN 50
Validador de la Definición del Módulo 54
Empaquetar tu Módulo 54
Instalación del Ensamblado 55
Resumen 56

Introducción

El propósito de esta guía es ayudar al desarrollador a expandir DotNetNuke (DNN) mediante el desarrollo de módulos que se conectan al núcleo de DNN.
En esta guía se cubrirá cada fase del desarrollo de módulos, desde crear tu proyecto Visual Studio.NET, hasta crear un paquete de distribución para instalar tu nuevo módulo en un portal. El objetivo de esta guía es que te familiarices con el desarrollo de módulos en un corto periodo de tiempo. En principio, lo que se pretende es tratar algunas de las funcionalidades básicas de un módulo que tu usuario querrá ver, para que te introduzcas en lo que un usuario se encontrará en tu portal DNN, y después pasar a algunos conceptos de arquitectura y codificación para que te familiarices con el desarrollo de tus propios módulos.
A través de esta guía se hará referencia a ejemplos reales que están incluidos como partes del archivo de distribución principal del DotNetNuke que se puede descargar de http://www.dotnetnuke.com. En la mayor parte de la guía los ejemplos en los que se centrará serán del módulo Survey. También se utilizará el código de otros puntos incluidos dentro del DotNetNuke, lo que significa que tendrás un fácil acceso a todo el código listado dentro de esta guía.

¿Qué son los Módulos?

Los Módulos proporcionan a los desarrolladores la capacidad de expandir la funcionalidad de DotNetNuke. DotNetNuke proporciona un marco de trabajo (framework) conectable que se puede expandir a través del desarrollo de módulos. DotNetNuke proporciona un módulo contenedor para alojar un módulo a medida (custom). Los Módulos se pueden desarrollar en cualquier lenguaje .NET, aunque DNN está desarrollado en VB.NET, un desarrollador en C# puede crear un módulo que se conecte al núcleo proporcionado por DNN. Este marco conectable se consigue al crear ensamblados privados compilados que exponen y utilizan interfaces específicas para el DNN. Una vez que el ensamblado ha sido compilado, entonces se crea una interfaz del usuario (UI) en forma de archivos ascx que permite a tu usuario interactuar con tu módulo.
Los módulos te proporcionan una máxima reusabilidad del código; puedes descodificar el valor ID del módulo (proporcionado por el marco DNN) que te permite mostrar información única para cada ejecución de tu módulo.
Desde la perspectiva del usuario, un modulo es un área de funcionalidad dentro de su instalación DNN. Por ejemplo, en la Figura 1, puede verse un ejemplo del módulo Text/HTML que viene en DNN. Este módulo proporciona al usuario la capacidad de editar contenido en línea por medio de su navegador Web. Proporciona varias funciones como la edición de texto básico y formateado, y la publicación de contenido.

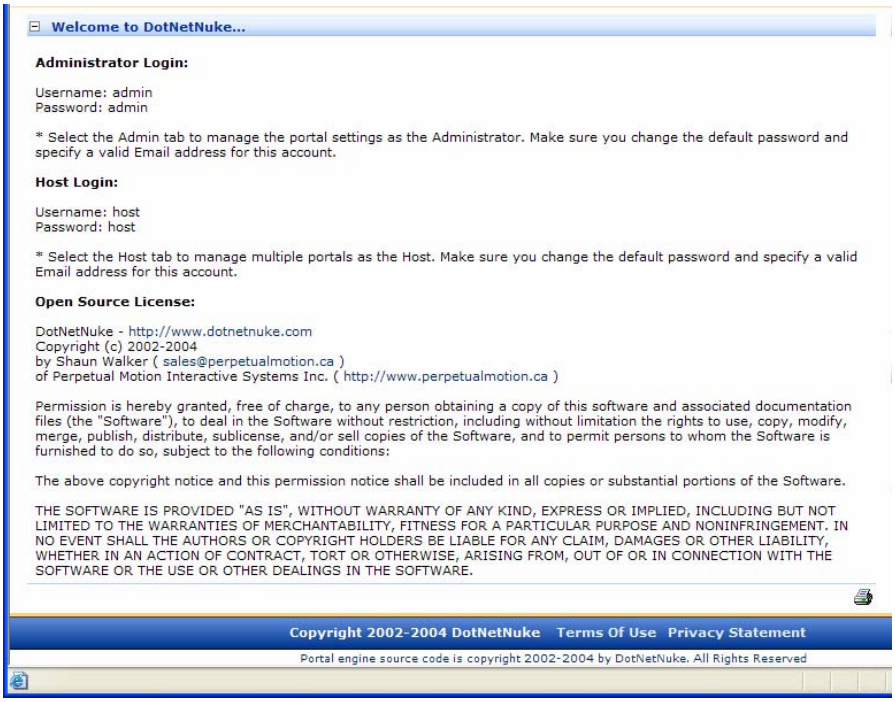


Figura 1 – Vista de usuario del módulo Text/HTML.

En este ejemplo de módulo específico, tenemos dos vistas o comportamientos distintos de este módulo. En la figura 1 tenemos la vista del usuario, que es la HTML que a su vez es la presentación final, y una vista de edición (ver Figura 2), que es donde el administrador del portal, o el autor del contenido pueden editar el contenido directamente en el portal. Por ejemplo, un administrador de portal puede configurar un rol asignando permisos para editar el contenido del módulo Text/HTML, y la próxima vez que un usuario inicie sesión en el portal, y pertenezca a este rol configurado, se le presentará un menú que proporciona varias opciones como la edición (ver Figura 3). El usuario puede entonces seleccionar, de las opciones del menú, editar el contenido.

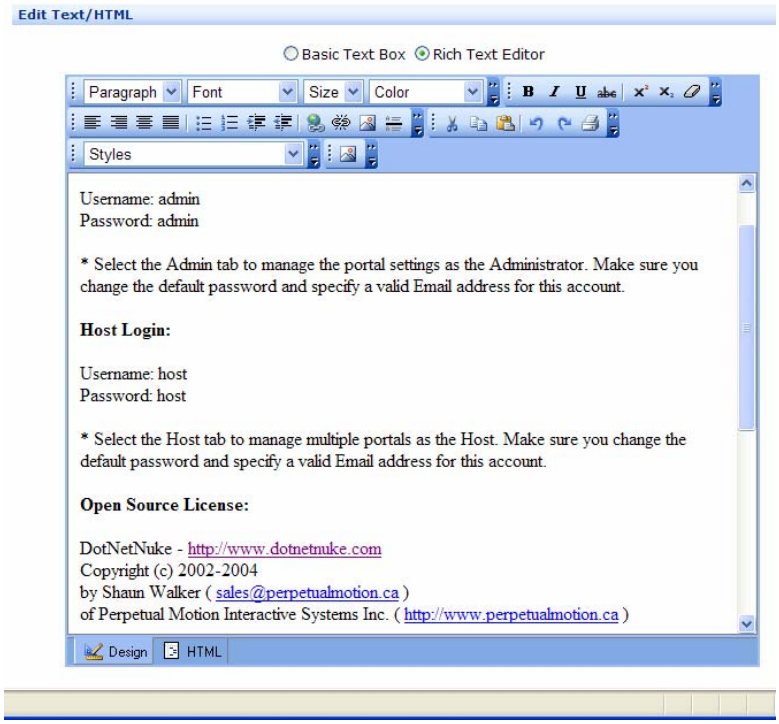


Figura 2 – Vista de la edición del módulo Text/HTML.

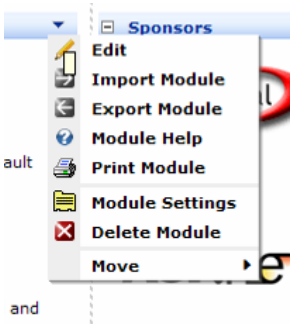


Figura 3 – Opciones del Menú para el módulo Text/HTML

Otro componente del menú a tener en cuenta es el de Configuraciones del Modulo ("Module Settings"). Las Configuraciones del Módulo es una opción global del menú del módulo, que te proporciona la capacidad de modificar algunos de los comportamientos del contenedor del módulo. El contenedor del módulo te proporciona opciones para configurar la seguridad, la apariencia y la caché del módulo. También comprobarás que dentro de las opciones del menú puedes ubicar, importar y exportar módulos

Contenedores del Módulo

Los contenedores del módulo son el contenedor principal (host) o interfaz para tu aplicación dentro de DNN, lo que te da el control para ubicar, cambiar la apariencia, y otras funcionalidades que puedes ver listadas en las opciones del menú. Tratemos las

características que proporciona el contenedor. Primero selecciona Configuraciones del Modulo ("Module Settings") del menú del módulo.
En Configuraciones del Modulo ("Module Settings") verás dos áreas:

Basic Settings (ver Figura 4)
Proporciona a tu módulo las siguientes opciones:

- **Module Title** – Titulo para que se muestre al principio del módulo contenedor.
- **Display Module Container** – Selecciona esta opción si deseas mostrar el elemento gráfico del módulo contenedor que rodea al módulo.
- **Display Module on All Pages** – Utiliza esta opción si quieres que el módulo sea un módulo global y que aparezca en todas las páginas dentro de tu portal DNN.
- **Personalization** – Selecciona esta opción para permitir que tus usuarios personalicen la apariencia del módulo que se muestra en la página, por ejemplo minimizar o maximizar
- **Tab** – Selecciona la página actual donde se alojará el módulo.
- **Header** – Además del Título, puedes mostrar algún texto como encabezado para que aparezca al principio del módulo.
- **Footer** – Introduce un texto para que se muestre como pie de página para la ejecución de este módulo.

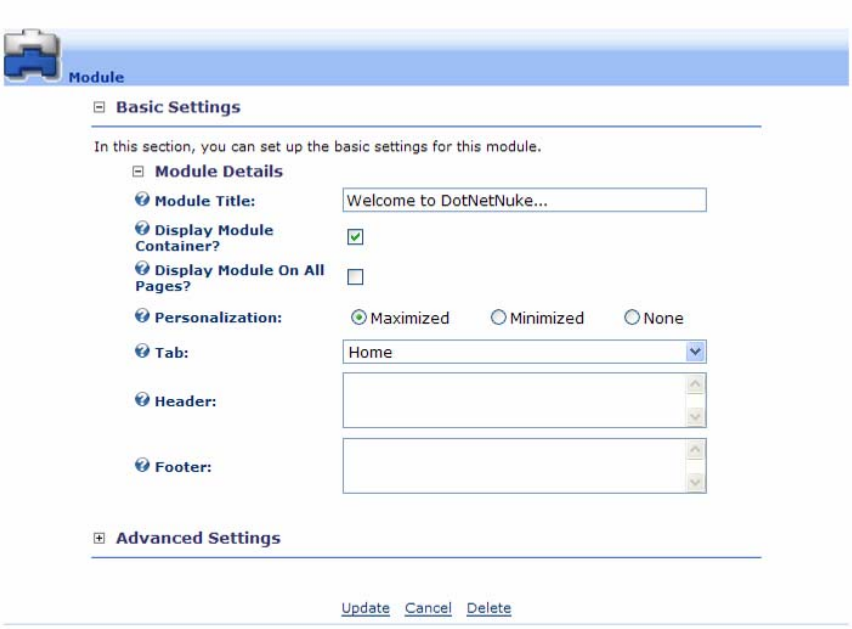


Figura 4 – Configuraciones Basicas para un Módulo

Advanced Settings
Además de las configuraciones básicas, el DNN ofrece algunas configuraciones avanzadas que determinan la apariencia y el ambiente de la ejecución del módulo. Haz clic en el signo más (⊞) en la parte izquierda del encabezamiento para ampliar el área de Advanced Settings (ver Figura 5). Entonces podrás modificar las siguientes configuraciones:

- **Icon** – Selecciona un icono para que se muestre al lado del Titulo del Modulo (primero tendrás que cargar un archivo a través del Administrador de Archivos para que se muestre en la lista desplegable)
- **Alignment** – Selecciona cómo quieres alinear el módulo, izquierda, derecha, centrado o No definido. Esto afecta a las etiquetas HTML cambiando el atributo "align" en la etiqueta de la celda de la tabla HTML que rodea la interfaz del módulo.
- **Color** – Cambia el atributo de color de fondo en la celda de la tabla que contiene el control del módulo.
- **Border** – Cambia el atributo del color del borde enla celda de la tabla que contiene el control del módulo.
- **Module Container** – Esta lista desplegable contiene módulos contenedores instalados en el DNN. Este módulo contenedor es la parte gráfica del módulo y se puede diseñar utilizando un editor estándar. Los módulos contenedores pueden proporcionar una separación entre la apariencia y la funcionalidad. En esta guía estamos tratando el desarrollo de módulos, para obtener más información sobre cómo crear archivos contenedores hay que dirigirse a la guía de skinning en la distribución de DNN.
- **Security Details** – DNN te proporciona módulos con un envoltorio de seguridad. Aquí puedes definir qué roles tienen permisos de edición para tu módulo y qué roles tienen permisos de visualización para tu módulo. Por defecto los permisos se heredan de la página padre.
- **Cache Time** – Si tu módulo no necesita conectarse a una base de datos en cada petición entonces puedes especificar un tiempo de espera en caché. Utiliza esto para las aplicaciones que proporcionan una especie de información estática, esto puede aumentar la ejecución de tu portal reduciendo las llamadas de la red a la bases de datos.
- **Start Date** – Te permite colocar una fecha de inicio de publicación para el módulo. Especifica una fecha en la cual el módulo se mostrará por primera vez.
- **End Date** – Una fecha de caducidad para la muestra del módulo. Introduce una fecha para que este módulo deje de mostrarse en la página.

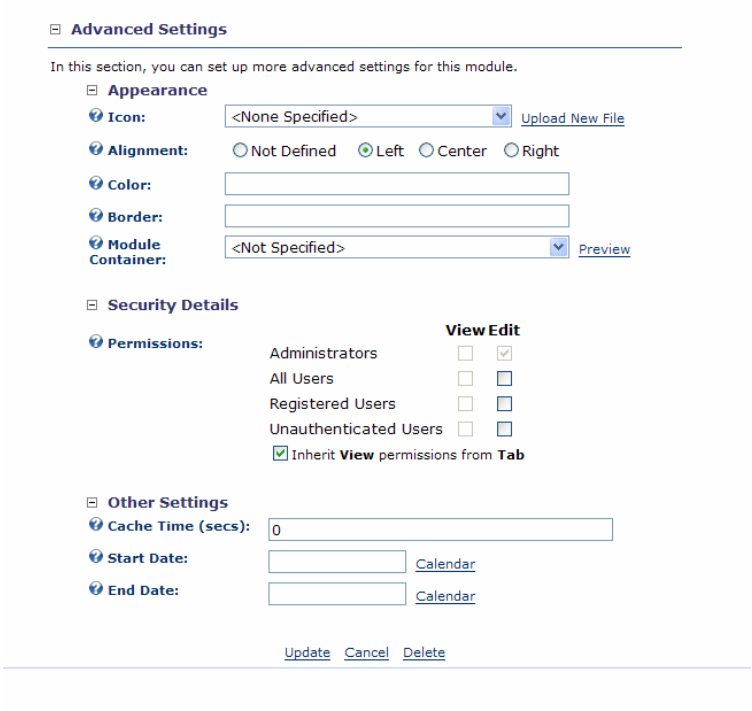


Figura 5 – Advanced Settings para un Módulo

Ubicando Módulos

El módulo contenedor proporciona al editor la capacidad de ubicar el módulo con varios paneles en una página del DNN. Un panel es un área especificada por un diseñador de skins cuando se diseñan skins para un portal del DNN, generalmente un panel será definido dentro de una etiqueta HTML table cell "<td>". Cuando los paneles se definen con el skin, éstos quedarán listados dentro de las opciones del menú en el módulo como aparece en la Figura 6. Un diseñador skin puede crear tantos paneles como se quiera dentro de una página para la ubicación de módulos.

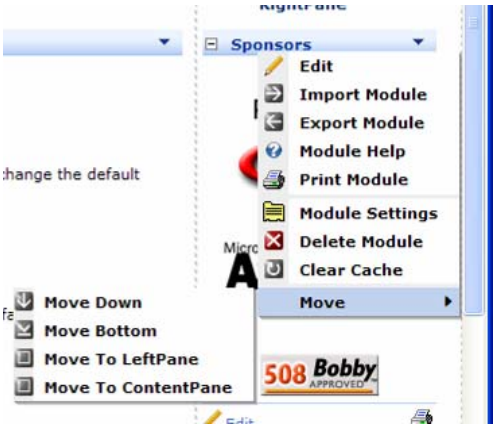


Figura 6 – Opciones de posición con un menú de módulo.

Interconectar un Módulo con el DotNetNuke

Ahora que ya se han tratado las características de un módulo, tenemos que entender cómo un módulo se conecta con el DNN. En este apartado se tratará cómo conectar un modulo manualmente dentro del DNN. En una sección posterior en uso se tratará cómo crear un archivo de definición que automatice el proceso de instalación de la conexión con el DNN.

Importar y Exportar Módulos

Una nueva característica en el DNN 3 es la capacidad de importar y exportar módulos. Esto te permite importar contenido de otro modulo del mismo tipo. Como se ha mencionado previamente, un módulo proporciona reusabilidad descodificando el valor ID de un Módulo proporcionado por el marco del DNN, las características importar/exportar del DNN duplican los datos para el modulo copiando y reemplazando un valor ID de un módulo antiguo por un valor ID de un módulo nuevo del contenedor de destino.
El proceso de exportación de un módulo es muy sencillo; simplemente selecciona la opción de Export Module con la lista del menú del módulo (ver Figura 6). Aparecerá un botón link con el valor de “Export”, sólo haz click en este botón para guardar esta definición de exportar dentro del DNN.
Para importar la información del módulo previamente exportada sólo crea una instancia nueva del módulo dentro de una página, y desde aquí selecciona importar. Entonces aparecerá una pantalla con una lista desplegable con todos los módulos previamente exportados. Selecciona la información del módulo que quieras importar a esta instancia de módulo, y haz click en el botón de “Import”. Más adelante, en esta guía se verá cómo puedes implementar esto en tus propios módulos.

Arquitectura del DotNetNuke

En este apartado se pretende proporcionar algunos conceptos básicos sobre la arquitectura del DNN y cómo ésta te afecta como desarrollador de módulos. Antes de que empieces con el desarrollo nos queremos asegurar de que estés familiarizado con algunos de los conceptos arquitectónicos que conforman la estructura del DNN.
El esquema siguiente (ver Figura 7) está sacado de la documentación del DotNetNuke. Esto proporciona una visión de la arquitectura del DNN. El DNN utiliza una propuesta archiquectónica de tres capas, primero con los controles ascx que proporcionan la interfaz grafica de usuario. La interfaz entonces se comunica con la Capa Lógica de Negocio (BLL), como en el esquema, esta BLL proporciona toda la información para la interfaz del usuario. Debajo de la BLL tenemos el Proveedor de Datos Abstractos **(Abstract Data Provider)**; esta clase proporciona una capa abstracta para nuestra ejecución. Este proveedor no es una base de datos específica; más bien nuestro proveedor de información proporcionará métodos que anulen la clase abstracta y la interacción con nuestra base de datos específica. Debajo de la capa abstracta tenemos nuestra Capa de Acceso de Datos **(Data Access Layer)**, esta clase es específica para un sistema de bases de datos concreto y se basa en la base de datos con la que nosotros queramos interactuar. Esta clase es el proyecto SQLDataProvider, como aparece en la imagen de arriba. Para finalizar, está el Microsoft.ApplicationBlocks.Data que proporciona funciones para la interacción con nuestra base de datos específica, y libera a un desarrollador de tener que escribir un código de Servidor SQL específico.

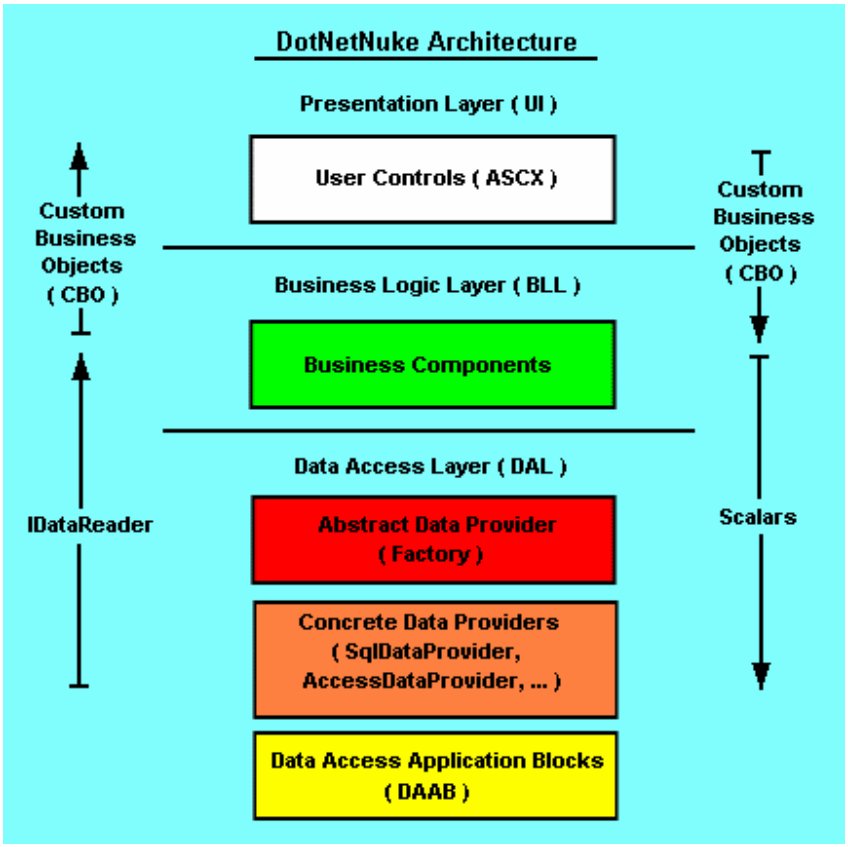


Figura 7 – Arquitectura de Aplicación del DotNetNuke

Modelo de Proveedor

DNN utiliza un modelo de proveedor de uso extendido. Un modelo de proveedor te permite extraer cierta funcionalidad. Por ejemplo, en el esquema arquitectónico (ver Figura 7) puede verse cómo hay un Proveedor de Datos Abstractos **(Abstract Data Provider)** entre los Componentes de Negocio **(Business Components),** y un Proveedor de Datos Concretos **(Concrete Data Provider),** lo que te permite conectarte a diferentes proveedores sin tener que modificar o recopilar el código central dentro del DNN.
DNN utiliza un modelo de proveedor de uso extendido en su arquitectura. Se habló sobre el proveedor de base de datos, pero muchas otras características utilizan también este patrón.

- Security and Membership Provider (Proveedor de Seguridad y de Admisión)
- Text/HTML Provider (Proveedor de Text/HTML)
- Logging Provider (Proveedor de Acceso)
- Scheduler (Planificador)
- Friendly URLs

Como se ha mencionado, los patrones de proveedor te permiten abstraer ciertas funciones ya que los proveedores físicos reales están definidos en el web.config. Por ejemplo, en el DNN si abres el web.config verás un apartado para los proveedores:

```
<dotnetnuke>
	<htmlEditor defaultProvider="Ftb3HtmlEditorProvider">
		<providers>
			<clear />
			<add name="FtbHtmlEditorProvider"
			type="DotNetNuke.HtmlEditor.FtbHtmlEditorProvider,
DotNetNuke.FtbHtmlEditorProvider"
providerPath="~\Providers\HtmlEditorProviders\FtbHtmlEditorProvider\" />
			<add name="Ftb3HtmlEditorProvider"
 type="DotNetNuke.HtmlEditor.Ftb3HtmlEditorProvider,
DotNetNuke.Ftb3HtmlEditorProvider"
ProviderPath="~\Providers\HtmlEditorProviders\Ftb3HtmlEditorProvider\" />
		</providers>
	</htmlEditor>
	<searchIndex defaultProvider="ModuleIndexProvider">
		<providers>
			<clear />
			<add name="ModuleIndexProvider"
type="DotNetNuke.Services.Search.ModuleIndexer, DotNetNuke.Search.Index"
providerPath="~\Providers\SearchProviders\ModuleIndexer\" />
		</providers>
	</searchIndex>
	<searchDataStore defaultProvider="SearchDataStoreProvider">
		<providers>
			<clear />
			<add name="SearchDataStoreProvider"
type="DotNetNuke.Services.Search.SearchDataStore,
DotNetNuke.Search.DataStore"
providerPath="~\Providers\SearchProviders\SearchDataStore\" />
		</providers>
	</searchDataStore>
	<data defaultProvider="SqlDataProvider">
		<providers>
			<clear />
			<add name="SqlDataProvider"
type="DotNetNuke.Data.SqlDataProvider, DotNetNuke.SqlDataProvider"
connectionStringName="SiteSqlServer"
upgradeConnectionString=""
providerPath="~\Providers\DataProviders\SqlDataProvider\"
objectQualifier=""
databaseOwner="dbo" />
		</providers>
	</data>
	<logging defaultProvider="XMLLoggingProvider">
		<providers>
			<clear />
			<add name="XMLLoggingProvider"
type="DotNetNuke.Services.Log.EventLog.XMLLoggingProvider,
DotNetNuke.XMLLoggingProvider"
```

```
configfilename="LogConfig.xml.resources"
providerPath="~\Providers\LoggingProviders\XMLLoggingProvider\" />
            </providers>
      </logging>
      <scheduling defaultProvider="DNNScheduler">
            <providers>
                  <clear />
                  <add name="DNNScheduler"
type="DotNetNuke.Services.Scheduling.DNNScheduling.DNNScheduler,
DotNetNuke.DNNScheduler"
providerPath="~\Providers\SchedulingProviders\DNNScheduler\"
debug="false"
maxThreads="-1"
enabled="true" />
            </providers>
      </scheduling>
      <friendlyUrl defaultProvider="DNNFriendlyUrl">
            <providers>
                  <clear />
                  <add name="DNNFriendlyUrl"
type="DotNetNuke.Services.Url.FriendlyUrl.DNNFriendlyUrlProvider,
DotNetNuke.HttpModules.UrlRewrite" />
            </providers>
      </friendlyUrl>
</dotnetnuke>
```

Si revisas la definición del web.config podrás ver que cada proveedor está definido por una clave, y las propiedades de la configuración específica están especificadas para definir cuál es el proveedor físico que hay que utilizar. Por ejemplo, en el apartado de datos se ve lo siguiente:

```
<data defaultProvider="SqlDataProvider">
```

El defaultProvider se define como SQLDataProvider, entonces dentro de la clave de datos puedes ver la información de configuración para el SQLDataProvider:

```
            <providers>
                  <clear />
                  <add name="SqlDataProvider"
type="DotNetNuke.Data.SqlDataProvider, DotNetNuke.SqlDataProvider"
connectionStringName="SiteSqlServer"
upgradeConnectionString=""
providerPath="~\Providers\DataProviders\SqlDataProvider\"
objectQualifier=""
databaseOwner="dbo" />
            </providers>
```

Lo que puedes ver en el apartado del proveedor del web.config es la información real de la base de datos física. Puedes tener tantos proveedores como quieras definir aquí, y puedes cambiar el valor del defaultProvider para señalar un nuevo proveedor de una base de datos física. Por supuesto, necesitarás tener un proveedor desarrollado y cargado dentro de tu carpeta bin para que se pueda utilizar. Además de tener un proveedor para el DNN, necesitarás tener un proveedor desarrollado para tu módulo que utilice la misma base de datos física que el DNN.

Consejo: Aprende más Cosas sobre los Modelos de Proveedor

Rob Howard detalla el Modelo de Proveedor en un programa de dos partes en el sitio Microsoft's MSDN:
Provider Model Design Pattern and Specification, Part 1:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspnet/html/asp02182004.asp
Provider Design Pattern, Part 2:
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspnet/html/asp04212004.asp?frame=true

Estructura de la Carpeta

DotNetNuke se esfuerza por estructurar lógicamente las carpetas dentro de la carpeta raíz de la aplicación. La siguiente lista proporciona las carpetas bajo una carpeta raíz que son importantes para el desarrollo de módulos y su contenido.

- **<app root>/Admin** – Contiene interfaces de varias funciones administrativas dentro del DNN, por ejemplo, el planificador, la administración del tab, el administrador de búsqueda, y cualquier interfaz de seguridad y accesible para un host o administrador del portal
- **<app root>/App_GlobalResources** – Contiene una aplicación para datos específicos como información del huso horario, y configuraciones de aplicación.
- **<app root>/bin** – Contiene los ensamblados compilados de tu solución DNN, y ensamblados secundarios.
- **<app root>/Components** – Contiene la funcionalidad central para el DNN.
- **<app root>/controls** – Contiene los controles varios para mejorar el interfaz del usuario del DNN, por ejemplo el proveedor de Text/HTML está ubicado dentro de esta carpeta.
- **<app root>/DesktopModules** – Contiene todos los proyectos de módulo para el DNN. Aquí es donde ubicas tu directorio para tu proyecto de módulo.
- **<app root>/HttpModules** – Todos los proyectos HTTPModule están contenidos aquí como el manejador de excepciones, el URLRewriter, el Planificador, y otros.
- **<app root>/images** – Contiene cualquiera de las imágenes utilizadas por el DNN como los iconos.
- **<app root>/js** – Contiene cualquiera de los scripts de cliente en uso por el DNN, scripts del lado del cliente en uso por el DNN, por ejemplo el control del calendario desplegable.
- **<app root>/portals** – Contiene la carpeta portal por defecto, e incluirá cualquiera de las carpetas portal para los portales creados recientemente. Esto proporciona a cualquier portal hijo nuevo su propia carpeta personal para almacenar archivos, y skins para el portal. Las carpetas se identifican por un valor entero único que

coincide con el valor del ID del Portal en la tabla de Portales en la base de datos del DNN.

- **<app root>/Providers** – Contiene los proveedores para tu portal DNN, por ejemplo los proveedores de base de datos están ubicados en un subdirectorio fuera de esta carpeta.
- **<app root>/Resources**
- **<app root>/Web references** – Esta carpeta contiene referencias para los servicios Web, actualmente las excepciones del servicio Web que permiten a un administrador del portal notificar opcionalmente al equipo central del DNN sobre los errores que el portal está generando.

Empieza el Desarrollo del Módulo.

Ahora que ya se ha tratado lo que son los módulos, y cómo se conectan con el DNN, empecemos a desarrollar módulos. En los siguientes apartados trataremos cómo se produce un proyecto Visual Studio.Net, y empezará el desarrollo.

Configurar tu Proyecto de Visual Studio.NET

Alguna de las suposiciones básicas de esta guia es que tengas una máquina de desarrollo configurada con una instalación del DNN por defecto dentro de un directorio virtual. También deberías tener algunos conocimientos básicos sobre el desarrollo de aplicaciones ASP.NET en Visual Studio.NET. Vamos a establecer nuestra instalación del DNN por defecto y a añadir nuestros proyectos de módulo a la solución. Añadiéndolo a la solución principal del DNN te aseguras de que estás compilando al directorio bin del DNN, y permites la depuración de tu proyecto de módulo.

Crear el Proyecto de Módulo

Para crear nuestro proyecto de módulo deberías tener tu solución del DNN abierta, y deberemos añadir un proyecto nuevo a la solución para poder empezar.
Hay varios métodos para crear módulos para el DNN, pero preferimos configurar un proyecto separado para la creación de un módulo y así compilar los ensamblados dentro de cada carpeta papelera del propio módulo. El siguiente procedimiento proporciona un método para configurar un proyecto de módulo del portal en Visual Studio.NET.

1. Crea un proyecto de clase nuevo, asegúrate de usar una estructura lógica de la nomenclatura. Por ejemplo: CompanyName.ModuleName. Utilizando este formato te aseguras de que tu módulo no tendrá ningún conflicto con módulos desarrollados por otras compañías o por otros desarrolladores.
2. Cuando creas una carpeta de proyecto nueva, crea una carpeta que no sea la del directorio del DotNetNuke Root\DesktopModules\.
3. Borra cualquier espacio de nombre de las propiedades del proyecto. Haz click con el botón derecho en las propiedades del proyecto, en las propiedades comunes, en general borra la casilla del espacio de nombres de la raíz.
4. Añade una referencia al proyecto del DotNetNuke en tu nuevo proyecto de módulo.
5. En el proyecto BuildSupport en el directorio <app root>\Solutions\DotNetNuke.DesktopModules\BuildSupport\ añade una referencia a tu proyecto. Al añadir la referencia al proyecto BuildSupport se cogerán las referencias de tu solución para que se formen dentro del directorio bin principal del DotNetNuke.

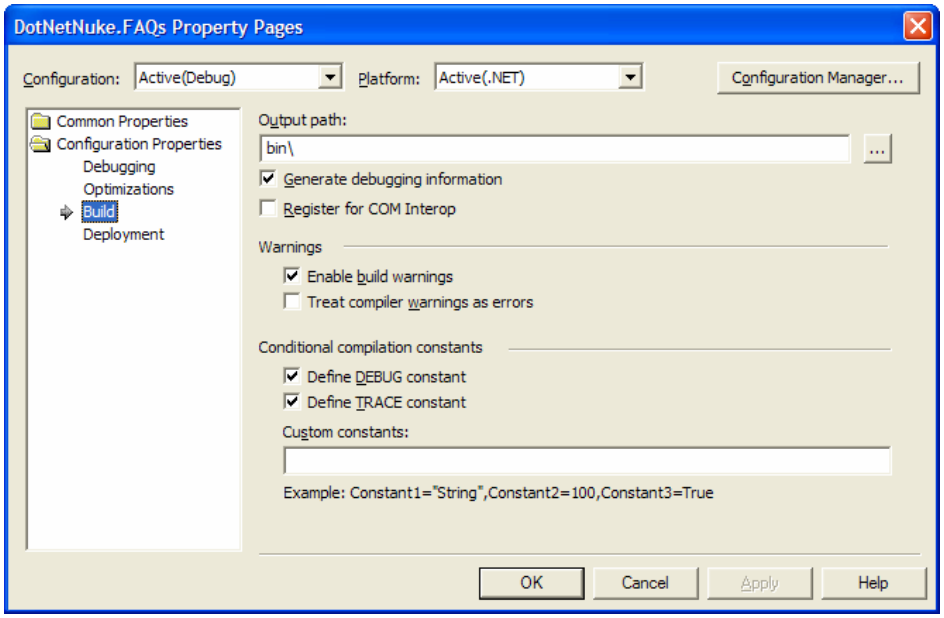


Figura 8 – Diálogo de Propiedades del Proyecto de clase

Ahora que has creado tu módulo crearemos algunos controles de usuario para la interfaz de tu módulo. Un módulo simple puede ser tan básico como un control de vista para visualizar una interfaz para tu usuario, y un control de configuraciones para especificar valores únicos específicos para un ejemplo de módulo.
Esta es una de las áreas de mayor dificultad a la hora de desarrollar proyectos de módulo para el DNN. Ya que queremos poder ver los resultados de nuestro desarrollo dentro del portal, el método para conseguirlo es utilizar un proyecto de clase que te permita cambiar en Visual Studio.NET la trayectoria de la compilación. Desde que trabajamos dentro de un proyecto de clase la capacidad de crear controles de usuario en Visual Studio.NET está limitada y sólo está disponible cuando estás en un proyecto Web. En este documento, el autor ha incluido el Consejo más abajo para indicarte la plantilla que te hará la creación de controles más fácil. Para alcanzar los propósitos de esta guia, crearemos manualmente un item y añadiremos nuestro código.
En nuestro ejemplo trabajamos con el módulo Survey que está incluido en la distribución del DNN. Este módulo está ubicado dentro del directorio /DesktopModules/Survey incluido en la distribución.
Se pueden ver algunos controles, además de algunos archivos de clases dentro de esta carpeta. Al principio, al empezar un proyecto, necesitarás crear algunos controles de usuario para que el usuario interactúe con tu módulo, y para que tus administradores configuren algunas configuraciones de ejemplo del módulo.

Consejo: Facilita la Creación del Archivo del DNN

En esta guia tratamos la creación de un proyecto de módulo desde cero; puedes hacer más eficiente este proceso de desarrollo utilizando recursos disponibles en la Web. Éstos están dirigidos inicialmente al DNN 2.x, pero deberían funcionar también con el DNN 3.x. Inténtalo utilizando las Plantillas de Proyecto del DNN disponibles en http://dnnjungle.vmasanas.net/Default.aspx?tabid=28 para crear tus proyectos de módulo, y controles.

Crear el Proyecto de Proveedor de Datos

El proyecto de proveedor de Datos sirve para desarrollar los métodos para interconectar con la base de datos física. En esta guia hablamos del SQL Server 2000 como nuestro proveedor de base de datos física, pero tu proveedor real podría ser en particular cualquier base de datos como Access, Oracle, o incluso MySQL. Una de las grandes ventajas de la arquitectura del DNN es un esquema del modelo de proveedor, que te permite conectarte a cualquier proveedor que quieras para el proveedor de base de datos física. Más adelante en esta guia hablaremos del modelo de proveedor.

En el desarrollo del módulo tú creas un proyecto separado para cada proveedor de datos física. Esto se hace de manera similar a la forma en la que configuraste tu proyecto de módulo.

1. Crea un proyecto de clase nuevo, asegurándote de que utilices una estructura de nomenclatura lógica. Por ejemplo: CompanyName.ModuleName.SQLDataProvider. De nuevo, esto evitará conflictos de nomenclatura con otros módulos, y te mantiene dentro de la estructura de nomenclatura del módulo que este proveedor va a admitir.
2. Cuando crees una carpeta de proyecto nueva, crea una carpeta que no esté en el directorio del DotNetNuke Root\DesktopModules\ModuleName\, llama a este directorio “Providers”, y crea un subdirectorio para cada tipo de proveedor que desarrolles, por ejemplo:
   <app root>\DesktopModules\ModuleName\Providers\SQLDataProvider\
3. Borra cualquier espacio en el nombre desde las propiedades del proyecto. Haz click con el botón derecho en las propiedades del proyecto, bajo general despeja la casilla de la raíz del espacio de nombre.
4. Ahora añade referencias del proyecto a los proyectos principales del DotNetNuke y de CompanyName.ModuleName.
5. Añade una referencia al proyecto BuildSupport como hicimos cuando elaboramos el módulo.

Una vez que ambos proyectos estén configurados, deberían ser similares a lo que aparece en la Figura 9.

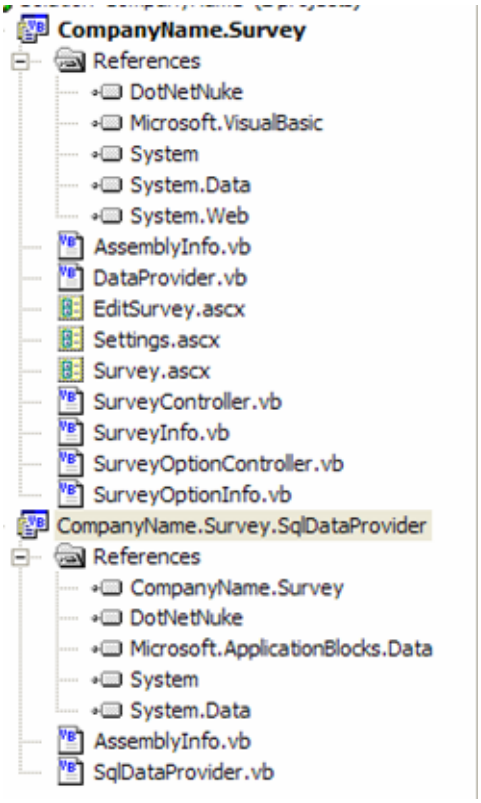


Figura 9 – Configuración de un Ejemplo de Módulo Survey en el DNN.

Configurar tu base de datos

Finalmente necesitaremos crear tablas y ya que estamos usando un Servidor SQL como base de datos, crearemos procedimientos almacenados para que admitan a nuestro módulo. Hay muchos métodos para crear objetos de base de datos, lo que está fuera del ámbito de esta guía. Proporcionaremos la estructura del módulo aquí, en la guia, y desde ahí podrás elaborar. Una manera fácil de construir la estructura es abrir el script del proveedor de la base de datos para el módulo, eliminar todas las ocurrencias que aparezcan de {databaseOwner}{objectQualifier} con el **notepad** u otro editor de texto simple, y entonces pasar el script resultante por el Query Analyzer.

Este módulo específico tiene la siguiente estructura de base de datos:

- Tabla: SurveyOptions
  - Campo: SurveyOptionID
  - Campo: SurveyID
  - Campo: ViewOrder
  - Campo: OptionName
  - Campo: Votes
- Tabla: Surveys
  - Campo: SurveyID
  - Campo: ModuleID
  - Campo: Question
  - Campo: ViewOrder

  - Campo: OptionType
  - Campo: CreatedByUser
  - Campo: CreatedDate
- Procedimiento almacenado: GetSurveys
- Procedimiento almacenado: GetSurvey
- Procedimiento almacenado: AddSurvey
- Procedimiento almacenado: UpdateSurvey
- Procedimiento almacenado: DeleteSurvey
- Procedimiento almacenado: GetSurveyOptions
- Procedimiento almacenado: AddSurveyOption
- Procedimiento almacenado: SurveyOptions
- Procedimiento almacenado: UpdateSurveyOption
- Procedimiento almacenado: DeleteSurveyOption
- Procedimiento almacenado: AddSurveyResult

Cuando lleguemos a la sección del proveedor de datos, verás que cada procedimiento almacenado tiene un método dentro de la clase de proveedor. Cuando creas módulos a medida (custom), la estructura de tu base de datos, y la cantidad de procedimientos almacenados existentes variaran dependiendo de las tareas que quieras realizar.

Consejo: Facilita la Creación del Procedimiento Almacenado en la Base de Datos y del Proveedor

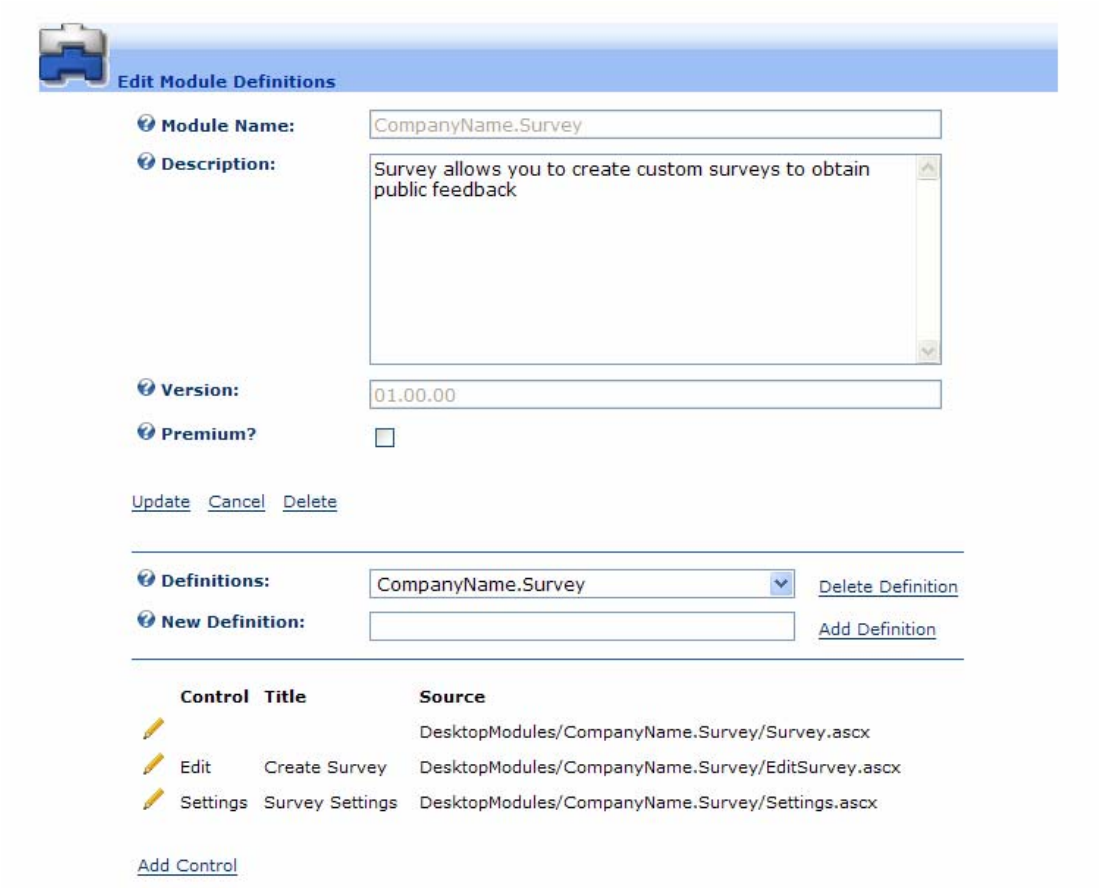
Puedes descargarte el Code Smith del autor Eric J. Smith,que te permitirá generar procedimientos almacenados, y el código para el DNN con las plantillas adecuadas. Descárgate el Code Smith en: http://www.ericjsmith.net/codesmith/. Scott McCulloch hace plantillas para el DNN disponibles en su Diario de Web: http://lucaslabs.net/blogs/mccullochs/archive/2004/02/15/487.aspx

Configurar el DNN para que se Interconecte con tus Controles

Ahora que ya tenemos nuestro proyecto definido, los controles creados, y la estructura de la base de datos hecha, tenemos que permitir a estos controles que se interconecten con nuestro portal del DNN. En nuestro ejemplo hemos creado tres controles, una vista, un editor y un control de configuración. Todos estos controles necesitan ser definidos dentro de las definiciones del módulo en nuestro portal. En los siguientes apartados configuraremos los controles de usuario para interconectarse con el DNN; esto nos permitirá ver al instante los resultados de una nueva compilación de nuestro módulo dentro de nuestra prueba del portal.
Deberías estar familiarizado con los archivos de instalación de Ensamblado Privado (Private Assembly), simplemente cargas un archivo zip al DNN y todo se instala automáticamente. Cuando estás desarrollando módulos normalmente tienes que hacer este proceso manualmente, y hacer que el DNN conozca dónde están tus controles de ascx. Tus unidades compiladas ya están compiladas y ubicadas en la carpeta bin del DNN así el DNN puede acceder a ellas cuando lo necesite. Pero aún necesitas indicar al DNN toda la información sobre tu Nuevo módulo. Puedes definir manualmente tus módulos siguiendo el procedimiento siguiente:

1. En el menú de Host Settings, ve a las Definiciones del Módulo “Module Definitions”
2. En las Definiciones del Módulo Selecciona Añadir Definición Nueva “Add new definition” desde el menú del módulo en la esquina izquierda de arriba.
3. Entonces te aparecerá la pantalla de Editar Definiciones de Módulo. “Edit Module”

4. Desde aquí añade una definición nueva.
5. Una vez que se ha creado tu definición añade tus controles de usuario a tu módulo. Para el módulo survey necesitarás añadir lo siguiente:
    - Survey.ascx - Type of View
    - EditSurvey.ascx - Type Edit, Key is Edit.
    - Settings.ascx - Type is Edit, Key is Settings.



Figur 10 – Definiciones de Módulo, Módulo Survey Definido

Ahora que has terminado de añadir las referencias a tu módulo, deberías crear una pantalla similar a la de la Figura 10, que muestra todos los controles que componen tu módulo, y las claves definidas.

Arquitectura del Módulo

Vamos a ver cómo los módulos están relacionados con el DNN. En la Figura 11, se puede ver que en la parte más alta hay un **tab**. Para un usuario, esta es la página que aparece al hacer click a través de la navegación. Cada tab contiene varios módulos de escritorio. Un módulo de escritorio se puede definir de varias maneras como hemos visto en ejemplos anteriores donde se definía un módulo a través de la interfaz del host utilizando definiciones del módulo. Dentro de esta definición de módulo puedes definir varios módulos que contengan varios tipos diferentes de controles del módulo. Los controles del módulo son

básicamente controles de usuario dentro de ASP.NET, pero nosotros utilizamos una clave para definir el tipo de control, estos controles pueden ser de Admin, Edit, or View dentro de un módulo de escritorio.

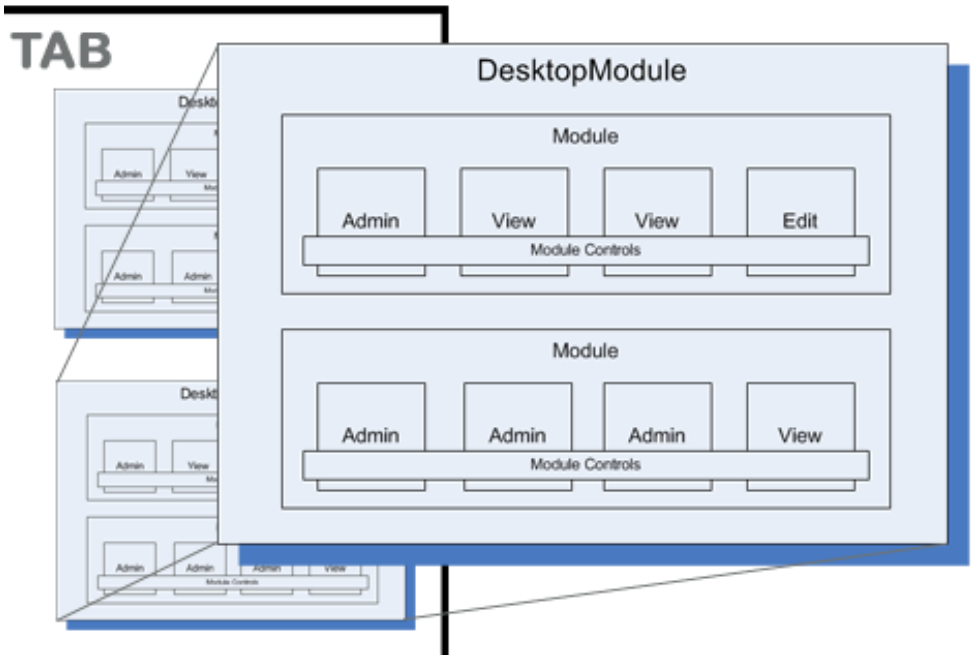


Figura 11 – Arquitectura del Módulo Relacionada con el DNN

Ahora que hemos tratado la abstracción de mayor nivel de cómo es la arquitectura de los módulos dentro del marco del DNN, empecemos escribiendo algún código real, y después construyendo un archivo de definición para nuestro módulo. Los siguientes apartados deberían proporcionarte información para que empieces a desarrollar tus propios proyectos de módulo para el DNN.

Arquitectura de Tres Capas

Antes de entrar en las diferentes capas del desarrollo de módulos del DNN, primero deberíamos tratar algunos conceptos arquitectónicos básicos del desarrollo de la aplicación.

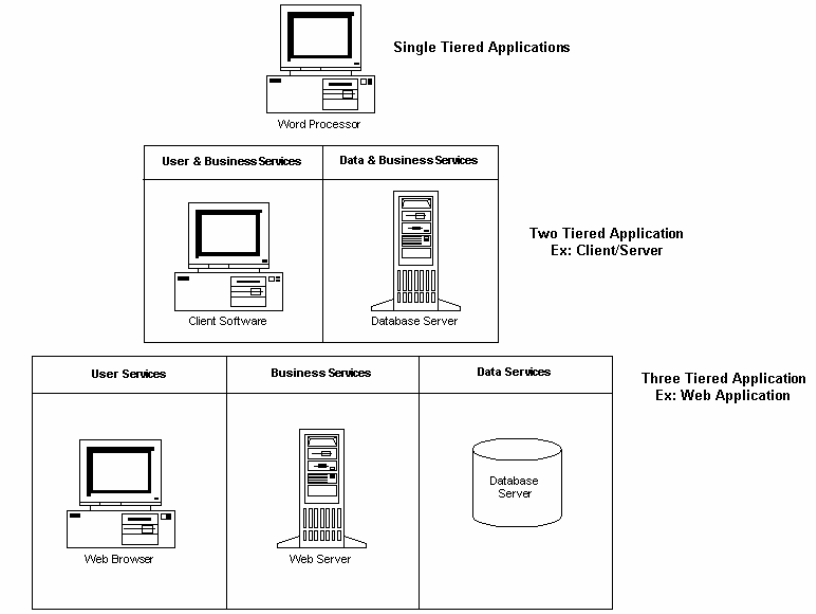


Figura 12 – Revisión de la Arquitectura de Tres Capas de la Aplicación

En la parte superior del dibujo (ver Figura 12) hay una sola capa de aplicación del procesador de textos que contiene todos los niveles lógicos de servicio— los archivos con los que se interconecta; los servicios comerciales— la lógica utilizada para definir cómo interactúa la aplicación con los archivos de documentos; y los servicios para el usuario—la GUI con la que el usuario interactúa para editar los documentos. Todos estos servicios están en una capa. No tiene lugar ninguna distribución de la aplicación.
En el nivel medio hay una aplicación cliente/servidor típica. Esta aplicación tiene los servicios para el usuario en una capa que es la GUI con la que interactúa el usuario. Este nivel puede tener también algunos servicios comerciales, donde reside una lógica que hace cumplir las normas comerciales. Este nivel comercial puede residir también en una segunda capa, estando la bases de datos en un servidor secundario que contiene funciones, o procedimientos almacenados. Nosotros seguimos teniendo tres capas de servicio, pero la aplicación ahora está dividida en dos capas físicas distintas.
Al final de la ilustración, está el modelo de aplicación de la Web, donde se define claramente cada capa con cada capa de servicio. Distribuimos la aplicación ubicando los servicios de datos en un Servidor SQL u otro proveedor de datos, los servicios comerciales en un objeto COM, y los servicios para el usuario en el código HTML o ASP que está distribuido para el usuario final a través del navegador de web.
Podemos expandir más este modelo de aplicación a un **nivel de n-capas**, que significa que los servicios comerciales se podrían dividir en múltiples objetos COM, o que podríamos tener muchos proveedores de datos diferentes que interactúen con la aplicación. Cada nivel adicional proporcionaría otra capa a la aplicación.

Una Página, Muchos Controles

Al usuario del DNN se le dirige desde un documento primario que es el archivo default.aspx contenido en el directorio raíz de la aplicación. Todas las peticiones de los usuarios se hacen a este documento y cambia su comportamiento según ciertos parámetros que se le pasan. Por ejemplo, deberías tener múltiples controles para tu módulo que se cargan al archivo

principal default.aspx. Recuerda que el proceso de carga del módulo se define a través de un **Tab id** específico para la página. Cada control que crees debe estar definido dentro del DNN (ver "Configurar el DNN para interconectarse con tus Controles" más adelante en esta guía) con un valor clave que designa el control a cargar.
Las páginas se cargan basadas en un Tab ID que se pasa dentro de un parámetro de peticiones a la aplicación del DNN. El Tab ID es un valor único dentro de la tabla de Tabs en una base de datos. Hay muchos otros parámetros importantes que deberías tener en cuenta a la hora de desarrollar fuera del marco del DNN. Lo siguiente es una lista de parámetros y lo que hacen:

- **TabId** – Las páginas dentro del DNN están designadas por un TabID. Por ejemplo, si creas una página en tu portal, a ésta se le asigna una clave primaria con el nombre de TabID dentro de la base de datos. Esta clave pasa ahora a asociarse con los módulos que insertes en la página y el comportamiento.
- **Mid** – Este parámetro está para especificar el Módulo ID de un módulo específico. Esto ocurre normalmente cuando quieres cambiar las configuraciones dentro de un módulo, o necesitas cargar un control para el usuario específico en tu proceso.
- **Ctl** – Este es el control de cargar para tu módulo específico. Por ejemplo, tu módulo debería consistir en múltiples controles para el usuario, mientras que éstos están definidos en las definiciones del módulo del portal, entonces puedes pasar un valor clave del control al parámetro Ctl para decirle al módulo que cargue ese control específico. Este parámetro se debe utilizar conjuntamente con el parámetro Módulo ID (mid) para designar cuál de los ejemplos de módulo está relacionado con el control especificado.

Por ejemplo, digamos que tengo un módulo que contiene múltiples controles para el usuario, cada control estaría entonces definido dentro de la definición del módulo con su propia clave única; entonces si quisiera cargar un control específico dentro de un módulo haría una petición al tab que contenga el módulo pero que incluya un control para cargar parámetros :

```
Response.Redirect(NavigateURL & "mid=" & ModuleID & "&ctl=ControlKey")
```

El bloque de código anterior tendría el efecto de cargar el módulo actual, pero con el control específico especificado al pasar su valor del nombre clave a través del parámetro Ctl. Mira el apartado de controles para el usuario del DNN para más información sobre cómo trabajan los controles con tus módulos.

Capa de Datos

En esta sección trataremos nuestra clase abstracta para interactuar con la base de datos. Desde que nuestro módulo utiliza un modelo de proveedor, nuestro módulo sólo contiene clases abstractas para interactuar con bases de datos. Uno de los principales beneficios de este modelo es la capacidad de cambiar diferentes proveedores en nuestra aplicación sin tener que modificar la base del código del módulo primario. Toda la interacción física está hecha dentro de otro ensamblado, y nuestro **proveedor abstracto** en nuestro módulo sólo va a contener métodos anulables para hacer operaciones con la base de datos.

Proveedor Abstracto de Datos

La clase DataProvider.vb ubicada dentro del proyecto de módulo es la clase que contiene los métodos **eliminables** para nuestro módulo. Cuando abres la clase DataProvider.vb dentro

del proyecto Survey, ves que contiene métodos que tienen que ser anulados por nuestro ensamblado del proveedor de base de datos física. Cada uno de los métodos en este ejemplo corresponde con uno de los procedimientos almacenados dentro de nuestra base de datos.

```
Imports System
Imports DotNetNuke
Namespace CompanyName.Survey
Public MustInherit Class DataProvider
        ' singleton reference to the instantiated object
        Private Shared objProvider As DataProvider = Nothing

        ' constructor
        Shared Sub New()
            CreateProvider()
        End Sub

        ' dynamically create provider
        Private Shared Sub CreateProvider()
                        objProvider = CType(Framework.Reflection.CreateObject("data",
"CompanyName.Survey", "CompanyName.Survey"), DataProvider)
        End Sub

        ' return the provider
        Public Shared Shadows Function Instance() As DataProvider
            Return objProvider
        End Function

        ' all core methods defined below

        Public MustOverride Function GetSurveys(ByVal ModuleId As Integer) As IDataReader
        Public MustOverride Function GetSurvey(ByVal SurveyID As Integer, ByVal ModuleId
As Integer) As IDataReader
        Public MustOverride Function AddSurvey(ByVal ModuleId As Integer, ByVal Question
As String, ByVal ViewOrder As Integer, ByVal OptionType As String, ByVal UserName As
String) As Integer
        Public MustOverride Sub UpdateSurvey(ByVal SurveyId As Integer, ByVal Question As
String, ByVal ViewOrder As Integer, ByVal OptionType As String, ByVal UserName As
String)
        Public MustOverride Sub DeleteSurvey(ByVal SurveyID As Integer)
        Public MustOverride Function GetSurveyOptions(ByVal SurveyId As Integer) As
IDataReader
        Public MustOverride Function AddSurveyOption(ByVal SurveyId As Integer, ByVal
OptionName As String, ByVal ViewOrder As Integer) As Integer
        Public MustOverride Sub UpdateSurveyOption(ByVal SurveyOptionId As Integer,
ByVal OptionName As String, ByVal ViewOrder As Integer)
        Public MustOverride Sub DeleteSurveyOption(ByVal SurveyOptionID As Integer)
        Public MustOverride Sub AddSurveyResult(ByVal SurveyOptionId As Integer)
End Class
End Namespace
```

Estos métodos los utiliza la capa de negocio de nuestro módulo para efectuar alguna acción en la base de datos, como añadir, eliminar, o actualizar operaciones, o para **llenar** un objeto de negocio.

Capa Lógica de Negocio (BLL)

Ya hablamos sobre la **población** de objetos de negocio; ahora trataremos un ejemplo de cómo hacer esto.
Un objeto es una clase de tipo específico que expone ciertas propiedades y acciones. En nuestro ejemplo tenemos dos clases en nuestra capa lógica de negocio (BLL), una es la clase información, y la otra es la clase controlador. En este ejemplo SurveyInfo, y SurveyController. Aquí tienes una descripción de estas dos clases:

- **ModuleInfo:** Asigna y obtiene propiedades públicas para nuestros objetos, que son generadas por nuestra clase controlador. Las propiedades expuestas por esta clase corresponden al nombre de los campos y a sus tipos dentro de la tabla de la base de datos que admite nuestro módulo. Por ejemplo, en el modulo Survey hay un SurveyID, dentro de la clase SurveyInfo, hay una propiedad con el mismo nombre.
- **ModuleController:** Se utiliza para obtener valores para los objetos, básicamente usa la clase ayudante de objetos de negocio a medida para tomar datos desde nuestro proveedor de datos a una recopilación de objetos.

Veamos el código fuente del archivo SurveyOptionsInfo.vb ubicado en el directorio del proyecto de módulo:

```
Imports System
Imports System.Data
Imports DotNetNuke

Namespace CompanyName.Survey

	Public Class SurveyOptionInfo

		' local property declarations
		Private _SurveyOptionId As Integer
		Private _SurveyId As Integer
		Private _ViewOrder As Integer
		Private _OptionName As String
		Private _Votes As Integer

		' initialization
		Public Sub New()
		End Sub

		' public properties
		Public Property SurveyOptionId() As Integer
			Get
				Return _SurveyOptionId
			End Get
			Set(ByVal Value As Integer)
```

```
                        _SurveyOptionId = Value
                End Set
        End Property

        Public Property SurveyId() As Integer
                Get
                        Return _SurveyId
                End Get
                Set(ByVal Value As Integer)
                        _SurveyId = Value
                End Set
        End Property

        Public Property ViewOrder() As Integer
                Get
                        Return _ViewOrder
                End Get
                Set(ByVal Value As Integer)
                        _ViewOrder = Value
                End Set
        End Property

        Public Property OptionName() As String
                Get
                        Return _OptionName
                End Get
                Set(ByVal Value As String)
                        _OptionName = Value
                End Set
        End Property

        Public Property Votes() As Integer
                Get
                        Return _Votes
                End Get
                Set(ByVal Value As Integer)
                        _Votes = Value
                End Set
        End Property

    End Class

End Namespace
```

Se puede ver que la clase expone propiedades que corresponden con la estructura de la tabla en la base de datos.
Ahora veámos la clase controlador para las Opciones del Survey dentro del archivo SurveyOptionController.vb.

```
Imports System
Imports System.Data
Imports DotNetNuke

Namespace CompanyName.Survey

	Public Class SurveyOptionController

		Public Function GetSurveyOptions(ByVal SurveyId As Integer) As
ArrayList

			Return
CBO.FillCollection(DataProvider.Instance().GetSurveyOptions(SurveyId),
GetType(SurveyOptionInfo))

		End Function

		Public Sub DeleteSurveyOption(ByVal SurveyOptionID As Integer)

			DataProvider.Instance().DeleteSurveyOption(SurveyOptionID)

		End Sub

		Public Function AddSurveyOption(ByVal objSurveyOption As
SurveyOptionInfo) As Integer

			Return
CType(DataProvider.Instance().AddSurveyOption(objSurveyOption.SurveyId,
objSurveyOption.OptionName, objSurveyOption.ViewOrder), Integer)

		End Function

		Public Sub UpdateSurveyOption(ByVal objSurveyOption As
SurveyOptionInfo)


	DataProvider.Instance().UpdateSurveyOption(objSurveyOption.SurveyOptionId,
objSurveyOption.OptionName, objSurveyOption.ViewOrder)

		End Sub

		Public Sub AddSurveyResult(ByVal SurveyOptionID As Integer)

			DataProvider.Instance().AddSurveyResult(SurveyOptionID)

		End Sub

	End Class
End Namespace
```

Cada método dentro de la clase controlador corresponde a un procedimiento almacenado dentro de la base de datos. Al revisar el código dentro de los métodos verás que crean una llamada al DataProvider.Instance que es una llamada a nuestra clase abstracta de la que se habló anteriormente. Estos métodos abstractos se pasan al objeto SurveyOptionInfo para mantener los valores utilizados para pasar los procedimientos almacenados definidos en nuestra clase proveedor de base de datos física. En el apartado Capa del Usuario te mostraremos cómo llevar a cabo una llamada a estas clases y a llenar los valores para los objetos.

Si estás devolviendo datos al objeto SurveyOptionInfo para luego unirlos en un control dentro de la interfaz del usuario, la clave ayudante Objeto de Negocio a Medida (CBO) proporciona la instanciación del objeto. El CBO es parte del marco central del DNN y proporciona la hidratación de los objetos cuando hay una devolución desde una llamada de la base de datos, como en este caso una llamada al procedimiento almacenado.

Capa del Usuario

En la arquitectura de tres capas, el nivel del usuario es dónde el usuario interactúa con tu aplicación. En el DNN esta capa está compuesta por archivos (ascx) de los controles para el usuario. En los siguientes apartados trataremos las diversas interconexiones con el DNN. Esto incluye el control de vista, con el interactuarán tus usuarios, y las interfaces administrativa y de edición de tu módulo.

Visualizar Datos desde la Perspectiva del Usuario

El control vista es el control inicial que el usuario verá cuando acceda a una página que contenga el módulo. En nuestro ejemplo, puedes ver el Survey.ascx. Éste contiene todos los controles que necesitamos para nuestra vista inicial del survey.
El control vista para este módulo, este es el control principal Survey.ascx, abre el code behind (código detrás del archivo). En esta clase veras lo siguiente:

```
Imports DotNetNuke
Namespace CompanyName.Survey
   Public MustInherit Class Survey
      Inherits DotNetNuke.Entities.Modules.PortalModuleBase
      Implements Entities.Modules.IActionable

       ::
       ::
End Class
```

Base del Módulo del Portal

Te darás cuenta de que en lugar de heredar desde la clase WebControls como haría un control para el usuario estándar, el módulo hereda desde DotNetNuke.Entities.Modules.PortalModuleBase proporcionado por el DNN. La clase PortalModuleBase, ubicada en <app root>\Components\Modules\PortalModuleBase.vb, proporciona algunas propiedades y métodos muy importantes expuestos por el DNN, por ejemplo:

- **IsEditable**: El valor booleano determina si el usuario actual ha editado permisos para el módulo o no.
- **LocalResourceFile** – Si estás proporcionando un recurso de ubicación para múltiples lenguajes.
- **SharedResourceFile**
- **ModuleSettings –** Las Configuraciones del Módulo junto con la tablas de identificación proporcionan al desarrollador del módulo la capacidad de almacenar valores de ejemplo de módulo.
- **Settings** – La tabla de numerar para almacenar claves y valores específicos para el módulo. Esto te permite almacenar las propiedades que definas que estén descodificadas del ModuleID para el ejemplo de módulo actual, permitiéndote ejecutar la misma base del código del módulo para varias ejecuciones.
- **HelpFile** – Para almacenar un valor de archivo de ayuda para tu módulo.
- **HelpURL** – Éste almacena un valor para una versión online de ayuda almacenada en un sitio Web.
- **Actions** – Para almacenar apartados del menú y sus acciones asociadas para tu módulo.
- **CachedOutput** – Para almacenar la **salida en caché** desde el módulo. Esta configuración de la cache está especificada en el nivel del módulo, y en niveles host.

En este ejemplo específico, tenemos alguna interacción básica con la base de datos a través del BLL. Recuerda que el código para este módulo está disponible en la descarga del DotNetNuke como ejemplo de desarrollo de módulo a medida. Utiliza esto como plantilla para tus propios proyectos.

Interfaz IActionable

Otra área sobre la que hablar es la implementación de la interfaz IActionable. Aquí vamos a exponer ciertas acciones del módulo en nuestro control del menú. En el ejemplo Survey puede verse la siguiente subrutina, que expone acciones específicas para nuestro módulo.

```
Public ReadOnly Property ModuleActions() As
DotNetNuke.Entities.Modules.Actions.ModuleActionCollection Implements
DotNetNuke.Entities.Modules.IActionable.ModuleActions
        Get
                Dim Actions As New Entities.Modules.Actions.ModuleActionCollection
Actions.Add(GetNextActionID,
Localization.GetString(Entities.Modules.Actions.ModuleActionType.AddContent,
LocalResourceFile), Entities.Modules.Actions.ModuleActionType.AddContent, "", "",
EditUrl(), False, Security.SecurityAccessLevel.Edit, True, False)
                Return Actions
        End Get
End Property
```

En este código, se exponen acciones del menú como propiedad del módulo, del tipo ModuleActions. Aquí añadiremos algunas acciones que se trasladarán al módulo contenedor para generar nuestra estructura del menú. Hay acciones específicas para nuestro módulo como añadir una opción de editar a los items del menú. Si miras en el método Actions.Add verás que estamos proporcionando contenido ubicado si está disponible, el URL del control de editar, estamos comprobando la seguridad para asegurar que el item del menú sólo lo visualicen los usuarios del portal que tienen permisos para editar.

Visualización de Datos

Una de las tareas fundamentales del control de vista es visualizar datos desde nuestra base de datos. En el ejemplo survey hay dos controles en el archivo Survey.ascx que están delimitados por el objeto SurveyInfo a la hora de visualizar las cuestiones en el survey y los resultados del survey.

```
<%@ Control Language="vb" AutoEventWireup="false" Inherits="CompanyName.Survey.Survey"
CodeBehind="Survey.ascx.vb" %>
<br>
<asp:panel id="pnlSurvey" runat="server" visible="False">
	<asp:datalist id="lstSurvey" datakeyfield="SurveyId" cellpadding="4" runat="server">
		<itemtemplate>
			<asp:HyperLink id="cmdEdit1" ImageUrl="~/images/edit.gif"
NavigateUrl='<%# EditURL("SurveyId",DataBinder.Eval(Container.DataItem,"SurveyId")) %>'
Visible="<%# IsEditable %>" runat="server" />
			<asp:Label ID="lblQuestion1" Runat="server" CssClass="NormalBold"
Text='<%#
FormatQuestion(DataBinder.Eval(Container.DataItem,"Question"),lstSurvey.Items.Count + 1)
%>'>
			</asp:label><br>
			<asp:radiobuttonlist id="optOptions" runat="server" cssclass="Normal"
visible="False" repeatdirection="Vertical" datavaluefield="SurveyOptionId"
datatextfield="OptionName"></asp:radiobuttonlist>
			<asp:checkboxlist id="chkOptions" runat="server" cssclass="Normal"
visible="False" repeatdirection="Vertical" datavaluefield="SurveyOptionId"
datatextfield="OptionName"></asp:checkboxlist>
			<br>
		</itemtemplate>
	</asp:datalist>
	<asp:linkbutton id="cmdSubmit" resourcekey="cmdSubmit" runat="server"
cssclass="CommandButton">Submit Survey</asp:linkbutton> 
	<asp:linkbutton id="cmdResults" resourcekey="cmdResults" runat="server"
cssclass="CommandButton">View Results</asp:linkbutton>
</asp:panel>
<asp:panel id="pnlResults" runat="server" visible="False">
	<asp:datalist id="lstResults" datakeyfield="SurveyId" cellpadding="4" runat="server">
		<itemtemplate>
			<asp:HyperLink id="cmdEdit2" ImageUrl="~/images/edit.gif"
NavigateUrl='<%# EditURL("SurveyId",DataBinder.Eval(Container.DataItem,"SurveyId")) %>'
Visible="<%# IsEditable %>" runat="server" />
			<asp:Label ID="lblQuestion2" Runat="server" CssClass="NormalBold"
```

```
Text='<%# FormatQuestion(DataBinder.Eval(Container.DataItem,"Question"),
lstResults.Items.Count + 1) %>'>
                    </asp:label><br>
                    <asp:label id="lblResults" runat="server" cssclass="Normal"></asp:label>
                    <br>
                </itemtemplate>
        </asp:datalist>
        <asp:linkbutton id="cmdSurvey" resourcekey="cmdSurvey" runat="server"
cssclass="CommandButton">View Survey</asp:linkbutton>
</asp:panel>
```

Veámos el control lstSurvey DataList, y como delimitamos nuestros datos a este control cuando el control está cargado en primera instancia.

```
Private Sub DisplaySurvey()

        Dim objSurveys As New SurveyController

        lstSurvey.DataSource = objSurveys.GetSurveys(ModuleId)
        lstSurvey.DataBind()

        pnlSurvey.Visible = True
        pnlResults.Visible = False

        If lstSurvey.Items.Count = 0 Then
                cmdSubmit.Visible = False
                cmdResults.Visible = False
        End If

End Sub
```

Puede verse en el código de arriba, que hemos creado un ejemplo de la clase SurveyController, entonces pasamos el valor del ModuleID. Recuerda que el ModuleID es nuestra clave para el ejemplo de módulo, el valor dentro del ModuleID nos permite reutilizar este componente tantas veces como queramos dentro de la implementación de nuestro portal del DNN. Es muy importante que cuando diseñes la estructura de tu base de datos incluyas un campo para que mantenga el valor del ModuleID. El ModuleID se pasa al procedimiento almacenado a través de nuestra clase abstracta, y luego a la clase del proveedor físico, y a su vez devuelve nuestros datos, que llenan nuestro objeto SurveryController. Éste es de un tipo ArrayList que está delimitado por el control DataList, y finalmente llenan la lista que presentamos al usuario.

Añadir Datos

Ahora hagamos un ejemplo de actualización de datos para nuestro módulo. Esto se hace creando un ejemplo de nuestro objeto SurveyOptionController, llamando al método AddSurveyResults, y pasando nuestros valores, que a su vez se ejecutan a través de nuestra clase abstracta como se vio anteriormente. Veamos nuestra clase Survey.ascx.vb en el método cmdSubmit_Click.

```
::
::
Dim objSurveyOptions As New SurveyOptionController
::
::
If Request.Form(lstSurvey.UniqueID & ":_ctl" & intQuestion.ToString & ":optOptions")
<> "" Then

        objSurveyOptions.AddSurveyResult(Int32.Parse(Request.Form(lstSurvey.UniqueID
& ":_ctl" & intQuestion.ToString & ":optOptions")))
End If
::
::
```

Actualizar Datos

Ahora actualicemos datos desde nuestro survey. Para actualizar nuestros datos necesitamos crear un ejemplo de la clase Info para el objeto que queramos actualizar. Pasaremos el ID del objeto fijado, y pasaremos los valores actualizados a las propiedades del objeto. Este objeto se pasará entonces a la clase controlador, que pasará el método a la clase abstracta y a su vez el proveedor físico ejecutará el procedimiento almacenado correspondiente. Ahora veamos un ejemplo de datos actualizados, abre el archivo EditSurvey.ascx.vb, y busca el método Update_Clic. Dentro del método Update_Clic verás el siguiente código:

```
Dim objSurveyOptions As New SurveyOptionController
::
Dim objSurveyOption As New SurveyOptionInfo

objSurveyOption.SurveyOptionId = Integer.Parse(lstOptions.Items(intOption).Value)
objSurveyOption.SurveyId = SurveyId
objSurveyOption.OptionName = lstOptions.Items(intOption).Text
objSurveyOption.ViewOrder = intOption
objSurveyOption.Votes = 0

If Null.IsNull(objSurveyOption.SurveyOptionId) Then
      objSurveyOptions.AddSurveyOption(objSurveyOption)
Else
      objSurveyOptions.UpdateSurveyOption(objSurveyOption)
End If
```

En este bloque de código creamos un ejemplo para nuestro objeto SurveyOptionInfo, y entonces pasamos los valores a actualizar al objeto. También verás aquí que comprobamos si el valor del SurveyID, que es nuestra clave principal para la tabla Survey, es nulo. Si es nulo, entonces significa que tenemos un nuevo item que añadir y que llamar al método AddSurveyOption. Si no lo es, entonces llamamos a la UpdateSurveyOption y pasamos nuestro objeto SurveyOptionInfo con los valores establecidos. El controlador llamará a la clase abstracta, que a su vez llama a la clase de proveedor de base de datos física, y finalmente ejecutará el procedimiento almacenado con los valores actualizados.

Eliminar Datos

Ahora que ya hemos tratado añadir, visualizar y actualizar, todavía necesitamos saber cómo se eliminan datos de las tablas. La forma de eliminar es muy parecida a la de añadir y actualizar, pero en ésta el único valor que hay que pasar al método es la clave principal del item que se quiera eliminar. En este ejemplo, hacemos una llamada a nuestra clase SurveyOptionController y llamamos al método DeleteSurveyOption. Para ver este código en el ejemplo encuentra el método cmdDeleteOption_Click en el archivo EditSurvey.ascx.vb.

```
Dim objSurveyOptions As New SurveyOptionController
If lstOptions.SelectedIndex <> -1 Then
      objSurveyOptions.DeleteSurveyOption(Integer.Parse(lstOptions.SelectedItem.Value))
      lstOptions.Items.RemoveAt(lstOptions.SelectedIndex)
End If
```

Editar Settings para Tu Módulo

Tratamos añadir, actualizar y eliminar datos de tu módulo, pero también hemos de hablar sobre el item que tiene la capacidad de crear una implementación distinta del mismo módulo múltiples veces a través de uno o múltiples portales. Recuerda que anteriormente en esta guía hemos dicho que a cada módulo se le asigna un único Module ID, y que este ID se utiliza para descodificar en nuestros módulos. Por ejemplo, digamos que tienes un módulo de facturación, sin embargo necesitas indicar diferentes sistemas de facturación para las implementaciones de dos módulos. Podrías crear un módulo de clave y valor específicos dentro del DNN para almacenar los valores de la conexión para cada sistema de facturación.

DNN contiene una tabla dentro de la base de datos llamada ModuleSettings que mantiene nombres clave y sus valores para cada ejemplo de módulo. Examinemos la estructura de la base de datos:

- **ModuleID** – Contiene el ID del ejemplo de módulo para el que estamos almacenando información.
- **SettingName** – Contiene un valor de cadena para almacenar el nombre clave para el que vamos a guardar datos.
- **SettingValue** – Éste almacena el valor real para la clave específica para este ejemplo específico de módulo.

Estos valores se activan haciendo una llamada al DotNetNuke.Entities.Modules.ModuleSettingsBase, revisemos el archivo Settings.ascx.vb en el proyecto de módulo Survey.

Si revisas el código en el archivo Settings.ascx.vb verás los dos métodos fundamentales que anulamos en nuestra clase base, éstos son los métodos LoadSettings y UpdateSettings. Éstos son métodos llamados para obtener los valores desde nuestra tabla para identificar las configuraciones, y posteriormente guardar cualquiera de las configuraciones que hayan entrado a través de cualquier sistema. En este caso estaremos guardando valores que afectan a la interfaz visual de nuestro sitio.

Revisemos la línea de código para guardar una configuración en una clave:

```
Public Overrides Sub UpdateSettings()
        Try
                Dim objModules As New DotNetNuke.Entities.Modules.ModuleController

                Dim datClosingDate As Date =
Convert.ToDateTime(Null.SetNull(datClosingDate))
                If txtClosingDate.Text <> "" Then
                        datClosingDate = Convert.ToDateTime(txtClosingDate.Text)
                End If

                'Update Module Settings
                objModules.UpdateModuleSetting(ModuleId, "surveyclosingdate",
DotNetNuke.Common.Globals.DateToString(datClosingDate))
                objModules.UpdateModuleSetting(ModuleId, "surveytracking",
(rblstPersonal.SelectedIndex).ToString)
                objModules.UpdateModuleSetting(ModuleId, "surveyresultstype",
(rblstSurveyResults.SelectedIndex).ToString)

                'Update Tab Module Settings
                objModules.UpdateTabModuleSetting(TabModuleId, "surveygraphwidth",
txtGraphWidth.Text)

        Catch exc As Exception                      'Module failed to load
                ProcessModuleLoadException(Me, exc)
        End Try
End Sub
```

Si miras el código anterior podrás ver que hemos creado un ejemplo de clase ModuleController, y llamado al método UpdateModuleSetting. El UpdateModuleSetting acepta el ID del módulo actual (ModuleID), un valor de cadena para el nombre de la clave que vayamos a guardar y el valor para la clave.
Ahora veámos cómo se obtienen valores desde claves previamente guardadas.

```
Public Overrides Sub LoadSettings()
        Try
                If (Page.IsPostBack = False) Then
                        cmdCalendar.NavigateUrl =
DotNetNuke.Common.Utilities.Calendar.InvokePopupCal(txtClosingDate)

                If Not CType(Settings("surveyclosingdate"), String) = "" Then
                                        txtClosingDate.Text =
CType(Settings("surveyclosingdate"), Date).ToShortDateString
                End If
                txtGraphWidth.Text = CType(Settings("surveygraphwidth"), String)
                rblstPersonal.SelectedIndex = CType(Settings("surveytracking"), Integer)
                rblstSurveyResults.SelectedIndex = CType(Settings("surveyresultstype"),
```

```
Integer)
                End If
        Catch exc As Exception                  'Module failed to load
                ProcessModuleLoadException(Me, exc)
        End Try
End Sub
```

En el código podemos ver que lo único que necesitamos hacer es obtener el valor dentro del número de configuración del módulo y pasarlo a nuestro control.

Proveedores de Base de Datos Física

Hemos revisado la estructura del DNN, y cómo especificamos un proveedor a través del web.config. En este apartado revisaremos el código requerido para crear un proveedor de base de datos de SQL. Recuerda que cada tipo de base de datos con el que conectes va a necesitar su propio proveedor específico.
Para este apartado, cambia el proyecto CompanyName.ModuleName.SQLDataProvider por el módulo Survey.
Comparado con el proyecto de Módulo padre, éste en cuanto a estructura es relativamente simple. Este proyecto contiene las clases requeridas para interconectarse con tu base de datos física. En nuestro ejemplo Survey, utilizaremos el proyecto SQLDataProvider para conectar el Servidor SQL. En muchos de los casos esta clase de proveedor incluirá uno a uno los métodos para cada procedimiento almacenado en tu base de datos para tu proyecto específico de módulo. Estos métodos realizan la interacción real con la base de datos y los métodos existentes dentro de tu proveedor abstracto para interactuar con tu módulo. Éste es uno de los beneficios principales del modelo de proveedor y uno de los que deberías entender llegados a este punto, puedes conectarte fácilmente a varios proveedores sin tener que recompilar o modificar de ninguna manera tu proyecto principal.
Para establecer nuestra clase necesitamos importar algunos documentos:

```
Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports Microsoft.ApplicationBlocks.Data
Imports DotNetNuke
```

El Microsoft.ApplicationBlocks.Data nos va a propoorcionar algunos métodos para facilitar nuestra conexión con la base de datos SQL secundaria. Se puede encontrar más información sobre eseta clase en: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daab-rm.asp

En nuestra clase de proveedor de datos heredaremos la clase DataProvider. La clase está principalmente construida de forma que refleje la clase de proveedor de datos en la solución principal del DNN. Primero hemos de declarar algunos miembros privados para nuestra clase.

```
Namespace CompanyName.Survey
    Public Class SqlDataProvider
        Inherits DataProvider
#Region "Private Members"
                Private Const ProviderType As String = "data"
                Private _providerConfiguration As Framework.Providers.ProviderConfiguration
= Framework.Providers.ProviderConfiguration.GetProviderConfiguration(ProviderType)
                Private _connectionString As String
                Private _providerPath As String
                Private _objectQualifier As String
                Private _databaseOwner As String
```

Puede verse que el primer miembro privado es una constante con el valor de "data", no hay que volver al web.config que listamos previamente en esta guía, el valor de datos señala el valor dentro del web.config. De aquí es de donde nuestro proveedor va a sacar los datos para la cadena de conexión, la ruta del proveedor, el calificador del objeto, y el propietario de la base de datos. Todos estos valores están definidos en nuestra clase de proveedor, los que se pueblan una vez que es creada la clase. Veámos el "nuevo" método para ver qué ocurre cuando el proveedor está instanciado.

```
Public Sub New()

        ' Read the configuration specific information for this provider
        Dim objProvider As Framework.Providers.Provider =
CType(_providerConfiguration.Providers(_providerConfiguration.DefaultProvider), Framework.Providers.Provider)

        ' Read the attributes for this provider
        If objProvider.Attributes("connectionStringName") <> "" AndAlso _
        System.Configuration.ConfigurationSettings.AppSettings(objProvider.Attributes("connectionStringName"))
<> "" Then
                _connectionString =
System.Configuration.ConfigurationSettings.AppSettings(objProvider.Attributes("connectionStringName"))
        Else
                _connectionString = objProvider.Attributes("connectionString")
        End If

        _providerPath = objProvider.Attributes("providerPath")

        _objectQualifier = objProvider.Attributes("objectQualifier")
        If _objectQualifier <> "" And _objectQualifier.EndsWith("_") = False Then
                _objectQualifier += "_"
        End If

        _databaseOwner = objProvider.Attributes("databaseOwner")
        If _databaseOwner <> "" And _databaseOwner.EndsWith(".") = False Then
                _databaseOwner += "."
        End If

End Sub
```

Se puede ver cuándo esta clase está instanciada; se lee la información sobre el proveedor desde el web.config. Revisemos el apartado de datos desde el web.config para ver exáctamente de qué estamos hablando.

```
<data defaultProvider="SqlDataProvider">
        <providers>
                <clear />
                <add name="SqlDataProvider"
                type="DotNetNuke.Data.SqlDataProvider, DotNetNuke.SqlDataProvider"
                connectionStringName="SiteSqlServer"
                upgradeConnectionString=""
                providerPath="~\Providers\DataProviders\SqlDataProvider\"
                objectQualifier=""
                databaseOwner="dbo" />
        </providers>
</data>
```

Puedes ver en el web.config dónde corresponden los valores definidos en el archivo SQLDataProvider.vb con los valores dentro del web.config.
Ahora necesitamos exponer nuestra información a través de algunas propiedades públicas:

```
Public ReadOnly Property ConnectionString() As String
        Get
                Return _connectionString
        End Get
End Property

Public ReadOnly Property ProviderPath() As String
        Get
                Return _providerPath
        End Get
End Property

Public ReadOnly Property ObjectQualifier() As String
        Get
                Return _objectQualifier
        End Get
End Property

Public ReadOnly Property DatabaseOwner() As String
        Get
                Return _databaseOwner
        End Get
End Property
```

Pasemos ahora al siguiente apartado dentro de la clase SQLDataProvider. Ahora necesitamos ejecutar las rutinas que realizan la conexión y la manipulación de la base de datos real.

Acuérdate de la capa abstracta de datos en la que había métodos anulables que anulamos desde dentro de nuestro proveedor de base de datos física. Revisemos los métodos utilizados para conectarnos a nuestra base de datos física en SQL.

```
Private Function GetNull(ByVal Field As Object) As Object
        Return DotNetNuke.Common.Utilities.Null.GetNull(Field, DBNull.Value)
End Function

Public Overrides Function GetSurveys(ByVal ModuleId As Integer) As IDataReader
        Return CType(SqlHelper.ExecuteReader(ConnectionString, DatabaseOwner &
ObjectQualifier & "GetSurveys", ModuleId), IDataReader)
End Function
Public Overrides Function GetSurvey(ByVal SurveyID As Integer, ByVal ModuleId As
Integer) As IDataReader
        Return CType(SqlHelper.ExecuteReader(ConnectionString, DatabaseOwner &
ObjectQualifier & "GetSurvey", SurveyID, ModuleId), IDataReader)
End Function
Public Overrides Function AddSurvey(ByVal ModuleId As Integer, ByVal Question As String,
ByVal ViewOrder As Integer, ByVal OptionType As String, ByVal UserName As String) As
Integer
        Return CType(SqlHelper.ExecuteScalar(ConnectionString, DatabaseOwner &
ObjectQualifier & "AddSurvey", ModuleId, Question, GetNull(ViewOrder), OptionType,
UserName), Integer)
End Function
Public Overrides Sub UpdateSurvey(ByVal SurveyId As Integer, ByVal Question As String,
ByVal ViewOrder As Integer, ByVal OptionType As String, ByVal UserName As String)
        SqlHelper.ExecuteNonQuery(ConnectionString, DatabaseOwner & ObjectQualifier &
"UpdateSurvey", SurveyId, Question, GetNull(ViewOrder), OptionType, UserName)
End Sub
Public Overrides Sub DeleteSurvey(ByVal SurveyID As Integer)
        SqlHelper.ExecuteNonQuery(ConnectionString, DatabaseOwner & ObjectQualifier &
"DeleteSurvey", SurveyID)
End Sub
::
::
```

En la revisión del código puedes ver que hay un método que corresponde con cada procedimiento almacenado en nuestra base de datos, a su vez cada método incluido dentro de nuestra capa de datos abstracta se anula y nuestros métodos con el proveedor de base de datos física devolverán los datos.
Esto permite que cualquier proveedor pueda ser utilizado dentro del portal DNN sin una recompilación a tu ensamblado base.
Además, presta también atención a la función GetNull dentro de la clase; utiliza esto para dar cuentas de los valores nulos dentro de los campos de tu base de datos.

Interconectar con el DNN

En este aparatado vamos a tratar las tareas más comunes con los que los desarrolladores se encuentran a la hora de escribir módulos del DNN. DNN proporciona al desarrollador diversos proveedores para implementar varias tareas; estas consisten en editar contenido, registrar eventos en el registro del portal, programar tareas desconectadas, etc.
Muchas de estas herramientas utilizan el modelo de proveedor del DNN. Incluimos proveedores nativos del DNN. El que crees tus propios proveedores para reemplazar los proveedores centrales está fuera del alcance de esta guía.

Personalización

DNN proporciona desarrolladores de módulo con la capacidad de almacenar valores únicos basados en el acceso del usuario específico al portal. Esto te permite crear una experiencia a medida para cada usuario de tu módulo. En el siguiente código, comprobaremos si la personalización está permitida para el módulo, comprobando primero el valor dentro de la clave “surveytracking” en los Settings numerados para este ejemplo de módulo. Si hemos permitido la personalización, entonces haremos una comprobación del método DotNetNuke.Services.Personalization.Personalization.GetProfile. El método GetProfile acepta el valor del Module ID actual y devuelve el valor de la clave especificada anteriormente en la lógica de aplicación. Revisemos el código dentro del evento Page_Load en el archivo Survey.ascx.vb.

```
' Check the settings to see if this module is using Personalization for vote tracking
If Not Settings("surveytracking") Is Nothing Then
        blnPersonal = CType(Settings("surveytracking"), Boolean)
Else
        blnPersonal = False
End If
::
::
If blnPersonal = True Then
' This means the module vote tracking is using personalization, so check the database to
see if this user has voted
' We first must make sure the user is a registered user who is logged in
        If UserId <> -1 Then
                ' Check if the user has voted before
                If Not
DotNetNuke.Services.Personalization.Personalization.GetProfile(ModuleId.ToString,
"Voted") Is Nothing Then
                        blnVoted =
CType(DotNetNuke.Services.Personalization.Personalization.GetProfile(ModuleId.ToString,
"Voted"), Boolean)
                Else
                        blnVoted = False
                End If
                'If not voted before show the Survey items, otherwise just show the results
                If blnVoted = False Then
                        DisplaySurvey()
                Else
```

```
                    DisplayResults()
                End If
        Else
        ' The User is not logged in and the module is using personalization, so present the
user a message that he needs to login first
                DotNetNuke.UI.Skins.Skin.AddModuleMessage(Me,
Localization.GetString("Survey_MustBeSignedIn", Me.LocalResourceFile),
DotNetNuke.UI.Skins.Controls.ModuleMessage.ModuleMessageType.YellowWarning)
                                                cmdSubmit.Visible = False
                                                DisplaySurvey()
        End If
Else
' Survey is not personalized, so just show Survey items
        DisplaySurvey()
End If
```

Ahora que hemos tratado cómo leer los valores de personalización, estando todavía en el archivo de clase Survey.ascx.vb, veámos cómo se establece un valor para la clave de personalización en el método cmdSubmit_Click:

```
::
::
If blnPersonal = True Then
 ' This means the module vote tracking is using personalization, so set the profile to show they voted

  DotNetNuke.Services.Personalization.Personalization.SetProfile(ModuleId.ToString,"Voted", True)
::
::
```

En este bloque de código en lugar de hacer un GetProfile como en el evento Page_Load, utilizamos el método SetProfile para guardar nuestro valor en la clave “Voted”.

Planificador

En DNN 2, el proveedor de programación se implementó dentro del marco central. Esto permite a los desarrolladores implementar un proceso sobre una base programada en su propio proceso, independiente de una petición Web.
La manera de trabajar del programador es que introduzcas una definición del evento en la tabla para Programar dentro de la base de datos del DNN. Veámos la estructura y una entrada de muestra para purgar el registro del sitio. Aquí tienes los nombres de campo junto con valores asociados para esta entrada específica:

- **ScheduleID**: 2
- **TypeFullName**: DotNetNuke.Services.Log.SiteLog.PurgeSiteLog, DOTNETNUKE – Este es el método que se ejecutará en una base programada.
- **TimeLapse**: 1 – Cuánto tiempo hay entre intervalos de ejecución.
- **TimeLapseMeasurement**: d – La unidad de medida, en este caso, días.
- **RetryTimeLapse**: 2 – Esto es cuánto tiempo habrá entre retrys (reintentos).
- **RetryTimeLapseMeasurement**: h – La medida del retry, en este caso, horas.

- **RetainHistoryNum**: 10 – Cuántos incidentes se registrarán en el proveedor de registro.
- **AttachToEvent**: NULL – Especifica si un evento debería quitarse del evento APPLICATION_ONSTART.
- **CatchUpEnabled**: 0 – Un indicador para especificar si este servicio recuperará desde eventos perdidos.
- **Enabled**: 1 – Un indicador para especificar si este item estará permitido o no.
- **ObjectDependencies**: SiteLog – Cualquier dependencia para esta clase, en este caso DotNetNuke.Services.Log.SiteLog.

Así que la idea aquí es que puedas introducir un item dentro de la tabla para programar sobre la instalación del módulo o hacerlo programáticamente. Te darás cuenta de que en el campo TypeFullName estamos pasando el nombre completo de la colección o de la clase que se necesita ejecutar. Echemos un vistazo a la estructura de la clase DotNetNuke.Services.Log.SiteLog.PurgeSiteLog y veámos cómo se estructura para registrar un evento.

```
Imports System.Threading

Namespace DotNetNuke.Services.Log.SiteLog

        Public Class PurgeSiteLog
                Inherits DotNetNuke.Services.Scheduling.SchedulerClient
                Public Sub New(ByVal objScheduleHistoryItem As
DotNetNuke.Services.Scheduling.ScheduleHistoryItem)
                        MyBase.new()
                        Me.ScheduleHistoryItem = objScheduleHistoryItem
                End Sub

                Public Overrides Sub DoWork()
                        Try
                                'notification that the event is progressing
                                Me.Progressing()                        'OPTIONAL
                                PurgeSiteLog()
                                Me.ScheduleHistoryItem.Succeeded = True
                                Me.ScheduleHistoryItem.AddLogNote("Site Log purged.")
                        Catch exc As Exception                  'REQUIRED
                                Me.ScheduleHistoryItem.Succeeded = False
                                Me.ScheduleHistoryItem.AddLogNote("Site Log purge failed. " +
exc.ToString)                           'OPTIONAL
                                'notification that we have errored
                                Me.Errored(exc)                         'REQUIRED
                                'log the exception
                                LogException(exc)                       'OPTIONAL
                        End Try
                End Sub

                Private Sub PurgeSiteLog()
               ‘run scheduled routine.
```

```
            ::
            ::
             End Sub
         End Class

End Namespace
```

Puedes ver desde la clase que estamos heredando el DotNetNuke.Services.Scheduling.SchedulerClient. Esto nos proporciona los métodos necesarios para registrar eventos en el programador. Otro item de interés es el método DoWork, puedes ver que se anula el método DoWork que está ubicado en el SchedulerClient; este es el método que ejecutará el servicio programador.

Proveedor de Registro

Otro servicio que proporcina el DNN es la capacidad de inscribir items en un registro. DNN incluye un Proveedor de Registro XML para captar eventos subidos del núcleo del DNN, y módulos. Esto puede ser útil para escribir eventos erróneos, o cualquier otra información a la que el administrador del portal quiera seguir la pista. Puedes ver en el apartado relacionado con el proveedor de Programación, que se hace una llamada al proveedor de Registro para añadir entradas al registro del portal.
Hay dos entradas de registro del evento fundamentales que puedes implementar; una entrada informativa estándar, o una entrada para atrapar un error dentro de tu código. Primero veámos la entrada informativa normal:

```
'Write out the log entry for this event
''''''''''''''''''''''''''''''''''''''''''''''''''

Dim objEventLog As New Log.EventLog.EventLogController
Dim objEventLogInfo As New Log.EventLog.EventLogInfo
objEventLogInfo.AddProperty("THREAD ID",
Thread.CurrentThread.GetHashCode().ToString)
objEventLogInfo.AddProperty("TYPE", objSchedulerClient.GetType().FullName)
objEventLogInfo.AddProperty("SOURCE",
objSchedulerClient.ScheduleHistoryItem.ScheduleSource.ToString)
objEventLogInfo.AddProperty("ACTIVE THREADS", ActiveThreadCount.ToString)
objEventLogInfo.AddProperty("FREE THREADS", FreeThreads.ToString)
objEventLogInfo.AddProperty("READER TIMEOUTS", ReaderTimeouts.ToString)
objEventLogInfo.AddProperty("WRITER TIMEOUTS", WriterTimeouts.ToString)
objEventLogInfo.AddProperty("IN PROGRESS", GetScheduleInProgressCount().ToString)
objEventLogInfo.AddProperty("IN QUEUE", GetScheduleQueueCount().ToString)
objEventLogInfo.LogTypeKey = "SCHEDULER_EVENT_STARTED"
objEventLog.AddLog(objEventLogInfo)
```

El ejemplo anterior se consigue desde el servicio programador. Primero hemos llamado al EventLogController, y creado un objeto EventLogInfo. Puedes ver que hemos creado propiedades del evento de registro para nuestro objeto EventLogInfo, y éstas contienen valores que se generan desde nuestra lógica. Finalmente el objeto EventLogInfo se pasa a nuestra clase controlador, que **inicia** el evento en el proveedor de registro.

Ahora veámos un envio de información excepcional al proveedor de registro:

```
Try
    'do some code.
Catch exc As Exception
     Dim objExceptionLog As New Log.EventLog.ExceptionLogController
     objExceptionLog.AddLog(exc,
Log.EventLog.ExceptionLogController.ExceptionLogType.MODULE_LOAD_EXCEPTION)
End Try
```

En el ejemplo anterior, pondremos nuestro evento de registro en la instrucción match para una instrucción Tuy que contiene alguna lógica. Si se produce un error, entonces creamos un ejemplo de clase ExceptionLogController, y pasamos la excepción al método AddLog. Además de la excepción, también definiremos un tipo de Module_Load_Exception, ya que estamos en un módulo a medida (custom).

Localización. Selección del lenguaje y configuraciones locales.

Una nueva característica del DNN 3 es la localización utilizando archivos de recurso. Ahora un usuario puede seleccionar un lenguaje de su elección para el portal del DNN modificando su perfil dentro del DNN. También puedes proporcionar archivos de recurso con tu módulo cuando lo distribuyas. Localizar tu módulo va más allá de lo que podemos tratar en esta guía. Se puede encontrar más información en el documento de localización en la carpeta de documentos incluida en la distribución del DNN.

Importación y Exportación

Otra característica del DNN es la capacidad de importar y exportar módulos dentro del portal. Tratamos esto en la guía anteriormente, pero ahora introduzcámonos en el código que te permite implementar esta característica en tus propios módulos. Esta funcionalidad la proporciona la interfaz DotNetNuke.Entities.Modules.Iportable. Esta interfaz te proporciona la capacidad de importar y exportar módulos.
Lo primero que necesitamos es implementar la interfaz dentro de nuestra clase, tomamos el código siguiente del módulo Text/HTML:

```
Public MustInherit Class RDMView
        Inherits RDMUserControl
        Implements Entities.Modules.IActionable
        Implements Entities.Modules.IPortable
```

Ahora que la interfaz está implementada en la clase hemos de encargarnos de dos eventos. Éstos son los módulos de importación y exportación. Primero echaremos un vistazo al archivo HTMLModule.ascx, al final de la clase veremos dos eventos:

```
Public Function ExportModule(ByVal ModuleID As Integer) As String Implements
Entities.Modules.IPortable.ExportModule
         ' included as a stub only so that the core knows this module Implements
Entities.Modules.IPortable
End Function
```

```
Public Sub ImportModule(ByVal ModuleID As Integer, ByVal Content As String, ByVal
Version As String, ByVal UserId As Integer) Implements
Entities.Modules.IPortable.ImportModule
        ' included as a stub only so that the core knows this module Implements
Entities.Modules.IPortable
End Sub
```

Puedes ver que los métodos están incluidos en la clase para dejar al núcleo que sepa que vamos a proporcionar habilidades de importar/exportar en la función. Ahora hemos de encargarnos de los datos reales incluidos en el módulo para que puedan exportarse. Abre el archivo HTMLTextController.vb para ver cómo se consigue esto. Como en el archivo HTMLModule.ascx, necesitamos implementar la interfaz IPortable en nuestra clase, entonces necesitamos crear otra vez los métodos ExportModule e ImportModule, pero esta vez nos encargaremos de importar y exportar:

```
Public Function ExportModule(ByVal ModuleID As Integer) As String Implements
Entities.Modules.IPortable.ExportModule
        Dim strXML As String = ""
        Dim objHtmlText As HtmlTextInfo = GetHtmlText(ModuleID)
        If Not objHtmlText Is Nothing Then
                strXML += "<htmltext>"
                strXML += "<desktophtml>" & XMLEncode(objHtmlText.DeskTopHTML) &
"</desktophtml>"
                strXML += "<desktopsummary>" &
XMLEncode(objHtmlText.DesktopSummary) & "</desktopsummary>"
                strXML += "</htmltext>"
        End If
        Return strXML
End Function

Public Sub ImportModule(ByVal ModuleID As Integer, ByVal Content As String, ByVal
Version As String, ByVal UserId As Integer) Implements
Entities.Modules.IPortable.ImportModule

    Dim xmlHtmlText As XmlNode = GetContent(Content, "htmltext")
    UpdateHtmlText(ModuleID, xmlHtmlText.SelectSingleNode("desktophtml").InnerText,
xmlHtmlText.SelectSingleNode("desktopsummary").InnerText, UserId)
End Sub
```

En la función ExportModule, vamos a llamar a nuestro método GetHtmlText y a pasar el ModuleID actual. Éste devolverá los datos para la instancia del módulo, que creará una cadena para exportar en formato XML. Este XML se guardará dentro de la base de datos y se podrá importar más tarde a otro ejemplo de módulo utilizado la función ImportModule. En la función ImportModule, llamamos al método GetContent para obtener los datos del módulo exportado, y entonces llenamos la instancia del módulo actual llamando al método UpdateHtmlText de nuestra clase controlador.

Búsqueda

Una novedad en el DNN 3.0 es nueva la capacidad de catalogar el contenido del módulo para realizar búsquedas dentro de una interfaz unificada. Para permitir a tu módulo la búsqueda necesitarás implementar la Interfaz DotNetNuke.Services.Search.ISearchable en tu archivo de clase para tu control. Por ejemplo, el siguiente código es del módulo de Anuncios que utiliza la interfaz de búsqueda del DNN. En el code behind (código fuente) implementarías la interfaz ISearchable en tu clase:

```
::
::
Public Class AnnouncementsController
    Implements DotNetNuke.Services.Search.ISearchable
::
```

Ahora que has implementado la clase, necesitas llenar el indice para realizar búsquedas.

```
Public Function GetSearchItems(ByVal ModInfo As Entities.Modules.ModuleInfo) As
Services.Search.SearchItemInfoCollection Implements
Services.Search.ISearchable.GetSearchItems
        Dim SearchItemCollection As New SearchItemInfoCollection

        Dim Announcements As ArrayList = GetAnnouncements(ModInfo.ModuleID)

        Dim objAnnouncement As Object
        For Each objAnnouncement In Announcements
                Dim SearchItem As SearchItemInfo
                With CType(objAnnouncement, AnnouncementInfo)

                        Dim UserId As Integer
                        If IsNumeric(.CreatedByUser) Then
                                UserId = Integer.Parse(.CreatedByUser)
                        Else
                                UserId = 2
                        End If
                        SearchItem = New SearchItemInfo(ModInfo.ModuleTitle & " - " & .Title,
ApplicationURL(ModInfo.TabID), .Description, UserId, .CreatedDate, ModInfo.ModuleID,
"Anncmnt" & ModInfo.ModuleID.ToString & "-" & .ItemId, .Description)
                        SearchItemCollection.Add(SearchItem)
                End With
        Next

        Return SearchItemCollection
End Function
```

Puedes ver en el bloque de código anterior que hemos hecho una llamada a nuestra clase Info para nuestro módulo, tal y como hubiéramos hecho al limitar un control dentro de nuestro archivo ascx. Pero en este caso los resultados van a llenar el SearchItemInfo, que a su vez llenan el índice del DNN con datos de nuestro módulo.

Ahora que el índice contiene los datos del modulo, los usuarios de tu portal podrán buscar información de tu módulo desde una interfaz unificada dentro del DNN.

Editor de Texto

Otro proveedor en el DNN es el Editor de Texto, actualmente el proveedor por defecto es el control FreeTextBox de John Dyer (http://www.freetextbox.com/). Puedes cambiar este proveedor por otro creando tu propio proveedor y cambiando el web.config para indicar el nuevo proveedor. En esta guía, proporcionaremos una pequeña visión general de cómo puedes implementar el FTB en tus propios módulos.

En el siguiente ejemplo de código puedes encontrar el proyecto de módulo Text/HTML en el directorio DesktopModules.

Para implementar el Editor de Texto primero necesitamos crear una referencia para este control en nuestro archivo ascx. Vemos en el inicio de la página de control ascx justo bajo la llamada a la clase del Code Behind=”ucEditRecord.ascx.vb” (codigo fuente) el registro del Editor de texto(TextEditor) TextEditor.ascx.

```
<%@ Control language="vb" Inherits="Santry.ResDirMgr.ucEditRecord"
CodeBehind="ucEditRecord.ascx.vb" AutoEventWireup="false" Explicit="True" %>
<%@ Register TagPrefix="dnn" TagName="TextEditor"
Src="~/controls/TextEditor.ascx"%>
```

Vemos que hemos registrado en el Register TagPrefix el prefijo del tag para el control del Editor de Texto (TextEditor.ascx), así podemos insertarlo en nuestro control. Más adelante en el código, puedes ver dónde está insertado el control para la interfaz visual.

```
<dnn:TextEditor id="TextEditor1" Width="600" Height="400" runat="server"
HtmlEncode="False" />
```

Ahora puedes tener la funcionalidad del editor de texto en tus propios módulos. Revisaremos el código de la página anterior para ver cómo declaramos el módulo y así modificar sus propiedades. Lo primero que hacemos es dar la referencia a una nueva interfaz llamada DotNetNuke.UI.UserControls.TextEditor:

```
Protected WithEvents TextEditor1 As DotNetNuke.UI.UserControls.TextEditor
```

Ahora que hemos creado nuestra referencia, modificamos las propiedades y utilizamos los eventos del control. Como si cogiéramos el texto de un control asignándole una variable:

```
strHTML = TextEditor1.Text
```

No tienes que preocuparte de escribir el código para cambiar de texto a la vista html, y todas las demás características que proporciona. Además, puedes crear tu propio proveedor o comprar un tercer proveedor y conectarlo en el portal.

Controles para el Usuario del DotNetNuke

Una novedad del DNN 3 es la disponibilidad de controles propios para el usuario. Con el DNN tienes más controles que los Web estándar proporcionados por ASP.NET. Como se trató en el apartado del Editor de texto, tienes la clase DotNetNuke.UI.UserControls, que te proporciona controles que se aprovechan del marco central del DNN. Entre los controles más comunes incluidos en esta interfaz están :

- **Address** – Este control proporciona el control de entrada de dirección que ves en el DNN cuando registras una cuenta dentro del portal.
- **DesktopModuleTitle** – Proporciona el área del título sobre un módulo.
- **DesktopPortalBanner** –El área de banner dentro del portal, incluye el registro, el banner, el logo, y otros elementos.
- **DesktopPortalFooter** – El área del pie de página de un portal.
- **DualListControl** – Un control que contiene dos controles de lista para mover datos entre los dos.
- **Help** – Control de ayuda de un módulo. Acepta información sobre la localización y referencias del archivo de ayuda.
- **LabelControl** – Contiene propiedades para almacenar información sobre la localización, información de ayuda, y otra información proporcionada por el núcleo del DNN.
- **ListItemComparer** – Se utiliza para comparar items dentro de una lista.
- **ModuleAuditControl** – Para consultar un módulo, como quién creó el módulo y cuándo se creó.
- **SectionHeadControl** – Para crear áreas colapsables dentro de tu módulo.
- **TextEditor** – Visto anteriormente
- UrlControl -
- URLTrackingControl

Utiliza el navegador del objeto dentro del Visual Studio para ver más información sobre los controles y las interfaces del DNN. Te recomendamos que utilices los controles para el usuario del DNN para aprovechar la localización, y la integración de ayuda dentro del marco central del DNN.

Captura de Errores

DotNetNuke proporciona a los desarrolladores un mecanismo para presentar errores personalizados al usuario, y determinar si se muestra información detallada sobre quién está registrado o no. Además de visualizar errores, DNN también proporcionará un mecanismo para registrar los errores en el proveedor de registro para revisarlos posteriormente.

```
Try
    'Do some code.
Catch exc As Exception                    'Module failed to load
    ProcessModuleLoadException(Me, exc)
End Try
```

En el bloque de código anterior puedes ver en la instrucción try/catch que hemos llamado a la función ProcessModuleLoadException y pasado nuestro error al método. Esto enviará nuestro error al portal y ejecutara las rutinas de la captura de errores del DNN.

Comunicación Inter-Módulo

La Comunicación Inter Módulo (IMC) es una manera muy versátil y significativa de transferir datos entre los módulos en tu portal. Aunque la IMC te puede hacer muchas cosas, el nombre es engañoso y este apartado intentará organizar tanto sus ventajas como sus inconvenientes.
En el 2.0.1 beta DNN, la IMC se aumentó para permitir cuatro propiedades nuevas en la interfaz. En la versión original se aplazaron desde la serie IBuySpy a la DotNetNuke 1.x, había una propiedad: [TEXT], y podías pasar una cadena desde un módulo remitente a un módulo receptor. Sin embargo no había una manera de determinar el remitente o el destinatario, o el tipo de mensaje, y no había forma de pasar un Objeto. Esto se solucionó y se añadieron las propiedades para facilitar esta funcionalidad requerida. No obstante, las mejoras se pasaron como 6 meses en el limbo mientras el 2.0 se estaba compilando. Y así es como salió, fue una suerte.
Un cambio en el último momento resolvió la dependencia de la IMC de la cache, la aísla de la funcionalidad del skinning determinando la interfaz a ser una función básica, y la IMC en 2.0.1 beta DNN funciona como se pretendía. Ahora, la pregunta es, ¿para qué utilizo yo la IMC, y para qué NO la utilizo?
Este apartado exhibirá diversos módulos que utilizan la IMC actualmente, y cómo se implementan. También se mostrarán circunstancias en las que la IMC no será funcional, incluso cuando “parezca” un uso apropiado.

Cómo Funciona

Las Comunicaciones Inter Módulo proporcionan una manera sencilla y directa de comunicación con otros módulos dentro de tu portal. La naturaleza de esta comunicación está en un evento al que llamaremos iTalk, que es iniciado en el siguiente postback de esa página (tab) y se sube a algún sitio durante la PageInit. Decimos algún sitio, porque en el evento real se incluyen dentro del control y del árbol de eventos, así que éstos variarán dependiendo del acuerdo con tu tab.
Los valores de este evento se hacen accesibles para todos los módulos que existan dentro de ese tab y pueden conocer a quien está dirigido, al tipo de módulo, a los valores del destino y remitente. Llamamos a esto iListen.
La clase ModuleCommunicator es heredada en tu módulo.
Entonces el evento ModuleCommunication es subido dentro del módulo iTalk que lleva tu comunicación a la siguiente pageInit, dónde el OnModuleCommunication Sub en tu módulo iListen puede interceptar y utilizar los valores.

```
Public Interface ImoduleCommunicator
Event ModuleCommunication As ModuleCommunicationEventHandler
End Interface 'IModuleCommunicator


_

Public Interface IModuleListener
Sub OnModuleCommunication(ByVal s As Object, ByVal e As
ModuleCommunicationEventArgs)
End Interface 'IModuleListener
```

Cómo No Utilizarlo

La suposición más común de la IMC es que te permitirá pasar objetos entre módulos. Aunque esto es verdad, el valor sólo persistirá para un solo postback. Cuando la página se renueva la IMC se convierte en nada. Esto se mantiene fácilmente por el uso de las variables Session o viewstate que se declaran dentro de tus módulos que buscan la IMC y la almacenan para ese módulo, usuario, etc. Si los datos necesitan ser interceptados y almacenados en la base de datos, o utilizados en otro postback actual, implementarías un Listener a medida para adjuntarlo a esos datos y procesarlo adecuadamente.
En ningún caso persistirá la cadena de la IMC más allá del postback inicial sin ayuda programática.
La IMC a menudo se confunde con un medio de transferir datos sin el uso de una base de datos. En algunos casos esto es correcto, pero utilizar la IMC de esta manera y acceder a los datos más allá del siguiente postback o petición requiere una capa más robusta, incluyendo la sesión o la base de datos.

Cómo utilizarlo

En cualquier caso la IMC está diseña para utilizarla dentro de módulos en un sólo tab, o en módulos que redirigen a otro tab y ese tab tiene un módulo en el que puede recibir los objetos de IMC.
Lo bueno es que puedes pasar eficientemente objetos entre módulos, o enviarlos a una función o sistema de escucha.

1. Paso Uno: Determinar el remitente [SENDER]: Aquí les das a tus módulos la capacidad de adjuntarse a sólo un objeto de IMC establecido desde un Sender seguro. Este es normalmente el ModuleName del módulo de emisión, como se estableció en las definiciones del módulo. Si un Sender está incluido en el objeto de iMC establecido, y un módulo Listener está configurado en Receive desde el Sender, entonces sólo se adjuntará a un objeto de IMC establecido para ese Sender.
2. Paso Dos: Determinar el receptor [TARGET]: Aquí les das a tus módulos la capacidad de adjuntarse sólo a un objeto de IMC con objetivo en un cierto Target. Esto es normalmente el ModuleName del módulo de recepción, como está establecido en las definiciones del módulo. Si un Módulo está configurado para escuchar a solo el Target del MyModule entonces sólo se adjuntará a un objeto de IMC establecido para ese target.
3. Paso Tres: Determinar el tipo [TYPE]: una propiedad configurable donde tu estableces el tipo en el Talker y entonces el Listener busca ese tipo. Si un Módulo está configurado para Escuchar a sólo el Tipo de MyModule entonces sólo se adjuntará a un objeto de IMC establecido de ese tipo.
4. Paso Cuatro: Determinar el valor [VALUE]: Ésta es la única propiedad del objeto en IMC, y puedes pasar cualquier objeto complejo entre módulos. Este incluye un conjunto de datos, un control dentro del estado, o cualquier otra cosa que puedas imaginar. El requisito para un listener es saber qué hacer con el Objeto y su estado.
5. Paso Cinco: Determinar el texto [TEXT]: Ésta es una respuesta relativamente directa ... un simple texto en cadena. En el IMC original, esta era la única propiedad.

Distribuir Tu Módulo

Hemos tratado cada capa de la arquitectura del módulo del DotNetNuke; ahora preparemos el módulo para su distribución. En los siguientes pasos vamos a crear un paquete de

ensamblado privado fuera de nuestro módulo survey, así puede distribuirse fácilmente a otros portales del DNN.

Crear Tu Base de Datos para la Distribución

Necesitamos crear un script SQL para generar la estructura de la base de datos de nuestra base de datos. ¿Recuerdas las cadenas de propietario de la base de datos y de calificador del objeto especificados en la clase proveedor de datos? Pues bien, necesitaremos especificar estas variables en nuestros scripts para la distribución. DotNetNuke comprobará los valores establecidos. Crear este script lleva un poco más de esfuerzo extra que si fuéramos simplemente al SQL Query Analyzer y exportáramos scripts de creación desde tu base de datos. Sí, así es como empezamos, primero generando scripts de creación a través del Query Analyzer o del Enterprise Manager, pero necesitamos añadir alguna información adicional a estos scripts para que el DNN cree adecuadamente la estructura de tu base de datos.
Después de que hayas generado scripts de creación SQL, necesitarás añadir los siguientes prefijos delante de tu tabla, y refencias de los procedimientos almacenados dentro de los scripts creados recientemente.
{databaseOwner}{objectQualifier}
Estos prefijos se reemplazarán por valores definidos dentro del web.config en el tiempo de generación de la base de datos durante la instalación del módulo. Así, en el siguiente ejemplo un script SQL genera una tabla:

```
if not exists (select * from dbo.sysobjects where
 id = object_id(N'{databaseOwner}{objectQualifier}[SurveyOptions]')
 and OBJECTPROPERTY(id, N'IsTable') = 1)
CREATE TABLE {databaseOwner}{objectQualifier}SurveyOptions (
        [SurveyOptionID] [int] IDENTITY (1, 1) NOT NULL ,
        [SurveyID] [int] NOT NULL ,
        [ViewOrder] [int] NOT NULL ,
        [OptionName] [nvarchar] (500) NOT NULL ,
        [Votes] [int] NOT NULL
) ON [PRIMARY]

GO
```

Puedes ver dónde hemos destacado los prefijos, entonces en tu script de generación de procedimiento almacenado verás que éstos son añadidos a cualquiera de las referencias a las tablas:

```
create procedure {databaseOwner}{objectQualifier}UpdateSurveyOption

@SurveyOptionID int,
@OptionName     nvarchar(500),
@ViewOrder      int

as

update {objectQualifier}SurveyOptions
set    OptionName = @OptionName,
```

```
    ViewOrder = @ViewOrder
where  SurveyOptionID = @SurveyOptionID

GO
```

Una vez que tu script SQL está completo, guárdalo utilizando la siguiente convención de nombre:
versionnum.dataprovidertype, por ejemplo: 01.00.00.SQLDataProvider
Esto se incluirá en tu paquete de distribución. Dependiendo de qué proveedor de datos esté configurado en el DNN, buscará el script de generación de la base de datos apropiado basado en un valor de extensión.
Uno de los archivos que deberías crear como parte de tu distribución es el script de desinstalación. Este script contiene básicamente rutinas de eliminación para borrar tus tablas y procedimientos almacenados si el administrador del portal desea eliminar tu módulo desde la instalación del DNN.
La convención de nombre de este script de desinstalación es la siguiente:
uninstall.dataprovidertype, por ejemplo: uninstall.SqlDataProvider

NOTA: Calificador del Objeto, y Propietario de la Base de Datos
Se requiere tener estos prefijos añadidos a los scripts de la base de datos de tu módulo. Estos prefijos permiten que múltiples implementaciones del DNN existan dentro de una sola base de datos. Además, si las tablas de tu módulo necesitan unirse a otras tablas en la estructura central del DNN, estos prefijos necesitan ser definidos para que las instrucciones SQL funcionen.

Crear Tu Definición para la Distribución del DNN

El archivo de definición para la instalación del ensamblado del DNN es un archivo XML que describe la estructura de tu módulo. Contiene diversos datos para que el DNN instale tu módulo dentro de la aplicación. Este archivo de definición reflejará la arquitectura para tu módulo específico, que va a variar según cómo estructures tu módulo. Veámos uno de los ejemplos del módulo survey.

```
<?xml version="1.0" encoding="utf-8" ?>
  <dotnetnuke version="3.0" type="Module">
  <folders>
    <folder>
      <name>CompanyName - Survey</name>
      <description>Survey allows you to create custom surveys
        to obtain public feedback</description>
      <version>01.00.00</version>
      <resourcefile>SurveyResources.zip</resourcefile>
      <businesscontrollerclass>CompanyName.Survey.SurveyController,
CompanyName.Survey</businesscontrollerclass>
      <modules>
        <module>
          <friendlyname>CompanyName - Survey</friendlyname>
          <controls>
```

```
        <control>
          <src>Survey.ascx</src>
          <type>View</type>
          <helpurl>http://www.dotnetnuke.com</helpurl>
        </control>
        <control>
          <key>Edit</key>
          <title>Create Survey</title>
          <src>EditSurvey.ascx</src>
          <iconfile>icon_survey_32px.gif</iconfile>
          <type>Edit</type>
        </control>
        ::
      </controls>
    </module>
  </modules>
```

Veámos el XML de arriba. Puedes ver que hemos definido nuestro módulo en el nivel más alto. Entonces necesitamos que el DNN sepa en qué consiste nuestro módulo, esta sería la principal interfaz del usuario o archivos ascx. En el módulo survey tenemos tres definidos, te proporcionamos dos en este ejemplo, un control de vista definido primero y un control para editar. Puedes ver dentro del control para editar que definimos el nombre del archivo utilizado, un nombre clave, un valor de título para el contenedor del módulo, el icono, y el tipo de control, en este caso es un control para editar. Ya que es un control para editar, podemos utilizar el valor booleano proporcionado por el DNN para comprobar los permisos para ver si el usuario ha accedido a las funciones de editar en el módulo (¿recuerdas "IsEditable" desde la clase PortalModuleBase?).
Presta también atención a que en los módulos que desarrollas puedes crear una definición del archivo muy diferente a la que utilizamos en esta guía. Un módulo puede contener varios módulos de adminstración y controles para editar. Todo esto esta determinado por la complejidad del módulo al ser desarrollado y las necesidades de negocio. Echémos un vistazo al módulo de Diarios Web (Weblogs) como ejemplo de un módulo que contenga multiples módulos, dentro de un archivo de definición:

```
::
<modules>
        <module>
                <friendlyname>Weblogs</friendlyname>
                <controls>
                        <control>
                                <src>Weblogs.ascx</src>
                                <type>0</type>
                        </control>
                        <control>
                                <key>Edit</key>
                                <title>Add Weblog entry</title>
                                <src>EditWebLogs.ascx</src>
                                <iconfile></iconfile>
                                <type>1</type>
```

```
                    </control>
                    <control>
                        <key>Options</key>
                        <title>Weblog Options</title>
                        <src>EditWebLogsOptions.ascx</src>
                        <iconfile></iconfile>
                        <type>1</type>
                        </control>
                    </controls>
        </module>
        <module>
            <friendlyname>Weblogs Calendar</friendlyname>
            <controls>
                    <control>
                        <src>NavCalendar.ascx</src>
                        <type>0</type>
                    </control>
            </controls>
        </module>
        <module>
            <friendlyname>Weblogs Archive</friendlyname>
            <controls>
                    <control>
                        <src>NavArchive.ascx</src>
                        <type>0</type>
                    </control>
            </controls>
        </module>
::
```

Puedes ver en el archivo de definición anterior, que hemos definido múltiples apartados del módulo dentro de esta instalación del módulo. Esto se debe a que el módulo es un poco complejo y requiere múltiples controles para que funcione adecuadamente. En este caso tenemos un calendario, y módulos archivados que están asociados con el Weblog. Puedes añadir tantos apartados de módulo como tu instalación necesite, y así asociar los controles con los módulos con los que interactúen tus usuarios.
Ahora que hemos definido la estructura básica de nuestro módulo, tenemos que definir los archivos reales que componen nuestro módulo. Estos archivos incluyen el DLL, controles ascx, proveedores de datos, imágenes, y cualquier otro archivo que admita nuestra aplicación específica.

```
    ::
    <files>
      <file>
        <name>Survey.ascx</name>
      </file>
      <file>
        <name>EditSurvey.ascx</name>
      </file>
```

```
      <file>
        <name>Settings.ascx</name>
      </file>
      <file>
        <name>YourCompanyName.Survey.dll</name>
      </file>
      <file>
        <name>YourCompanyName.Survey.SqlDataProvider.dll</name>
      </file>
      <file>
        <name>01.00.00.SqlDataProvider</name>
      </file>
      <file>
        <name>Uninstall.SqlDataProvider</name>
      </file>
      <file>
        <name>YourCompanyName.Survey.AccessDataProvider.dll</name>
      </file>
      <file>
        <name>01.00.00.AccessDataProvider</name>
      </file>
      <file>
        <name>Uninstall.AccessDataProvider</name>
      </file>
      <file>
      <file>
          <path>App_LocalResources</path>
          <name>Survey.ascx.resx</name>
        </file>
        <file>
          <path>App_LocalResources</path>
          <name>EditSurvey.ascx.resx</name>
        </file>
        <file>
          <path>App_LocalResources</path>
          <name>Settings.ascx.resx</name>
        </file>
    ::
    </files>
  </folder>
 </folders>
</dotnetnuke>
```

Si miras el ejemplo de código, puedes ver que hemos definido nuestros archivos para que estén incluidos en el módulo. Asegúrate de que tienes todos los archivos incluidos en tu archivo zip. Además, tenemos el nodo <path> para definir si los archivos están ubicados en un subdirectorio, en este ejemplo, tenemos archivos de recurso para la localización. Los archivos de recurso se colocarán en el directorio App_LocalResources fuera del directorio home del módulo.

Ahora que el archivo está creado, coge todos tus archivos para tu módulo y comprímelos en un archivo comprimido zip. Este es el ensamblado privado que puedes distribuir. Recuerda, no necesitas incluir ningún archivo fuente en esta distribución, sólo el ensamblado compilado, los archivos ascx, y otros archivos de apoyo --- no se necesita ninguno de los archivos de clase filename.vb.
Veámos algunos de los puntos que deberás seguir cuando crees tus propios archivos de definición.

- Asegúrate de que añades el valor del atributo de la versión del DNN apropiado en el nodo del DotNetNuke.
- Además de los nodos tratados en el ejemplo, puedes añadir el nodo helpURL para definir una página Web para un editor, administrador o control de vista específico.
- Se ha añadido un nodo nuevo llamado la businesscontrollerclass al archivo de definición. Esto se hace para definir la clase asociada para tu módulo, por ejemplo:

```
<businesscontrollerclass>CompanyName.Survey.SurveyController, CompanyName.Survey</businesscontrollerclass>
```

Validador de la Definición del Módulo

DNN te proporciona una herramienta para comprobar el archivo de definición y asegurarte que sea válido antes de su distribución. Para comprobar la validez de tu archivo .dnn, navega por el menú Module Definitions, que está ubicado bajo el menú Host dentro de tu portal. Una vez en el tab Module Definitions, desplázate al final de la página para ver el validador. Haz click en el botón de buscador para encontrar el fichero .dnn ubicado en tu máquina local. Esto abrirá un diálogo del explorador, encuentra tu archivo .dnn, y haz click en “Abrir”. Ahora deberías ver la trayectoria local de tu archivo .dnn en el cuadro de texto. Haz click en el botón de “Validar” para que comience el proceso de validación. Si tu archivo es válido, deberías ver el siguiente mensaje:

Resultados de la Validación

El CompanyName.Survey.dnn es un Archivo de la Definición del Módulo.

Si el archivo no es válido, te notificará el problema.

Empaquetar tu Módulo

Tu módulo será un archivo comprimido zip que incluirá todos los archivos relevantes para la operación de tu módulo. Normalmente la distribución de un módulo contiene los siguientes archivos:

- **Company.ModuleName.SQLDataProvider.dll** – Este es el código comprimido desde tu proyecto de proveedor de datos. En este ejemplo estamos utilizando el SQL como nuestro secundario, pero tu nombrarás este archivo según el proveedor de la base de datos física que estés utilizando.
- **Company.ModuleName.dll** – Este es el proyecto de módulo principal que está compilado en un ensamblado.
- **All ASCX Files** – Esto incluye todos los archivos de la interfaz del usuario. En este ejemplo incluiremos nuestro archivo Survey.ascx, EditSurvey.ascx, y Settings.ascx.
- **Images** – Si tu módulo necesita algunos archivos de imagen, como iconos, u otras imágenes, empaquétalos en el archivo zip.
- **CSS** – Si quieres incluir una hoja de diseño en cascada específica para tu módulo. Hace referencia al apartado siguiente de estructura de la hoja de estilo.

- **Data Provider Definition** – Incluye el script de creación de proveedor de base de datos física. Si vas a proporcionar múltiples proveedores para admitir a los múltiples proveedores para el DNN, inclúyelos también.
- **Data Provider Uninstall** – Incluye el script de desinstalación .Si tienes múltiples proveedores de base de datos incluye un archivo de desinstalación para cada proveedor.
- **Resource Files for Localization** – Si pretendes proporcionar múltiples idiomas para tu módulo, necesitarás distribuir los archivos de recurso admitidos en tu distribución.

Consejo: Compila En Modo Release

Asegúrate de que compilas tus ensamblados en modo release y no en modo debug antes de añadirlo a tu paquete de distribución. Esto reducirá la cantidad de recursos requeridos para ejecutar tu paquete.

Nota: No necesitas incluir el código fuente en tu paquete de distribución.

Hojas de Estilo

Uno de los temas que queremos tratar son las convenciones de nomenclatura standard para tus hojas de estilo en cascada (CSS). En primer lugar, tu objetivo como desarrollador de módulos es crear una única convención de nomenclatura basada en la convención CompanyName/ModuleName, para así evitar conflictos CSS con los módulos centrales o módulos desarrollados por otros ISVs. Por ejemplo, en el archivo CSS del módulo Survey:

```
/* Survey Custom Style */
.YourCompanyNameSurveyResults
{
   font-family: Tahoma, Arial, Helvetica;
   font-size: 12px;
   font-weight: normal;
}
```

Si no necesitas estilos patentados para tu módulo, entonces deberías utilizar los estilos del DNN que pueden definirse dentro de cada portal para proporcionar al administrador del portal una mayor flexibilidad para definir estilos para su sitio.

Instalación del Ensamblado

Finalmente, ahora que ya tenemos nuestro módulo empaquetado en una distribución, veámos cómo instalarlo dentro de un portal DNN. DNN proporciona un método simple para instalar módulos en el portal a través del Administrador de Archivos. Este es el porqué de todo el trabajo que hiciste al crear un ensamblado, un script de creación de SQL, y un archivo .dnn, ahora todo se junta de modo que un administrador puede fácilmente instalarlo en su instalación del DNN. Utiliza el siguiente procedimiento para instalar tu módulo:

1. Entra al sistema como acceso Host.
2. Ve al Administrador de Archivos bajo el menú de Host.
3. Haz click en el botón de Cargar en la barra de herramientas de arriba del Admistrador de Archivos.

4. En la pantalla de Archivo de Cargar, selecciona el Módulo Custom desde la selección del botón radio.
5. Haz click en el botón buscador para encontrar tu paquete de ensamblado.
6. Haz click en Añadir para visualizar el paquete en la lista.
7. Haz click en Cargar para cargar el ensamblado en el servidor.

Una vez que el archivo está cargado, el DNN buscará el archivo .dnn para encontrar la información que necesitas para configurar tu módulo en el portal. El Administrador de Archivos visualizará el estado de la instalación de tu módulo en la pantalla. Si se encuentra cualquier error durante el proceso de instalación, el DNN te lo hará saber visualizando el problema específico en la pantalla con un texto rojo fuerte.
Si todo se instala correctamente, ahora deberías poder añadir tu módulo a la página seleccionándolo desde el menú desplegable.

Documentación Adicional

Para más información sobre el desarrollo del DNN mira en la documentación apropiada incluida en la carpeta de documentación <app root>/documentation folder. La Ubicación, el cliente API, el Programador, y el proveedor de miembros están explicados más detalladamente en su respectivo documento.

Resumen

En esta guía hemos tratado los temas que deberían haberte iniciado en el desarrollo de módulos del DotNetNuke. Hemos tratado la implementación de módulos dentro del marco del DNN y cómo un administrador de un portal puede modificar las propiedades como la posición, la interfaz visual y la seguridad en un ejemplo de módulo.
Pasamos también por la parte técnica y creamos un proyecto Visual Studio.NET tanto para el módulo como para el proveedor de base de datos física.
Desde ahí pasamos a la arquitectura de tres capas y cómo el desarrollo del módulo del DNN utiliza este concepto arquitectónico. Empezamos con la Capa de Abstracción de Datos, la Capa Lógica de Negocio, la Capa del Usuario y por último el proveedor de base de datos física.
Después tratamos las maneras en las que puedes conectarte al marco central tanto si quieres buscar, programar o personalizar tu módulo.
Finalmente, empaquetamos nuestro módulo para la distribución y lo instalamos en el portal.
Esperamos que esta guía te ayude a ser productivo a la hora de desarrollar módulos interactivos y muy funcionales. Asegúrate de visitar los recursos y de participar en la comunidad para estar siempre al corriente de las actividades más recientes del DNN.

Un saludo,
el Equipo de DotNetNuke