4 Pilares Portal web

La valoración de un Portal depende de cuatro variables Básicas contenido, usabilidad, eficiencia y diseño

Pieza fundamental de una página, y por extensión, del Sitio. Es el conjunto de información, y servicios, que el usuario puede encontrar en el.

Contenido

El grado de cumplimiento de las expectativas buscadas con el sitio por parte de la Organizacion. Por tanto, exige facilitar que el usuario ejecute las acciones perseguidas

Eficiencia

La disposición de los elementos de la página que la hace más o menos atractiva a la vista del usuario, y más o menos sencillos los procesos de hallar lo que el visitante requiere y a la vez lo que el dueño persigue.

Diseño

El grado de respuesta a las necesidades de información y servicio del usuario. Engloba:

- Que el usuario encuentre lo que busca y existe en el sitio.
- Que tenga vías rápidas de acceso a la información sin tener que dar demasiados "clic" del ratón o rellenar casillas de formulario.

Usabilidad

Es importante determinar el resultado de las combinaciones entre los distintos grados de estas variables descritas anteriormente:

Contenido pobre, nulo o escaso. Una debilidad en la información o servicios ofrecidos da poco valor al sitio entero ya que es lo que el usuario busca al entrar a la página, independientemente de si el diseño es atractivo o no. La usabilidad y la eficiencia carecen de sentido porque serán nulas en este caso. Partiendo de que el contenido sea rico o aceptable, fijémonos en las demás variables:

- Una eficiencia buena o aceptable con una usabilidad mala permite que el propietario recoja sus resultados, pero el usuario expresara quejas y malas experiencias.
- Un diseño pobre repercutirá en la eficiencia, porque este tipo de disposiciones se traducen en poca profesionalidad y dedicación. Especialmente el diseño de baja calidad repercute en la eficiencia ya que hace menos fiable la gestión y servicios de la empresa. Por tanto, aunque el diseño no sea fundamental para sitios personales que sólo buscan transmitir un determinado contenido (siempre y cuando la usabilidad sea alta), en el ámbito empresarial o profesional es un factor básico que altera los grados de usabilidad y eficiencia.
- Un diseño rico pero con usabilidad y eficiencia pobres dará iguales resultados que el contenido nulo: ni el usuario quedará satisfecho ni la organización conseguirá sus objetivos. Si una de las variables es buena y la otra mala, nos remitiremos a lo ya explicado.
- Un diseño atractivo, usabilidad media o alta, y eficiencia alta, junto con el contenido accesible o rico es lo ideal, ya que consigue un grado de profesionalidad suficiente, una satisfacción del usuario que le hace volver en el futuro (que a su vez es otro resultado imprescindible) y conseguir los objetivos de la página.

Soluciones

Como dijimos, el contenido es la base que hace que el usuario entre. Por tanto ha de dotarse a la página de toda la información imprescindible sobre el tema o temas tratados para que el usuario encuentre lo que se supone que la página indica que va a encontrar. Si no se puede abarcar todo, será preciso especializarse en un determinado nicho o parcela de la que sí se pueda ofrecer toda la información.

- Usabilidad. En este terreno no hay como ponerse en lugar del usuario y buscar dentro de la página las diferentes informaciones ofrecidas. Evaluaremos entonces las diferentes dificultades para acceder a ellas, y las pondremos más alcance del usuario en su caso. Además invitaremos a usuarios objetivos a opinar y evaluar, haciendo caso de sus advertencias. Una buena métrica es el conteo de Clicks necesarios para acceder a la información específica.
- Eficiencia. Evaluaremos por qué no se consiguen los resultados buscados: si es porque lo que perseguimos no está lo suficientemente visible y accesible entre la información, si el usuario no tiene motivaciones para elegirlo y sólo se queda en que él busca o sencillamente lo que ofrecemos no es atractivo comparado con otros sitios. Podemos hacer encuestas entre los visitantes para conocer estas respuestas. Puede ser que ofrezcamos tanta información que los objetivos queden escondidos. En este caso deberemos averiguar la forma de resaltarlo.
- Diseño. Si la página es personal y no busca grandes objetivos, más que transmitir nuestra información, basta con que el diseño esté lo suficiente bien distribuido como para que el lector acceda fácilmente a la información. Es decir, en páginas personales la usabilidad sí puede sustituir al diseño. En cuanto a páginas de empresas o profesionales el diseño es imprescindible para transmitir confianza y dedicación. En este caso es necesario ponerse en manos de profesionales del sector.

En consecuencia, es preciso no descuidar ninguna de estas cuatro variables si se pretende conseguir los resultados que compensen los esfuerzos de construir y mantener un Portal, ya sea para Internet o Intranet.

Héctor Insua (hinsua@gmail.com)

Quick Tips

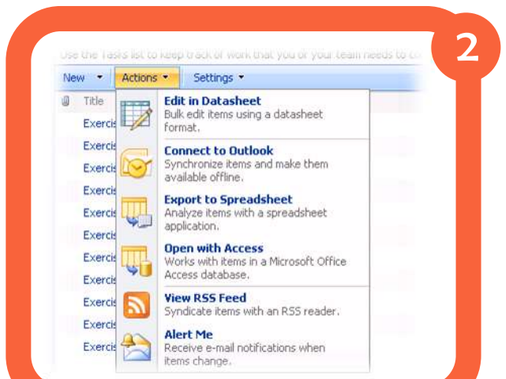
TIPS

Opciones de Editar en Hoja de Cálculo (Potente y Escondida)

- Una de las funciones más interesantes para gestionar Listas, es la de Editar en Hoja de Cálculo (Datasheet).

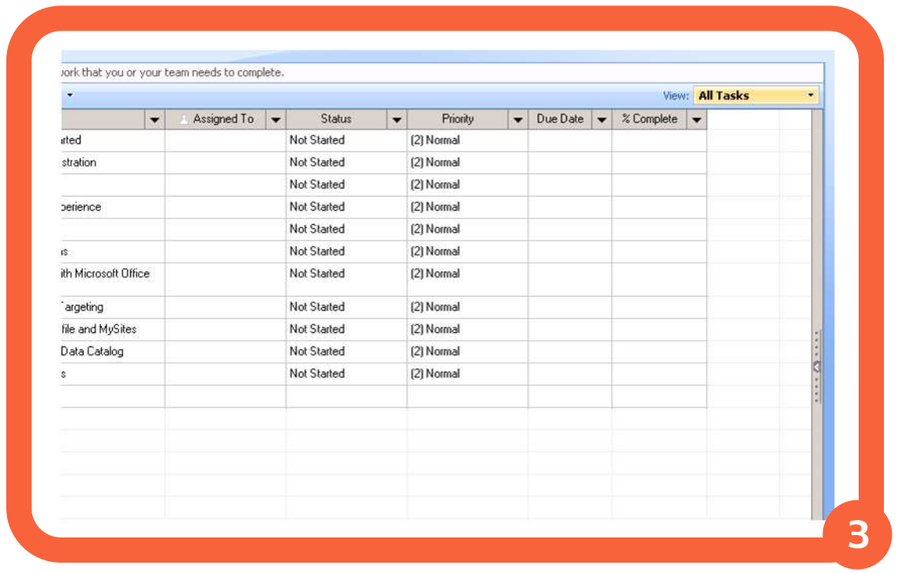
1

- Accedemos desde el Menú de Acciones:



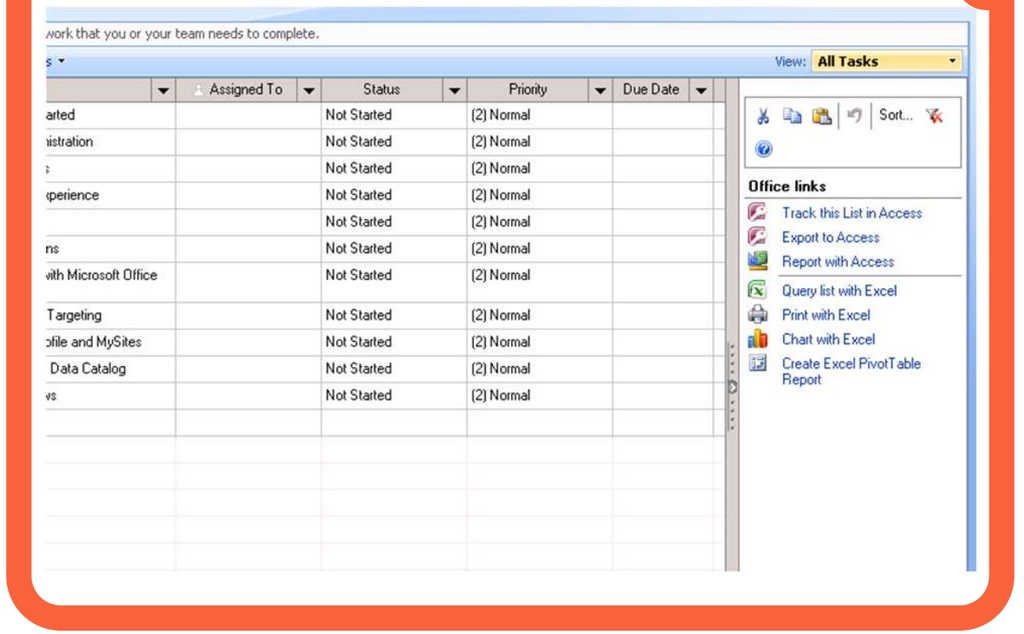
2

- Ahora bien, si prestamos atención, cuando estamos manejando esta vista, encontraremos un menú expansible sobre el sector derecho de nuestra pantalla.

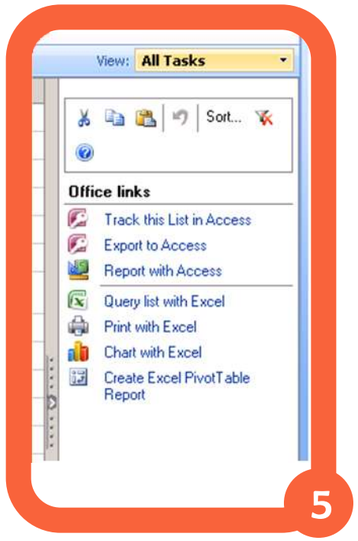


3

- Hacemos clic en el Icono y se nos desplegará la siguiente pantalla de opciones:



4



5

- Ese menú, tiene opciones muy interesantes, como Graficar los datos en Excel, realizar consultas o imprimir el contenido de la lista directamente desde Excel.

El completo menú de herramientas adicionales, permanece escondido en esta vista y en muchas ocasiones puede resultar en un ahorro de tiempos muy interesante, sobre todo al trabajar con listas de muchos elementos. Generalmente esta opción permanece “escondida”.

Hector Insua – hinsua@gmail.com

El Centro de Registros de MOSS

Para empresas de cualquier tipo y tamaño, conservar información de una forma segura y confiable se ha convertido en una labor imprescindible, tanto para su propio funcionamiento, como por las implicaciones legales que puede tener. Todos hemos visto en los últimos meses las consecuencias legales y financieras que puede tener la "perdida" voluntaria o involuntaria de información. SharePoint, en su función como depósito central de información empresarial, dispone de diferentes mecanismos para regular el acceso a ella en el sistema (autenticación y autorización), para registrar su utilización (auditorias, vea el articulo al respecto en el primer número de CompartiMOSS) y para preservarla en un lugar seguro e inaccesible a usuarios normales (Centro de Registros).

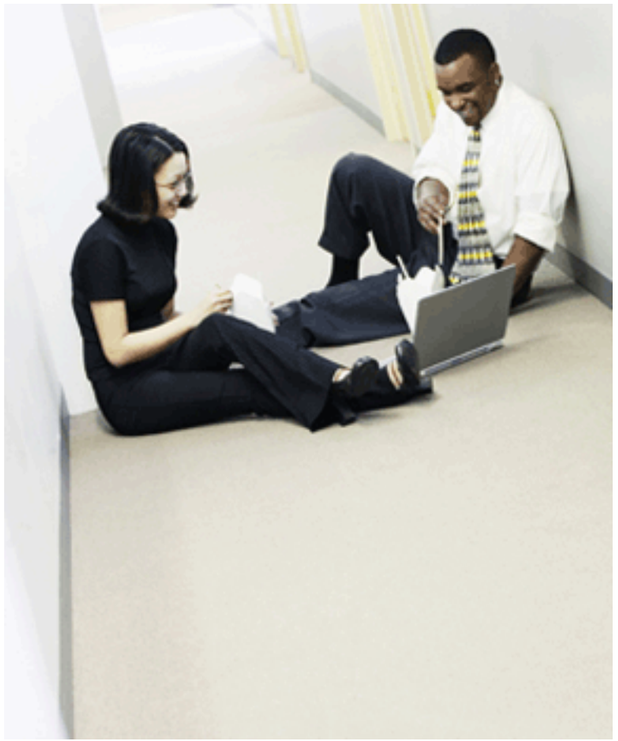
El Centro de Registros de Microsoft Office SharePoint Server (MOSS) es el componente diseñado para guardar información que se desea preservar fuera del uso cotidiano por una u otra razón. Por ejemplo, una empresa que genere informes financieros mensuales destinados a la declaración de impuestos; si en algún momento ocurre un conflicto con la oficina de impuestos, un juez puede exigir a la empresa que recobre los informes, y que garantice que no han sido alterados en el tiempo transcurrido.

En este caso, el departamento financiero de la empresa puede dirigir mensualmente los informes al Centro de Registros de su intranet de SharePoint y acoplarlos a una póliza de utilización (hasta cuando se deberán guardar, por ejemplo). Los informes serán guardados por SharePoint en un sitio a donde solo personal autorizado puede entrar, junto con su auditoria y pueden ser "bloqueados" en cualquier momento.

Nota: El Centro de Registros es un componente de MOSS, y no se encuentra disponible en Windows SharePoint Services (WSS).

El Centro de Registros de MOSS no es más que una plantilla para crear sitios en SharePoint, y se pueden crear tantos centros como sea necesario, pero solamente uno puede estar activo cada vez. El sitio para el Centro dispone de las Listas mínimas necesarias para su funcionamiento, junto con los mecanismos para encaminar la información hacia las Listas y todo el sistema de autorización necesario para que usuarios normales nó pueden verlo, pero sí enviarle documentos.

El diseño del Centro de Registros debe ser bien realizado desde un principio y Microsoft ha publicado extensa información al respecto ("Records Management Guide for Microsoft Office SharePoint Server 2007"



Creación del Centro de Registros

1. Cree una nueva Aplicación Web para el Centro de Registros, o utilice una Aplicación Web diferente a la Aplicación donde están las Librerías que van a hacer uso de él. El teoría, y según las especificación de Microsoft, esto no es necesario, pero la experiencia en la práctica indica que el Centro de Registros funciona óptimamente en una Aplicación Web aislada

2. En la nueva Aplicación Web cree una Colección de Sitios utilizando la plantilla "Centro de

3. En la Administración Central de SharePoint, en la pestaña "Administración de aplicaciones", bajo la sección "Conexiones de servicios externos", utilice el vinculo "Centro de registros" para configurar la Colección de Sitios acabada de crear. En la nueva pantalla, seleccione "Conectarse a un centro de registro" y en "Dirección URL" configure la dirección del sitio creado con el Centro de Registro, utilizando la sintaxis "http://NombreServidor/SubSitio/_vti_bin/officialfile.asmx" (por ejemplo, "http://MiServidorMoss/_vti_bin/officialfile.asmx" si el Centro de Registros ha sido creado bajo la raíz de la Colección de Sitios); note que la parte ".../_vti_bin/officialfile.asmx" es obligatoria. Finalmente seleccione un "Nombre para mostrar" que identifique al Centro de Registros para el usuario

Figura 1. Configuración del Centro de Registros en la Administración Central

Configuración de las Bibliotecas que van a usar el Centro de Registro

4

Cree una Directiva.

En el sitio en donde se va a crear la Librería (o Librerías; también es posible usar el Centro de Registros con Listas) que va a utilizar el Centro de Registros, vaya a "Acciones del sitio", "Configuración del sitio", "Directivas de colección de sitios" y cree una nueva Directiva. Para el ejemplo se ha creado una "Directiva notarial" en donde se ha usado "Habilitar Caducidad", con un periodo de retención basado en la Última modificación más un año y que el elemento se deberá Eliminar cuando haya transcurrido el tiempo configurado.

Esto le indicará al Centro de Registros que el documento deberá ser eliminado, sin pasar por la Papelera de reciclaje, un año después de la última modificación

5

Cree un Tipo de Contenido.

Vaya a "Acciones del sitio", "Configuración del sitio", "Tipo de contenido de sitio" y cree un nuevo Tipo de Contenido: para el ejemplo se ha creado un Tipo de Contenido llamado "Documento notarial" del Tipo de Contenido primario "Tipos de contenido de documento" - "Documento"

6

Si lo considera necesario, agréguele columnas al nuevo Tipo de Contenido creado. Desde la pantalla de configuración del Tipo, vaya a "Configuración de la directiva de administración de la información", seleccione "Utilizar una directiva de colección de sitios" y seleccione la Directiva creada en el punto 4

7

Regrese al sitio y cree una Biblioteca de Documentos (llamada "Documentos para registrar" en el ejemplo), utilizando todas las opciones por defecto

8

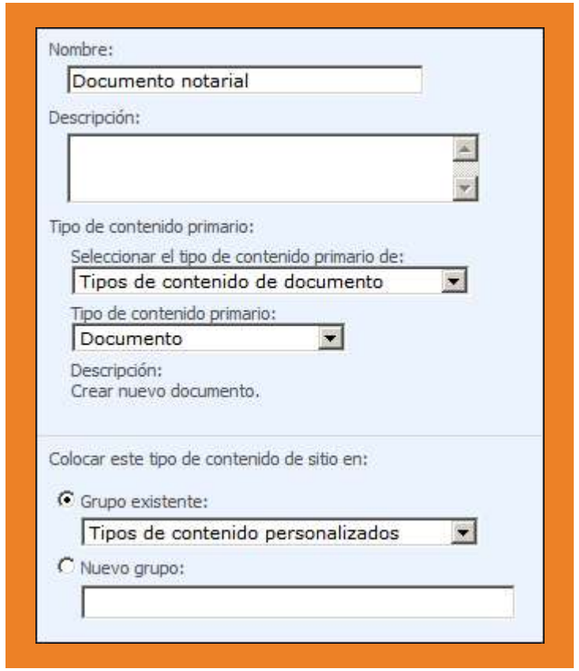
Desde la página de la Biblioteca, vaya a "Configuración", "Configuración de biblioteca de documentos", "Configuración avanzada" y seleccione que desea permitir la administración de tipos de contenido ("Si")

9

De regreso en la página de configuración de la Biblioteca, seleccione "Agregar a partir de tipos de contenido de sitio" bajo la sección "Tipos de contenido"; bajo "Tipos de contenido personalizados" encontrara el Tipo de Contenido "Documento notarial" creado en el punto 4; agréguelo al documento.

De la misma manera, seleccione el Tipo de Contenido "Documento" (creado por defecto) y elimínelo de la Biblioteca, de tal forma que quede solo el Tipo de Contenido acabado de crear

Figura 2. Creación del Tipo de Contenido



Configuración del Centro de Registros

10

Vaya al sitio que contiene el Centro de Registros (punto 2). Cree una nueva Biblioteca que contendrá los registros del tipo acabado de crear. En el ejemplo se ha creado una Librería llamada "Centro Notarial" utilizando todas las opciones por defecto de una Biblioteca

11

Vaya a la Lista "Distribución de registros" y cree una nueva entrada. En el "Titulo" utilice el nombre del Tipo de Contenido creado en el punto 4: "Documento notarial"; este campo le indica al Centro de Registros que todos los documentos creados con el Tipo de Contenido configurado deberán ser guardados en esta Librería. En "Ubicación" escriba el nombre de la Librería a donde se deben dirigir los documentos, en el caso del ejemplo, "Centro Notarial"

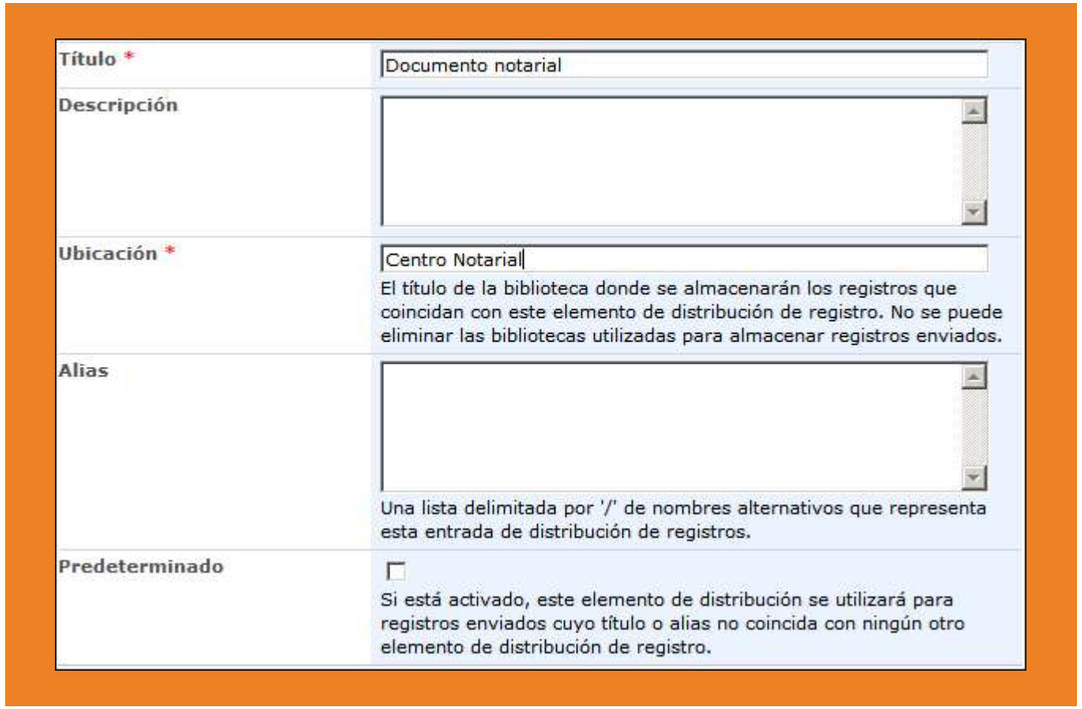


Figura 3. Configuración del Centro de Registros

Utilización del Centro de Registros

SharePoint está configurado desde este momento para guardar documentos creados en Librerías que utilizan el Tipo de Contenido "Documento notarial" en la Librería "Centro Notarial" del Centro de Registros. Vaya a la Librería creada en el punto 6 (Biblioteca "Documentos para registrar") y cree un nuevo documento. Desde el menú contextual del documento verá un vinculo a "El Centro de Registros" bajo "Enviar a"

Figura 4. Envió de un documento al Centro de Registros

Después de algunos instantes SharePoint indicara que la operación se ha realizado con éxito (o si ha habido algún error, de que tipo).

Si se va al Centro de Registros se podrá comprobar que en la Librería configurada se ha creado una carpeta en donde se encuentra una copia del documento enviado, con algunos caracteres únicos al final del nombre que lo identifican en el caso de que el documento sea subido más de una vez a la Librería (y evitan que sea sobre-escrito). También se encuentra un sub-folder llamado "Propiedades" que contiene un documento xml con el mismo nombre del documento en el que se puede encontrar toda la configuración del documento y Librería originales.

Configuración y uso de "Suspensiones"

12 El Centro de Registros contiene una Lista llamada "Suspensiones". Cree un nuevo elemento en ella ("Suspensión para documentos notariales" en el ejemplo), indicando una Descripción y Administrador si es necesario

Vaya a la Librería donde está el documento que se desea suspender (directamente en el documento, folders no se pueden suspender) y desde su menú contextual elija "Administrar suspensiones". Seleccione "Agregar a una suspensión" y seleccione la suspensión creada en el punto 12

De la misma forma se puede eliminar una suspensión del documento. En la página de la Librería se podrá ver el "Estado de la suspensión" en la columna respectiva

Si se selecciona una Suspensión específica en la Lista de Suspensiones, en su pantalla de propiedades se podrán ver informes sobre que documentos están usando la Suspensión.

Programación del Centro de Registros

Tanto utilizando el Modelo de Objetos de MOSS como sus WebServices es posible trabajar programáticamente con el Centro de Registros. El Modelo de Objetos provee seis Librerías (Microsoft.Office.RecordsManagement. Holds, InformationPolicy, PolicyFeatures, RecordsRepositoy, Reporting y SearchAndProcess) que provee todas las clases necesarias para interactuar con el Centro. El WebService "Official File Web Service" permite obtener alguna información sobre el servidor en donde se encuentra el Centro, sobre el Centro que está activo y subir documentos a él

Probablemente la labor más importante para usar programáticamente el Centro de Registros es subir documentos para guardar y poderlos suspender o reactivar:

Cargar documentos en el Centro de Registros usando el Modelo de Objetos

La siguiente rutina en una aplicación de consola sube un documento de una Librería en un sitio de MOSS al Centro de Registros (todo el código puede ser descargado desde el sitio de CompartiMOSS)

"InformacionAdicional" es una variable (string) necesaria para enviar información complementaria sobre el documento al Centro de Registros; "ResultadoOperacion" es otra variable (OfficialFileResult) que guardará el resultado de la subida del documento indicando si todo ha salido bien, o si ha ocurrido un error (los valores podrán ser fileChecedOut, FileRejected, InvalidConfiguration, MoreInformation, NotFound, Succes o UnknownError).

Dos objetos del tipo SPSite y SPWeb son creados para poder referenciar el documento ("miDoc" del tipo SPFile) a guardar en el Centro. Finalmente, el método "SendToOfficialFile" de la clase SPFile se encarga de enviar el documento seleccionado al Centro de Registros, y devuelve el resultado de la acción.

Nota: El código necesita una referencia a "Windows SharePoint Services" (Microsoft.SharePoint.dll) y una directiva a "using Microsoft.SharePoint;".

```
static void EnviarDocumentoACentroRegistrosMO()
{
    try
    {
        string InformacionAdicional = string.Empty
        OfficialFileResult ResultadoOperacion;

SPSite miSitio = new SPSite("http://NombreServidor");
        SPWeb miWeb = miSitio.OpenWeb();

        SPFile miDoc = miWeb.GetFile("LibreriaDoc/NombreDoc.docx");

        ResultadoOperacion = miDoc.SendToOfficialFile(out InformacionAdicional);

        Console.WriteLine("Estado de la Operacion: " + ResultadoOperacion);
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error - " + ex.ToString());
    }
}
```

Cree una referencia Web en Visual Studio a “_vti_bin/officialfile.asmx” (llamada “CentroRegistrosWS” en el ejemplo). Los primeros renglones de código crean el objeto necesario, localizan el archivo asmx en el servidor y Aplicación Web en donde el Centro de Registros se encuentra, y le indica que utilice las credenciales apropiadas.

Los objetos SPSite, SPWeb y SPFile han sido creados en el ejemplo para tener una referencia al documento a guardar, pero por estar usando un WebService, probablemente el documento se encontrará en un sistema externo a SharePoint, por lo que la referencia será obtenida en alguna otra forma. Lo importante es tener una imagen del documento en forma de un byte array (“miDocByte” en el ejemplo).

La variable “PropiedadesRepositorio” es un array que contiene información sobre el documento a guardar, y que será encontrada posteriormente en el archivo correspondiente en el folder “Propiedades” del Centro; en el array se pueden crear tantas propiedades como sea necesario, y pueden contener cualquier tipo y contenido deseable.

Finalmente, el método “SubmitFile” utiliza la corriente de bytes (stream) con el documento, la variable con sus propiedades, el nombre del Tipo de Contenido de la Librería del documento, su dirección URL y el usuario que lo sube como parámetros para enviar el documento al Centro de Registros. El método entrega a su vez el resultado de la operación (Success, MoreInformation, InvalidConfiguration, InvalidArgument, InvalidUser, NotFound, FileRejected o UnknownError).

Suspender documentos usando el Modelo de Objetos

Solamente el Modelo de Objetos provee los métodos necesarios para suspender o reactivar documentos en el Centro de Registros, aunque es posible crear un WebService para el objeto sin grandes problemas. La siguiente rutina suspende un documento ya presente en una Librería del Centro:

```
static void SuspenderRegistro()
        {
            SPSite miSite = new SPSite("http://NombreServidor:Puerto");
            SPWeb miWeb = miSite.OpenWeb();

            SPList miListaRetension = Hold.GetHoldReportsList(miWeb);
            Console.WriteLine(miListaRetension.Title);

            int SuspensionID = 0;
            foreach(SPListItem unaRetension in miListaRetension.Items)
            {
                if(unaRetension.Name.Contains("Suspension para documentos notariales")
= = true)
                    SuspensionID = unaRetension.ID;
            }

            SPList CentroNotarial = miWeb.Lists["Centro Notarial"];
            SPListItem DocASuspender = null;
            foreach(SPListItem unDoc in CentroNotarial.Items)
             {
                if (unDoc.Name.Contains("Un documento para el Registro") = = true &&
                    unDoc.Name.Contains(".docx") = = true)
                        DocASuspender = unDoc;
             }

             Hold.SetHold(SuspensionID, DocASuspender, "Hecho con codigo");
        }
```

Nota: Fuera de la referencia al dll de SharePoint, el código necesita una referencia a “Microsoft Office Server DLC component” (Microsoft.Office.Policy.dll) y una directiva a “using Microsoft.Office.RecordsManagement.Holds;”

Primero se crean objetos SPSite y SPWeb conteniendo el Centro de Registros. La variable “miListaRetension” (SPList) es una referencia a la Lista de Retención utilizada por Centro, y el bucle siguiente tiene por objetivo solamente encontrar el identificador (integer) de una de las pólizas de suspensión (para el ejemplo se usa el nombre de la suspensión, pero probablemente alguna otra forma de encontrar el identificador sea más efectiva).

De la misma forma se crea una referencia a la Librería en donde se encuentra el documento a suspender (“CentroNotarial”) y el objeto que contiene al documento (“DocASuspender” del tipo “SPListItem”). Finalmente, el método “SetHold” de la clase “Hold” perteneciente al NameSpace “Microsoft.Office.RecordsManagement.

Holds” permite suspender el documento y agregar un comentario. Un Override de SetHold permite suspender todos los documentos contenidos en un “SPListItemCollection”, que puede ser utilizado para suspender documentos masivamente.

El método “ReleaseHold” que utiliza como parámetros de entrada el identificador de la suspensión, una referencia al SPWeb de la Librería en donde se encuentra el documento y un comentario, permiten eliminar suspensiones de documentos.

En el Modelo de Objetos existe un método “RemoveHold”, pero es solamente para uso interno de SharePoint.

Conclusión

El Centro de Registros es un componente de SharePoint muy importante para empresas que necesitan guardar de forma segura información crucial o legalmente sensitiva.

El Centro garantiza el acceso solamente a personal autorizado, y mantiene la información aislada de los sitios utilizados en el uso cotidiano del Portal.

El Modelo de Objetos de MOSS y uno de sus WebService permiten el acceso programático al Centro de Registros, permitiendo enviarle documentos y manipular su suspensión o reactivación.

Gustavo Velez
gustavo@gavd.net

Validación de datos en la Data Form Web Part

Una de las grandes características de la plataforma SharePoint es su carácter extensible y abierto, de manera que permite desarrollar y encajar "casi" cualquier tipo de solución, así como resolver problemáticas de negocio realmente complejas.

A la hora de crear soluciones en SharePoint, tenemos dos entornos de desarrollo posibles: SharePoint Designer 2007 (SD 2007) o bien Visual Studio 2005 / 2008. Sin duda, SD 2007 permite crear rápidamente vistas avanzadas y formularios en una solución SharePoint.

Y la clave de este desarrollo rápido y sin código está en la versatilidad y juego que nos da la Data Form Web Part.

Esta Web Part tan especial permite consumir datos de orígenes de datos diversos como: listas y bibliotecas de SharePoint, fuentes RSS, datos de una base de datos, servicios web o bien combinaciones de estos.

Si nos centramos en la interoperabilidad con bases de datos, la Data Form Web Part nos permite mostrar datos en modo vista múltiple, vista simple o bien modelar formularios de vista múltiple, vista simple o de inserción de datos.

Figura 1: Ejemplo de aplicación modelada con SD 2007.

Si nos centramos en la funcionalidad de inserción de datos [1], SharePoint no proporciona por defecto mecanismos para la validación de los datos que el usuario está introduciendo en los distintos campos del formulario. Sin embargo, estas validaciones son necesarias para implementar verdaderas aplicaciones de negocio.

Por otro lado, a veces es necesario que las validaciones no penalicen el rendimiento de la aplicación o bien incumplan determinadas políticas de seguridad, por lo que hay que evitar realizarlas utilizando código de servidor.

Llegados a este punto, se nos plantea un gran interrogante: ¿Cómo podemos realizar estas validaciones sin penalizar el rendimiento e incumplir las políticas de seguridad?

La respuesta está en la capacidad que nos proporciona JavaScript para realizarlas en el cliente sin desmejorar el rendimiento de nuestras aplicaciones de negocio.

Si inspeccionamos de nuevo nuestra aplicación de ejemplo (Figura 1), nos daremos cuenta de que se trata de una página ASP.NET que permite insertar datos en la BD SQL Server [2] mediante un formulario de inserción.

Ahora bien, y retomando el párrafo anterior, por defecto la Data Form Web Part configurada en modo New Item Form no implementa ningún mecanismo de validación.

Para construir esta funcionalidad de validación, utilizaremos por un lado SD 2007 como entorno de desarrollo, y por otro JavaScript como lenguaje de modelado. Además, tendremos que tener en cuenta que este código irá encajado dentro de etiquetas XSL (Extensible Stylesheet Language), por lo que habrá que prestar especial atención al uso de caracteres extraños no tolerados por XSL.

Modelando la solución

Tal y como se comenta en [1] y [3], el principal escollo que hay que superar para implementar el mecanismo de validación de datos consiste en determinar cómo añadir un evento a alguno de los controles de tipo botón que genera la Data Form Web Part. Estos controles, que son creados en el momento en el que estamos insertando una vista de datos desde SD 2007 de acuerdo a la opción New Item Form [2], no exponen de manera pública un evento que se pueda programar. De acuerdo a [1], la solución a este inconveniente pasa por utilizar la función AttachEvent de JavaScript que nos permite, para Internet Explorer (¡restricción muy importante!), registrar eventos para cualquier tipo de control y especificar la función JavaScript que se encargará de manejar dicho evento

(Nota: Para el caso Firefox la función JavaScript equivalente es addEventListener). Una vez que sabemos cómo registrar un evento para un control, el siguiente paso es elegir el evento que vamos a registrar. Dado que una vez cargado, el formulario ya habrá pasado por el proceso de Submit, no podremos registrar el evento click y por lo tanto no se podrá cancelar el proceso (en este caso se trata del proceso de guardar los datos en la BD). Ahora bien, como alternativa, podemos utilizar otro evento típico para controles de tipo botón, hiperlink o imagen como es onmouseup o bien onmouseover (que es el que voy a utilizar en el artículo).

Una vez que sabemos cómo registrar el evento para un control y hemos elegido el evento que vamos a registrar, necesitamos una forma de obtener el control especifico al que vamos a añadirle este evento. En este caso, el modelo DOM (Document Object Model) de JavaScript expone los métodos o . El primero nos devuelve la referencia al objeto que tiene como identificador (id) el valor que recibe.

El segundo devuelve dicha referencia al objeto que tiene como nombre (name) el valor que recibe.

Finalmente, necesitamos saber dónde colocar todo el código JavaScript que vamos a utilizar para registrar los eventos de los controles, así como añadirlo en el evento Load de la carga del formulario.

Para el primer punto, se puede decir que no hay una regla fija, sino que simplemente hay que pensar en la estructura de XSL y utilizar un poco de experiencia e intuición para saber dónde colocar todo el código JavaScript con el que implementamos la lógica de validación.

En el caso que estamos modelando, se ha colocado el código JavaScript a continuación de los controles de tipo botón para los que queremos registrar el evento (botones Save y Cancel) y antes del tag </xsl:template>

(Nota: Para comprender mejor la estructura XSL de la Data Form Web Part, se recomienda la lectura de [4]). Por inspección del markup de la página ASP.NET, tendríamos que el código JavaScript tendría que ser insertado en la zona subrayada en el siguiente listado (Listado 1):

```
<xsl:template name="dvt_1.formactions">
        <td nowrap="nowrap" class="ms-
vb">
                <input type="button"
                value="Save"
name="btnSave" onclick="BLOCKED SCRIPT
{ddwrt:GenFireServerEvent('__commit')}" />
                        </td>
                                <td
nowrap="nowrap" class="ms-vb"
width="99%">
                                        <input
type="button"

value="Cancel" name="btnCancel"

onclick="BLOCKED SCRIPT
{ddwrt:GenFireServerEvent('__cancel')}" />
                        </td>
                <script
language="javascript"
type="text/javascript">
                        //Código JavaScript
                </script>
</xsl:template>
```

Para el segundo de los puntos, es necesario añadir todo este código JavaScript a la pila de funciones JavaScript que dispara el evento Load de la página dónde se embebe el formulario.
Para ello, utilizaremos la siguiente función JavaScript (propia de SharePoint):

```
_spBodyOnLoadFunctionNames.push("EventButton");
```

Listado 2: Función que permite añadir una función a la pila de funciones JavaScript.

Esta función es la que aparece en la cabecera de todo el código JavaScript que vamos a utilizar para implementar el mecanismo de validación.

Una vez modelada la validación, vamos a ver como implementarla.

Implementando la validación

Lo primero que tenemos que hacer es llamar a la función JavaScript anterior que se encarga de añadir la función EventButton a la pila de funciones JavaScript de la página. Una vez que hemos hecho esto, creamos la función y en la misma tenemos que realizar los siguientes pasos:

- Declarar dos objetos que se correspondan con los controles (los botones Save y Cancel del formulario) a los que les vamos a registrar los eventos.
- Localizar estos objetos dentro de toda la colección de controles del formulario. Para ello utilizaré la función getElementById, puesto que para el caso de los botones SharePoint devuelve un ID limpio y conocido (lo que no sucede con los controles de tipo caja de texto).
- Registrar mediante attachEvent el evento onmouseover para cada control y especificar la función que se encargará de manejarlo.

La implementación de estos tres primeros pasos es la que se recoge en el Listado 3.

```
function EventButton()
{
        PonerFocoInicial();
        var objAceptar=document.getElementById("btnSave");
        objAceptar.attachEvent("onmouseover",Aceptar);
        var objCancelar=document.getElementById("btnCancel");
        objCancelar.attachEvent("onmouseover",Cancelar);
}
```

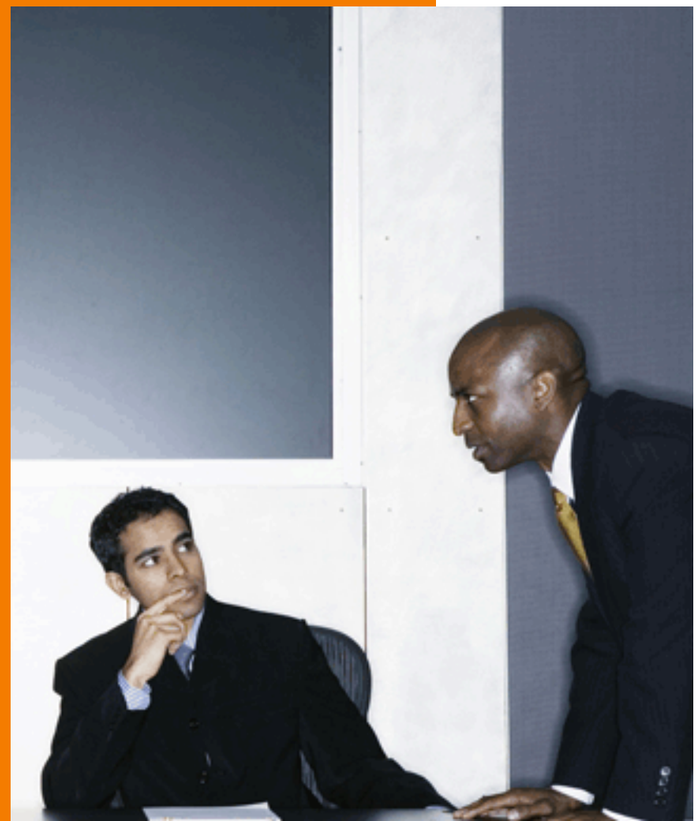
Listado 3: Implementación de la función EventButton.

Nota: En el código he añadido una función adicional PonerFocoInicial() que simplemente pone el foco del cursor en una de las cajas de texto del formulario.

Una vez que hemos registrado los eventos y añadido los correspondientes manejadores, pasaremos a definir e implementar estos últimos:

Manejador Cancelar, que simplemente mostrará por pantalla un mensaje informativo (utilizando alert()).

```
function Cancelar()
{
        var ICancelarEvento=false;
        alert("Cancelada la inserción de un nuevo curso");
        return ICancelarEvento;
}
```



Listado 4: Codificación del manejador Cancelar.

Manejador Aceptar, que es dónde introduciremos la lógica de validación. Lógicamente es aquí dónde empiezan las complicaciones, y dónde más obstáculos y problemas de todo tipo nos vamos a encontrar, puesto que para buscar el control a comprobar no tenemos una referencia clara de identificador tal y como ocurre con los controles de tipo botón. De hecho, el ID de una caja de texto tiene la siguiente forma (nada fácil de conocer de antemano):

```
ctl00$m$g_5437d929_64a3_4a68_be70_f2452c6f6981$ff2_new
```

Sin embargo, si nos fijamos en el ID, podremos comprobar que hay una parte que se corresponde con el ID que aparece en SD 2007 cuando examinamos los controles que ha generado la Data Form Web Part: ff2_new. Por lo tanto, nos podemos aprovechar de esta circunstancia para utilizar la función getTagFromIdentifierAndTitle que aparece en [3] (si bien, y como se podrá comprobar, ha sido necesario modificar e implementar de modo personalizado la función getTagFromIdentifierAndTitle para que funcione correctamente encajada dentro del XSL de la Data Form Web Part).

La implementación final del manejador Aceptar es la que se muestra en el siguiente listado (Listado 5):

```
function Aceptar()
{
        var ICancelarEvento=false;
        var lsNombreProducto=null;

lsNombreProducto=document.getElementById(getTagFromIdentifierAndTitle("input","ff2_new",""));
        if(lsNombreProducto.value==="")
        {
                alert("EL campo Nombre Producto no puede estar vacío!");
                lsNombreProducto.focus();
        }
        return ICancelarEvento;
}
```

Listado 5: Implementación del manejador Aceptar.

Cómo se puede deducir del listado, la clave de la función anterior está en encontrar el control a validar está de nuevo en el método getElementById dentro de la cual se llama a la función getTagFromIdentifierAndTitle que recibe como parámetros el tipo de control y la cadena de texto a buscar entre los ID de la colección de controles del formulario. La codificación de dicha función es la que se muestra a continuación (Listado 6):

```
function getTagFromIdentifierAndTitle(tagName, identifier, title)
{
  var len = identifier.length;
  var tags = document.getElementsByTagName(tagName);
  for (var i=0;i&amp;lt;tags.length;i++) {
    var tempString = tags[i].id;

    if (tags[i].title == title &amp;amp;&amp;amp;
          (identifier == "" || tempString.indexOf(identifier) == tempString.length - len))
    {
      return tempString
    }
  }
}
```

Listado 6: Codificación de la función getTagFromIdentifierAndTitle.

En este caso, aparte de la sencillez del algoritmo de búsqueda del control cuyo nombre contiene un tag concreto, lo más peculiar de esta función es que debido a las características propias de XSL, en el bucle for se ha tenido que usar la representación estándar HTML del símbolo < (i<) pues SD 2007 y SharePoint no admiten que especifiques este símbolo a no ser que indique un etiqueta real. Dentro del bucle for, que recorre todos los controles del formulario (y que se almacenan en el objeto tags, completado mediante el método), simplemente se busca y devuelve aquel control cuyo ID contenga la palabra clave que se ha pasado a la función (por cierto, de nuevo el lector habrá notado que en la sentencia if se ha reemplazado && por && por el mismo motivo comentado para el símbolo <).

- Función PonerFocoInicial, esta es la más sencilla de todas una vez explicadas las funciones anteriores. La implementación de la misma es la que se detalla en el Listado 7.

```
function PonerFocoInicial()
{
        var ICancelarEvento=false;
        var lsNombreProducto=null;

lsNombreProducto=document.getElem
entById(getTagFromIdentifierAndTitle("in
put","ff2_new",""));
        if(lsNombreProducto.value==="")
        {
                lsNombreProducto.focus();
        }
        return ICancelarEvento;

}
```

Listado 7: Implementación de la función PonerFocoInicial.

Como vemos, utilizando el método getElementById buscamos el control concreto en el que queremos poner el foco inicial, y una vez encontrado forzamos mediante la función Focus().

Juntándolo todo y probando la validación

Una vez que tenemos modelada e implementada la validación, ya estamos listos para probar si funciona de manera correcta. Para ello, podemos optar por abrir nuestra solución con Internet Explorer o bien directamente desde SD 2007 haciendo un Preview de la página:

Al pasar el ratón por el botón Save, aparece un mensaje indicando que el usuario no ha introducido un cierto dato definido como obligatorio (Figura 3).

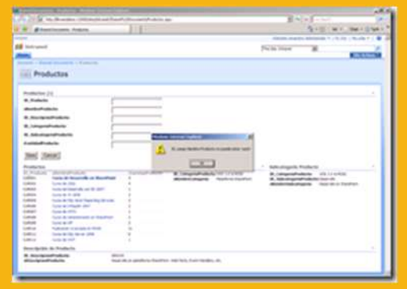


Figura 3: Prueba de la validación del botón Save.

Al pasar el ratón por el botón Cancel, aparece un mensaje indicando que el usuario ha cancelado la operación de inserción (Figura 3).

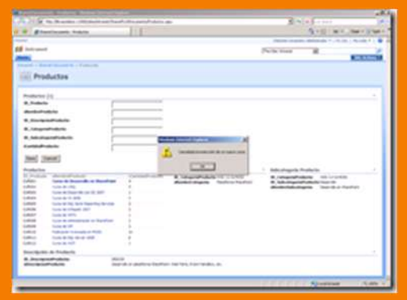


Figura 3: Prueba de la validación del botón Cancel.

Finalmente, para concluir este artículo, el código completo que tendríamos que incluir a continuación de los controles de tipo botón (y antes de </xsl:template) es el que a continuación se muestra (Listado 8):

```
<xsl:template name="dvt_1.formactions">
        <td nowrap="nowrap" class="ms-vb">
                <input type="button" value="Save" name="btnSave"
                        onclick="BLOCKED SCRIPT {ddwrt:GenFireServerEvent('__commit')}" />
                </td>
                <td nowrap="nowrap" class="ms-vb" width="99%">
                        <input type="button" value="Cancel"
                        name="btnCancel" onclick="BLOCKED SCRIPT
{ddwrt:GenFireServerEvent('__cancel')}" />
                </td>
<script language="javascript" type="text/javascript">
// Esta función es propia de Sharepoint y nos permite poner
//una función nuestra dentro de la pila de funciones que serán
//disparadas en el evento Load del mismo cuando
//se carge.
_spBodyOnLoadFunctionNames.push("EventButton");
function EventButton()
{
        PonerFocoInicial();
        var objAceptar=document.getElementById("btnSave");
        objAceptar.attachEvent("onmouseover",Aceptar);
        var objCancelar=document.getElementById("btnCancel");
        objCancelar.attachEvent("onmouseover",Cancelar);
}

function getTagFromIdentifierAndTitle(tagName, identifier, title)
{
  var len = identifier.length;
  var tags = document.getElementsByTagName(tagName);
  for (var i=0;i&lt;tags.length;i++) {
    var tempString = tags[i].id;

    if (tags[i].title == title &amp;&amp; (identifier == "" || tempString.indexOf(identifier) ==
tempString.length - len))
    {
      return tempString
    }
  }
}
function Aceptar()
{
        var ICancelarEvento=false;
        var lsNombreProducto=null;
        lsNombreProducto=
document.getElementById(getTagFromIdentifierAndTitle("input","ff2_new",""));
        if(lsNombreProducto.value==="")
        {
                alert("EL campo Nombre Producto no puede estar vacío!");
                lsNombreProducto.focus();
        }
        return ICancelarEvento;
}
function Cancelar()
{
        var ICancelarEvento=false;
        alert("Cancelada la inserción de un nuevo curso");
        return ICancelarEvento;
```

```
}
function PonerFocoInicial()
{
	var lCancelarEvento = false;
	var lsNombreProducto = null;
	lsNombreProducto =
document.getElementById(getTagFromIdentifierAndTitle("input","ff2_new",""));
	if(lsNombreProducto.value == "")
	{
		lsNombreProducto.focus();
	}
	return lCancelarEvento;
}
	</script>
</xsl:template>
```

Listado 8: Implementación completa de la funcionalidad de validación.

Conclusiones

Sin duda, SD 2007 es una alternativa muy válida para el desarrollo rápido de soluciones sencillas de integración en plataforma SharePoint.

La clave de esta integración está en la flexibilidad y posibilidades de la Data Form Web Part.

Ahora bien, en escenarios complejos de integración que impliquen validaciones de datos sin penalizar el rendimiento y sin comprometer la política de seguridad, es necesario recurrir a funciones y funcionalidades clásicas y estándar de JavaScript y DOM para realizar estas validaciones en el cliente, de una manera intuitiva y sin introducir tiempos de respuesta no tolerables para el usuario final.

Referencias

[1] Formularios personalizados de listas en SharePoint – Parte III. Fabián Imaz Siderys Elite Software

[2] WSS 3.0 & MOSS: Construyendo vistas avanzadas con SharePoint Designer 2007 (II). Juan Carlos González Martín. Centro de Innovación en Integración.

[3] Using Javascript to Manipulate a List Form Field. Rob Howard, Microsoft SharePoint Designer Team Blog.

[4] Spaghetti Code: Estudiando la estructura del xsl de un DataForm WebPart.

Centro de Innovación en Integración (CIIN)
Juan Carlos González Martín
Arquitecto de Soluciones
MVP de MOSS

ENLOQUECIENDO

al Elemento Web Editor de contenido

La WebPart por defecto “Elemento Web Editor de contenido” permite incluir cualquier tipo de código HTML o JavaScript en una página de SharePoint:

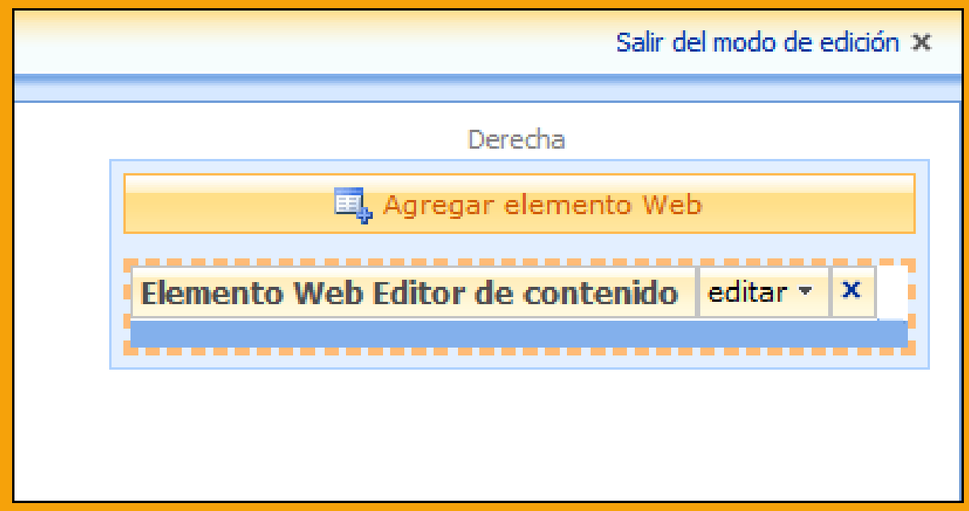
Figura 1. El Elemento Web Editor de contenido

Para enloquecerlo, abra la pantalla de “Editor de código fuente” de la WebPart e introduzca las siguientes dos líneas de código:

```
<script>
< /script>
```

Note que entre “<” y “/script> hay un espacio vacío. El resultado es que la WebPart “se come” todas las WebParts que hay en la zona, impidiéndolas ver, elimina el panel de configuración y, si se usa el vinculo “Salir del modo de edición” y se intenta regresar a “Editar pagina” por medio del menú “Acciones del sitio”, aparece un error terminal en la pagina

Figura 2. Error en el Elemento Web Editor de contenido



Timer JOBS

1 Introducción/motivación

Muchas veces debemos extender SharePoint para que se adapte a los requerimientos de un proyecto y no nos queda otra opción que abrir Visual Studio y compilar. En mi opinión, SharePoint es un producto donde se cumple el "Principio de Pareto". Así el 80% de la funcionalidad de un proyecto concreto se cumple con la configuración y adaptación del producto a través de la interficie de usuario o a través de SharePoint Designer, lo que no requiere un gran esfuerzo y el riesgo es mínimo. En cambio, para cubrir el 20% restante de los requerimientos funcionales de un proyecto, es necesario muchas veces desarrollar componentes a medida (web parts, flujos de trabajo, trabajos temporizados, ...).

De aquí, la importancia de minimizar los desarrollos y sacar el máximo rendimiento a la funcionalidad que el producto ofrece. Para ello es necesario conocer a fondo el producto para evitar desarrollos costosos que se podrían sustituir por pequeñas extensiones de SharePoint.

En el siguiente artículo trataremos el desarrollo de tareas programadas en SharePoint. Las tareas programadas o Timer Jobs nos permiten ejecutar un proceso de forma periódica. Una de las mayores ventajas que nos ofrecen respecto a otras alternativas como servicios Windows o tareas programadas son que el despliegue se realiza de forma sencilla a través del comando stsadm, el cual se encarga de distribuirlo a lo largo de nuestro conjunto de servidores.

Por el contrario la versión actual de SharePoint no ofrece una forma fácil de gestión y configuración de las tareas programadas, por lo que es necesario crear nuestros propios mecanismos para su administración. Este ejemplo no pretende solucionar ninguna necesidad real sino que intenta mostrar una solución simple pero completa donde intervengan el mayor número de componentes.

2 Entorno de desarrollo

Es primordial disponer de un entorno de desarrollo o test donde poder desarrollar nuestros componentes y probarlos sin ningún riesgo. Yo soy partidario de disponer de una máquina virtual con todas las herramientas necesarias y desarrollar en ella, de modo que desarrollamos nuestros componentes en el servidor.

Aunque es posible desarrollar fuera del servidor de SharePoint, es mucho más cómodo si todos los desarrolladores tienen su propio servidor, ya que pueden de este modo probar su código en tiempo real del mismo modo que probamos una aplicación Windows. En un entorno de desarrollo SharePoint 2007 estas son las herramientas básicas con las que contamos:

- Visual Studio 2005/2008
- Visual Studio 2005 Extensions for Windows SharePoint Services
- Visual Studio 2005 extensions for .NET Framework 3.0 (Windows Workflow Foundation)

A parte de estas cada uno de nosotros es libre de usar aquellos complementos y herramientas que le faciliten su labor diaria.

3 Manos a la obra

El primer paso consiste en estructurar el proyecto.

En nuestro caso este estará compuesto por una sola librería con el código de nuestro trabajo temporizado.

Por otro lado existirá el proyecto que generará la solución WSP.

Proyecto TimerJob con FeatureReceiver

El objetivo de la solución es crear una nueva tarea en nuestro portal que inserte un elemento en una lista cada cierto tiempo.

Para especificar el nombre del servidor y otras configuraciones del proyecto es recomendable usar la clase que permite almacenar el estado de los objetos de forma permanente.

En este caso no la usaremos y “hardcodearemos” el valor de esas variables.

El proceso para crear la nuestro trabajo temporizado es el siguiente:

1. Crearemos un nuevo proyecto de tipo Class Library donde crearemos las clases necesarias.

2. Borramos la clase Class1.cs y creamos una nueva clase CompartiMOSSTimerJob.cs

3. Añadimos la referencia Microsoft.SharePoint.dll (C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\ISAPI\Microsoft.SharePoint.dll)

4. Añadimos una nueva clave al proyecto y firmamos con ella el proyecto. El motivo no es otro que el poder deployar las librerías en la GAC de los servidores.

5. Copiamos el siguiente código en la clase:

```
namespace Raona
{
   class CompartiMOSSTimerJob : SPJobDefinition
   {
      //Constructor por defecto
      public CompartiMOSSTimerJob()
         : base()
      {
      }
      public CompartiMOSSTimerJob(string jobName, SPWebApplication webApplication,
String description)
            : base(jobName, webApplication, null, SPJobLockType.ContentDatabase)
      {
         this.Title = description;
      }

      //Ejecución
      public override void Execute(Guid contentDbId)
      {
         //Abrimos el sitio
            SPWebApplication webApplication = this.Parent as SPWebApplication;
        SPContentDatabase contentDb = webApplication.ContentDatabases[contentDbId];
         SPWeb web = contentDb.Sites[0].RootWeb;

         //Abrir la lista e insertar un elemento
         SPList list = web.Lists["CompartiMOSS"];
         SPListItem item = list.Items.Add();

         //Asignar valores al elemento
            item["Title"] = "Inserción: " + DateTime.Now.ToShortDateString() + " " +
DateTime.Now.ToShortTimeString();

         //Salvar estado
         item.Update();

         //Liberar recursos
         web.Dispose();
      }
   }
}
```

6. Una vez tenemos el trabajo temporizado implementado debemos instalarlo a través del modelo de objetos. Para ello usaremos la clase SPFeatureReceiver que permite la ejecución de código al operar con una feature de SharePoint 2007.

Así creamos una nueva clase TimerJobInstaller con el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Administration;

namespace Raona
{
    class TimerJobInstaller : SPFeatureReceiver
    {
        const string NombreTimerJob = "CompartiMOSS Timer Job";

        public override void FeatureInstalled(SPFeatureReceiverProperties properties)
        {
        }

        public override void FeatureUninstalling(SPFeatureReceiverProperties properties)
        {
        }

        //Activación de la feature
        public override void FeatureActivated(SPFeatureReceiverProperties properties)
        {
            //Abrimos el sitio
            SPSite site = properties.Feature.Parent as SPSite;
            SPWeb web = site.OpenWeb();

            //Crear la lista
            web.Lists.Add("CompartiMOSS", "Lista de ejemplo", SPListTemplateType.GenericList);

            //Solo admitimos un trabajo temporizado. Por lo tanto borramos si había alguno previamente.
            foreach (SPJobDefinition job in site.WebApplication.JobDefinitions)
            {
                if (job.Name == NombreTimerJob)
                    job.Delete();
            }

            // Instalación del job
            CompartiMOSSTimerJob NewJob = new CompartiMOSSTimerJob(NombreTimerJob,
site.WebApplication, "Ejecución de Prueba");

            //Recurrencia
            NewJob.Schedule = SPSchedule.FromString("every 1 minutes");

            NewJob.Update();

            //Liberar recursos
            web.Dispose();
            site.Dispose();
        }
```

```
    }

    //Desactivación de la feature
    public override void FeatureDeactivating(SPFeatureReceiverProperties properties)
    {
        //Abrimos el sitio
        SPSite site = properties.Feature.Parent as SPSite;
        SPWeb web = site.OpenWeb();

        //Borramos el job
        foreach (SPJobDefinition job in site.WebApplication.JobDefinitions)
        {
            if (job.Name == NombreTimerJob)
                job.Delete();
        }

        //Borramos la lista
        web.Lists["CompartiMOSS"].Delete();

        //Liberar recursos
        web.Dispose();
        site.Dispose();
    }}}
```

Proyecto para deploy

Llegados a este punto debemos empaquetar la solución en un fichero WSP para desplegarlo en nuestro conjunto de servidores.

Un paquete solución SharePoint no es más que un fichero CAB renombrado a WSP, por lo tanto necesitaremos tener instalada la utilidad en nuestro entorno de desarrollo (la generación de WSP ya está incluida en las extensiones para Visual Studio 2005).

Lo que a continuación mostramos es el proceso para integrar la generación del fichero WSP dentro de Visual Studio 2005.

1. Creamos un nuevo proyecto (Class Library) y borramos la clase Class1.cs.

2. Creamos los siguientes ficheros dentro de la solución:

2.1. WSPPackage.ddf: Especifica qué ficheros se incluyen en el paquete y su estructura de directorios:.

```
Option Explicit
.Set DiskDirectoryTemplate=CDROM
.Set CompressionType=MSZIP
.Set UniqueFiles=Off
.Set Cabinet=On
;*****************************************************
manifest.xml
..\Raona\bin\Debug\Raona.dll

.Set DestinationDir=CompartiMOSSTimerJob
feature.xml

;***End
```

2.2. WSPPackage.target: Genera el paquete a partir del WSPPackage.ddf

```
<?xml version="1.0" encoding="utf-8" ?>
<Project DefaultTargets="WSPPackage"
xmlns="http://schemas.microsoft.com/developer/msbuild/
2003">
  <PropertyGroup>

<MakeCabPath>"C:\cabsdk\bin\MAKECAB.EXE"</MakeC
abPath>
  </PropertyGroup>
```

```
<Target Name="WSPPackage">
  <Exec Command="$(MakeCabPath) /F WSPPackage.ddf
/D CabinetNameTemplate=$(MSBuildProjectName).wsp /D
DiskDirectory1=$(OutputPath)SpPackage\"/>
 </Target>
</Project>
```

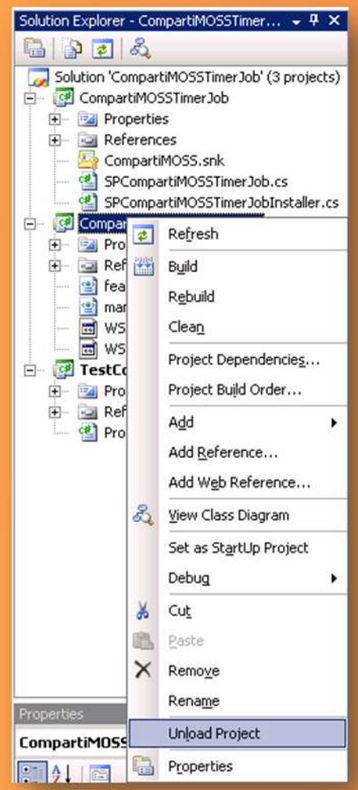
2.3. feature.xml: Describe la feature

```
<?xml version="1.0" encoding="utf-8" ?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
        Id="740C47EA-FBCC-42b0-BE14-9D24E618515C"
      Title="Ejemplo CompartiMOSS de trabajo temporizado"
      Description="Instala el job en la granja de servidores"
       Scope="Site"
       Hidden="FALSE"
       Version="1.0.0.0"
           ReceiverAssembly="Raona, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=5788b16638363c0b"
            ReceiverClass="Raona.TimerJobInstaller">
</Feature>
```

2.4. manifest.xml: Indica donde desplegar cada fichero de la solución

```
<?xml version="1.0" encoding="utf-8" ?>
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
                DeploymentServerType="WebFrontEnd"
       ResetWebServer="TRUE"
                SolutionId="527B0575-8117-429f-B654-
AC432E8C32A8">
  <Assemblies>
   <Assembly DeploymentTarget="GlobalAssemblyCache"

Location="Raona.dll"/>
  </Assemblies>
  <FeatureManifests>
                        <FeatureManifest
Location="CompartiMOSSTimerJob\feature.xml"/>
  </FeatureManifests>
</Solution>
```

3. El último paso es modificar el proyecto para que al compilarlo genere el WSP de forma automática. Para ello debemos modificar el fichero de proyecto con la opción "Unload Project":

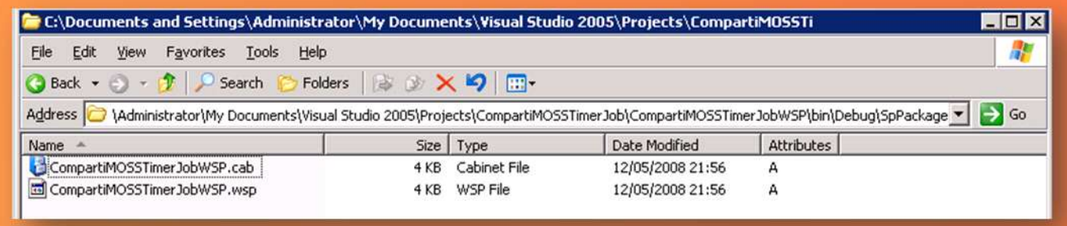


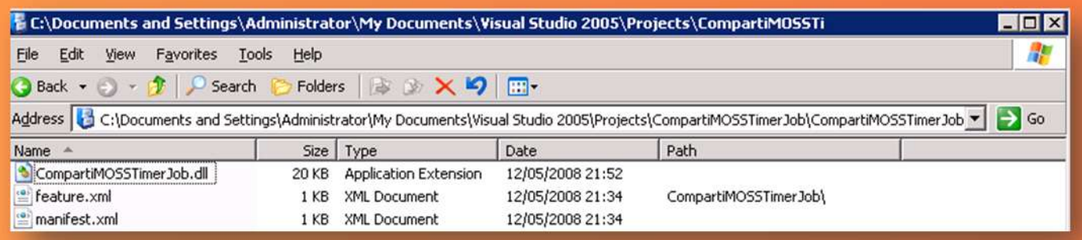
4. Al editar el proyecto debemos modificar los siguientes atributos sombreados para que al compilar generemos el fichero WSP.

```
<Project DefaultTargets="WSPPackage"
xmlns="http://schemas.microsoft.com/developer/msbuild
/2003" ToolsVersion="3.5">
...
<Import Project="WSPPackage.targets" />
...
</Project>
```

5. Con el fichero modificado y salvado cargamos de nuevo el proyecto de forma normal y ya estamos listos para compilar.
En primer lugar debemos compilar el proyecto con las clases y luego, si todo ha ido correctamente compilar el proyecto que genera el WSP.

6. Si todo ha ido correctamente, encontraremos una carpeta con el fichero CompartiMOSSTimerJob.WSP.
Si copiamos el fichero y le ponemos la extensión CAB el contenido es el siguiente:





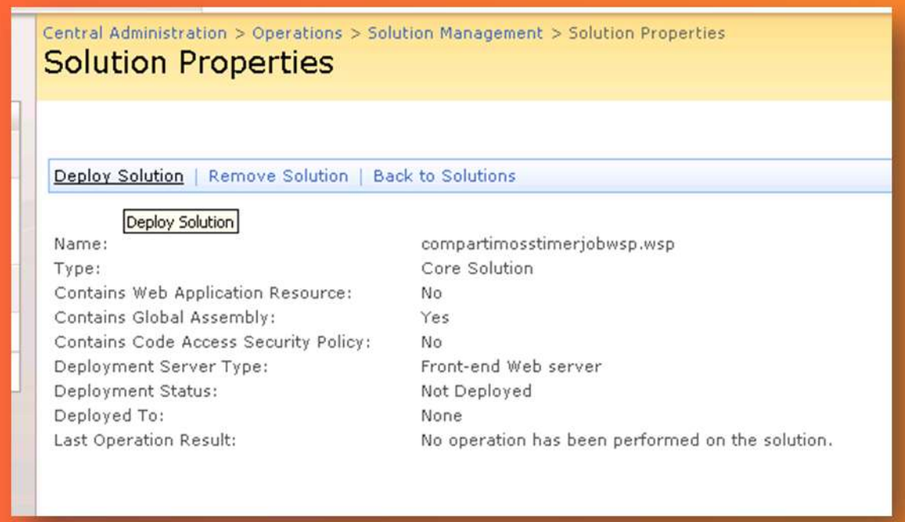
Despliegue de la solución

Llegados a este punto, acaba el trabajo de nuestro desarrollador y entramos en el escenario perfecto para cualquier administrador de SharePoint.
El administrador, ha recibido un fichero WSP y debe desplegarlo en su granja.
La ventaja del WSP es que su despliegue se hace a través del comando stsadm y por lo tanto representa un procedimiento estándar. Así lo único que debemos hacer para instalar nuestro trabajo temporizado es instalar la solución y activar la feature siguiendo estos pasos:

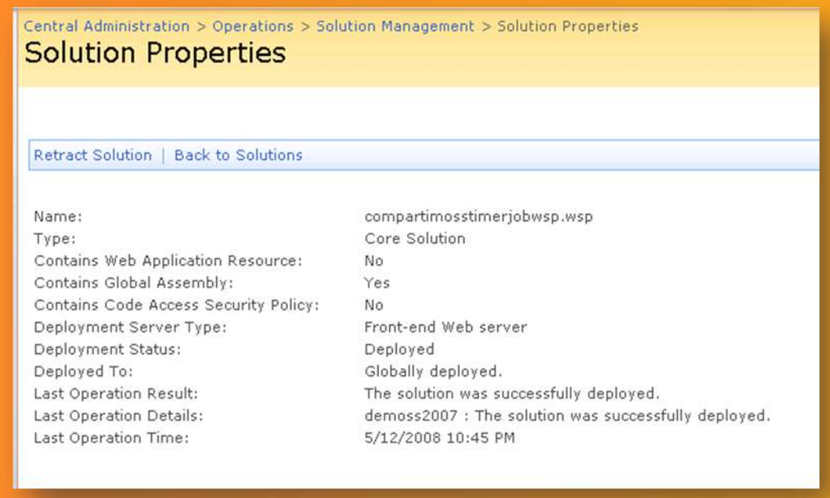
6.1. Instalar el WSP con el comando:
stsadm -o addsolution -filename CompartiMOSSTimerJobWSP.wsp

```
C:\Documents and Settings\Administrator>cd developments
The system cannot find the path specified.

C:\Documents and Settings\Administrator>cd \Developments

C:\Developments>stsadm -o addsolution

Missing required argument: filename.


stsadm.exe -o addsolution
           -filename <Solution filename>
           [-lcid <language>]

C:\Developments>stsadm -o addsolution -filename CompartiMOSSTimerJobWSP.wsp

Operation completed successfully.


C:\Developments>_
```

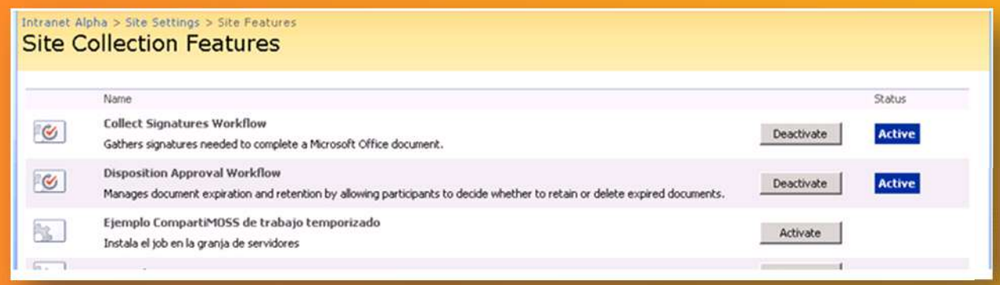
6.2. Una vez instalada podemos ir a la consola de administración central y desplegar la solución desde el administrador de soluciones:

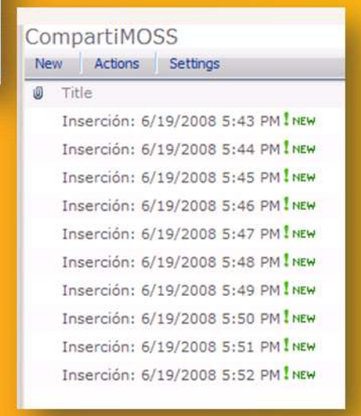


En caso de que algo fuera mal podemos tirar para atrás la solución desactivando la feature y luego desinstalando la solución con un retract y uninstall del comando stsadm.



6.3. Por último nos queda activar la feature y ver cómo se van almacenando los ítems en la lista que hemos creado:





Cambios

Si se modifica el proyecto deberemos tener en cuenta que los cambios en el temporizador no tendrán efecto hasta haber reiniciado el servicio de timer (owstimer.exe).

Conclusión

Hemos visto cómo generar y desplegar un paquete WSP en el que incluir un componente desarrollado por nosotros para cubrir cierta funcionalidad que SharePoint 2007 no ofrece. Dejando aparte el desarrollo de trabajos temporizados, me gustaría destacar el uso de ficheros WSP como artefactos del desarrollo en SharePoint.

Las soluciones SharePoint son el método perfecto para desplegar desarrollos entre entornos y evitar los procesos de despliegue que implican complejos procesos automatizados o copias manuales de ficheros (sobre todo si se ha usado SharePoint Designer).

Así, mi recomendación es usar WSP en proyectos de envergadura para automatizar el despliegue, reducir los errores y facilitar la tarea de los administradores.

Referencias

En el siguiente enlace encontraréis más información acerca de la creación de Timer Jobs en WSS 3.0: http://msdn.microsoft.com/es-es/library/cc406686(en-us).aspx

Àlex Peláez Membrado
Mail: Alex.Pelaez@raona.com
Blog: http://sharepointxperience.blogspot.com