|  |
| --- |
| Meetings & Events International |
| SharePoint Document System |
| Documentation for the MEI\_SP\_Documents project, SharePoint sites and document libraries, the file system, the SPDocumentsUploader project, and the databases that they use |

|  |
| --- |
| Jason Nichols  6/17/2011 |

Contents

[Adding a New Document to the System 4](#_Toc296087894)

[On the File System 4](#_Toc296087895)

[As a SharePoint Document Library 4](#_Toc296087896)

[In the MEI\_SP\_Documents Project Code 6](#_Toc296087897)

[Imports 6](#_Toc296087898)

[Namespaces 7](#_Toc296087899)

[Inherits 7](#_Toc296087900)

[Fields 8](#_Toc296087901)

[Attributes 8](#_Toc296087902)

[Constructors 9](#_Toc296087903)

[Properties 10](#_Toc296087904)

[Methods 12](#_Toc296087905)

[DocumentBroker 15](#_Toc296087906)

[In The Database 16](#_Toc296087907)

[DocumentType 16](#_Toc296087908)

[Permissions 16](#_Toc296087909)

[Adding a New Client SharePoint Site 18](#_Toc296087910)

[Creating the Site 18](#_Toc296087911)

[Site Permissions 19](#_Toc296087912)

[Document Libraries 20](#_Toc296087913)

[In the MEI\_SP\_Documents Project Code 20](#_Toc296087914)

[CustomConfig Entries 20](#_Toc296087915)

[Web Services 20](#_Toc296087916)

[Code Additions 21](#_Toc296087917)

[In the Database 22](#_Toc296087918)

[SharePoint Automatic Document Uploader 24](#_Toc296087919)

[Parameters 24](#_Toc296087920)

[Config Settings 24](#_Toc296087921)

[Permissions 25](#_Toc296087922)

[Event Database 26](#_Toc296087923)

[Schema 26](#_Toc296087924)

[DocumentType 26](#_Toc296087925)

[DocumentAccessControl 26](#_Toc296087926)

[DocumentPrivilege 26](#_Toc296087927)

[DocumentPrivilegeGroup 27](#_Toc296087928)

[DocumentPrivilegeGroupPrivilege 27](#_Toc296087929)

[Logging 28](#_Toc296087930)

[Schema 28](#_Toc296087931)

[DocumentLogs 28](#_Toc296087932)

[Web Services 29](#_Toc296087933)

[Intranet Site 33](#_Toc296087934)

[Client / Year Dropdowns 33](#_Toc296087935)

[Menu, Sitemap, & Breadcrumbs 33](#_Toc296087936)

[Business Data Columns 35](#_Toc296087937)

[Making SharePoint Aware of SQL1 35](#_Toc296087938)

[Creating the Document Library Column 36](#_Toc296087939)

[Adding a BDC Column to The MEI\_SP\_Documents Project 36](#_Toc296087940)

[The MEI\_SP\_Documents Project 38](#_Toc296087941)

[Known Issues 39](#_Toc296087942)

[Document File Name Standards and Column Names 40](#_Toc296087943)

# Adding a New Document to the System

For the purpose of this section, the Venue Contract document will be used as the example.

## On the File System

Folders will need to be created in the base document directory. That is, in the H:\Database\Documents directory, a folder named ‘VenueContract’ will need to be created. Inside of that folder, you will need to create additional folders which are prefixed by the parent folder’s name.

These folders include:

* ArchivedDocs
* ConvertedDocs
* FailedToConvertDocs
* FailedUploadDocs
* MergedDocs

When finished, the directory structure would look like this:

H:\Database\Documents\VenueContract\  
 VenueContractArchivedDocs\  
 VenueContractConvertedDocs\  
 VenueContractFailedToConvertDocs\  
 VenueContractFailedUploadDocs\  
 VenueContractsMergedDocs\

Most of these folders are used for the automated PDF conversion on PJ and the upload of documents to SharePoint. It is still a good idea to have these present even if neither of these options is used.

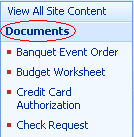
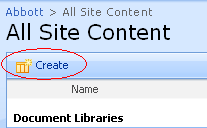
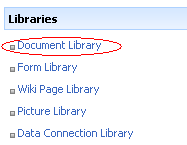
Do not put any spaces in any of the folder names. This naming convention should be used across all clients to ensure all automated processes work accordingly.

## As a SharePoint Document Library

The document library will need to include any information relevant to linking the document to other information such as fields like ProgramId, ExpenseCounter, SpeakerCounter, etc. Do not put fields in SharePoint document libraries which could otherwise be looked up in our regular database using one of these example ids, such as names or addresses, unless that info is needed for keeping a kind of history.

When a document library is setup on one client’s SharePoint site, it is usually a good idea to set it up across all of them. This is not necessary, but you may run across errors if you try to read or write to them in an automated process and they are not present.

Starting on the default page of a given client’s SharePoint site, click on the Documents link on the left hand menu of the page. Click the ‘Create’ link under the Heading ‘All Site Content’. Click the ‘Document Library’ link under the ‘Libraries’ subheading.

Your settings here will be:

Name Venue Contract  
 Display on Quick Launch Yes  
 Receive E-Mail No  
 Create a Version for Each Edit Yes  
 Document Template None

After you click the ‘Create’ button, you will be brought to the ‘All Documents’ view of your new document library. On the toolbar under the main title of the document library go to Settings 🡪 Document Library Settings.

You will first want to create any custom fields that you will need. Click the ‘Create column’ link under the ‘Columns’ subheading on the page. When it comes to naming the fields, try to be consistent across all documents and clients. This will ease the process of writing the object code later.

Unless you are using a Business Data column, the settings for your new column will most likely be:

Type of information Single line of text 🡨 Works for most fields, formats easier  
 Require that this column contain info Yes 🡨 Almost Always  
 Maximum number of characters 20 🡨 Works for most ID fields  
 Default value Text  
 Add to default view Yes

If you are using a Business Data column, read the [Business Data Column](#_Business_Data_Columns) section of this document for special instructions.

Click the ‘Indexed Columns’ link under the ‘Columns’ subheading. Check all of the fields that you just created.

Under the ‘Views’ subheading, click the ‘All Documents View’ link. You will want to reorder the fields so that they are easier to look at later. The order found best to work is:

Name  
 <The columns you created with the ID fields first>  
 Modified  
 Version  
 ID  
 Title

Once you click ‘OK’, you are done in SharePoint.

## In the MEI\_SP\_Documents Project Code

As of the writing of this document, the MEI\_SP\_Documents project is stored in:  
 TFS source control.

Team collection: MEI

Team project: ClassLibrary

Folder name: MEI\_SP\_Documents

Be sure that you are set the build configuration to ‘Debug’ when testing.

We will start by getting the guids for the new document library. For this you will need to Remote Desktop to the   
vs-sharepoint1 server. On the C:\ there is a shortcut to an application called ‘SPInfo.exe’. Run it. Enter the number index for the ‘Get document library guids’ operation and press enter. It was 2 at the time of this writing. Enter the number index for the SharePoint site that you are interested in and press enter. Enter the number index for the document library that you are interested in and press enter. The first guid will be displayed and the entire configuration file line for it will be added to your clipboard. Open the CustomConfig.xml file and navigate to the appropriate client and section of that client and paste what is on your clipboard. Place focus back on the SPInfo.exe application. Press enter to display the next guid and to have its configuration file line copied to your clipboard. Open the CustomConfig.xml file and navigate to the appropriate client and section and paste what is on your clipboard.

Now we will start writing the code file itself. Under the Documents folder, create a new class file. The name of the class should match the name of the document. So, in this case, it would be VenueContract.vb.

Follow the examples of the other Documents here to make the setup of this new one easiest. Set this new document class up with the following elements:

### Imports

|  |  |
| --- | --- |
| **Name** | System.Runtime.InteropServices |
| **Code** | Imports System.Runtime.InteropServices |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | System.Xml |
| **Code** | Imports System.Xml |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | MEI.SPDocuments.Type |
| **Code** | Imports MEI.SPDocuments.Type |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | MEI.SPDocuments.Document.FieldName |
| **Code** | Imports MEI.SPDocuments.Document.FieldName |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | System.Text |
| **Code** | Imports System.Text |
| **Summary** |  |
| **Comments** |  |

### Namespaces

|  |  |
| --- | --- |
| **Name** | SPDocuments.Document |
| **Code** | Namespace SPDocuments.Document  End Namespace |
| **Summary** |  |
| **Comments** |  |

### Inherits

|  |  |
| --- | --- |
| **Name** | SPDocumentBase |
| **Code** | Inherits SPDocumentBase |
| **Summary** |  |
| **Comments** | The class must inherit the SPDocumentBase abstract class and properly implement all of its abstract members. The class can also override any of the virtual members as needed. |

### Fields

There will be a field for every field that you created inside of the document library as well as the ClassId field.

|  |  |
| --- | --- |
| **Name** | ClassId |
| **Code** | Public Const ClassId As String = "4B7E0640-35FD-46e5-859B-75D80B4461AD" |
| **Summary** |  |
| **Comments** | To get the value for this field, use the Visual Studio => Tools => Create GUID to generate a guid. |

|  |  |
| --- | --- |
| **Name** | \_programId |
| **Code** | Private \_programId As String |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | \_vendorId |
| **Code** | Private \_vendorId As Integer? |
| **Summary** |  |
| **Comments** |  |

### Attributes

You will also need the following attributes above your class declaration. These are used for COM purposes when using these documents in Access.

|  |  |
| --- | --- |
| **Name** | GuidAttribute |
| **Code** | <GuidAttribute(VenueContract.ClassId)> |
| **Summary** |  |
| **Comments** | Be sure to change the class in the GuidAttribute to match the new class name! |

|  |  |
| --- | --- |
| **Name** | ClassInterface |
| **Code** | <ClassInterface(ClassInterfaceType.None), ComSourceInterfaces(GetType(IDocument))> |
| **Summary** |  |
| **Comments** |  |

### Constructors

|  |  |
| --- | --- |
| **Name** | New |
| **Code** | Public Sub New()  MyBase.New()  End Sub |
| **Summary** |  |
| **Comments** | Used strictly for COM purposes. |

|  |  |
| --- | --- |
| **Name** | New |
| **Code** | Friend Sub New(ByVal programId As String, ByVal vendorId As Integer?, ByVal contents As Byte(), ByVal fileExtension As String, ByVal company As CompanyCode, ByRef spDocs As SP\_Documents)  Me.New(contents, fileExtension, company, spDocs)  \_programId = programId  \_vendorId = vendorId  \_documentYear = ExtractDocumentYear(\_programId)  End Sub |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | New |
| **Code** | Friend Sub New(ByVal programId As String, ByVal vendorId As Integer?, ByVal filePath As String, ByVal fileExtension As String, ByVal company As CompanyCode, ByRef spDocs As SP\_Documents)  Me.New(company, fileExtension, spDocs)  \_programId = programId  \_vendorId = vendorId  \_documentYear = ExtractDocumentYear(\_programId)  \_filePath = filePath  End Sub |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | New |
| **Code** | Friend Sub New(ByVal fileName As String, ByVal contents As Byte(), ByVal company As CompanyCode, ByRef spDocs As SP\_Documents)  MyBase.New(fileName, contents, company, spDocs)  End Sub |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | New |
| **Code** | Friend Sub New(ByVal filePath As String, ByVal company As CompanyCode, ByRef spDocs As SP\_Documents)  MyBase.New(filePath, company, spDocs)  End Sub |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | New |
| **Code** | Private Sub New(ByVal contents As Byte(), ByVal fileExtension As String, ByVal company As CompanyCode, ByRef spDocs As SP\_Documents)  MyBase.New(contents, fileExtension, company, spDocs)  End Sub |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | New |
| **Code** | Private Sub New(ByVal company As CompanyCode, ByVal fileExtension As String, ByRef spDocs As SP\_Documents)  MyBase.New(company, fileExtension, spDocs)  End Sub |
| **Summary** |  |
| **Comments** |  |

### Properties

|  |  |
| --- | --- |
| **Name** | FileName |
| **Code** | Public Overrides ReadOnly Property FileName() As String  Get  Return MakeFileName(ProgramId, VendorId.Value.ToString())  End Get  End Property |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | IsValid |
| **Code** | Public Overrides ReadOnly Property IsValid() As Boolean  Get  Dim baseValid As Boolean = MyBase.IsValid  If String.IsNullOrEmpty(ProgramId) Then  Return False  ElseIf Not VendorId.HasValue Then  Return False  End If  Return baseValid  End Get  End Property |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | PrefixText |
| **Code** | Public Overrides ReadOnly Property PrefixText() As String  Get  Return Name  End Get  End Property |
| **Summary** | Used as the prefix in the filename generation. |
| **Comments** | This property should only be overridden if the prefix for the filename is something other than the base class’ implementation of it which is, Acronym. In the case of the VenueContract, it uses Name instead of Acronym. |

|  |  |
| --- | --- |
| **Name** | AllowedFileTypes |
| **Code** | Protected Overrides ReadOnly Property AllowedFileTypes() As Collection(Of String)  Get  Return New Collection(Of String) From {"pdf", "doc"}  End Get  End Property |
| **Summary** |  |
| **Comments** | This property should only be overridden if the allowed file types for the document differ from those of the base class’ which are ‘pdf’.  Note: VenueContract does not actually override this property. It is just being shown as an example in case other documents might. |

|  |  |
| --- | --- |
| **Name** | ProgramId |
| **Code** | Public ReadOnly Property ProgramId() As String  Get  Return \_programId  End Get  End Property |
| **Summary** | Backing field: [\_programId](#FieldmProgramId) |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | VendorId |
| **Code** | Public ReadOnly Property VendorId() As Integer?  Get  Return \_vendorId  End Get  End Property |
| **Summary** | Backing field: [\_vendorId](#FieldmVendorId) |
| **Comments** |  |

### Methods

|  |  |
| --- | --- |
| **Name** | ValidateFields |
| **Return Type** | Boolean |
| **Code** | Protected Friend Overrides Function ValidateFields() As Boolean  If Not IsValid Then  Return False  End If  If Not FieldValidator.ValidateProgramId(Company, \_documentYear, ProgramId) Then  ThrowFileNameExceptionNoDbMatch(VenueContractFieldName.ProgramId, ProgramId)  End If  If Not FieldValidator.ValidateVendorId(Company, \_documentYear, VendorId.Value) Then  ThrowFileNameExceptionNoDbMatch(VenueContractFieldName.VendorId, VendorId.Value.ToString())  End If  Return True  End Function |
| **Summary** | Ensures that the document is valid first. Then, it ensures that all of id type fields exist in their corresponding databases. |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | Setup |
| **Parameters** | objects() As Object |
| **Return Type** | Boolean |
| **Code** | Public Overrides Function Setup(ByRef objects() As Object) As Boolean  Dim userFieldCount As Integer = GetUserFieldCount()  'Add three to userFieldCount for the contents, fileExtension, and company  userFieldCount += 3  If objects.Length <> userFieldCount Then  Return False  End If  \_programId = objects(0).ToString()  \_vendorId = Convert.ToInt32(objects(1))  Contents = CType(objects(2), Byte())  \_fileExtension = Convert.ToString(objects(3)).Replace(".", String.Empty)  \_company = CType(objects(4), CompanyCode)  \_documentYear = ExtractDocumentYear(\_programId)  Return IsValid  End Function |
| **Summary** | Assigns all of the fields in the document with their corresponding values in the specified object array. This method is used as a constructor overload replacement for com instantiations. |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | GetUploadXml |
| **Return Type** | XmlNode |
| **Code** | Protected Friend Overrides Function GetUploadXml() As XmlNode  Dim node As XmlNode = MyBase.GetUploadXml()  Dim uploadXml As New StringBuilder()  uploadXml.Append(MakeQueryField(VenueContractFieldName.ProgramId, ProgramId))  uploadXml.Append(MakeQueryField(VenueContractFieldName.VendorId, VendorId.Value.ToString()))  node.InnerXml &= uploadXml.ToString()  Return node  End Function |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | Initialize |
| **Code** | Protected Overrides Sub Initialize()  \_documentType = SPDocumentTypeCode.VenueContract  \_spFields.Add(VenueContractFieldName.ProgramId, "Program ID", "Program\_x0020\_ID", SPFieldTypeCode.Text, True)  \_spFields.Add(VenueContractFieldName.VendorId, "VendorID", SPFieldTypeCode.Text, True)  \_spFields.Add(VenueContractFieldName.Id, "ID", SPFieldTypeCode.Counter)  \_spFields.Add(VenueContractFieldName.Created, "Created", SPFieldTypeCode.DateTime)  \_spFields.Add(VenueContractFieldName.Title, "Title", SPFieldTypeCode.Text)  \_spFields.Add(VenueContractFieldName.Modified, "Modified", SPFieldTypeCode.DateTime)  \_spFields.Add(VenueContractFieldName.FileName, "FileName", "FileLeafRef", SPFieldTypeCode.Text)  End Sub |
| **Summary** | Initializes the SPField collection with the appropriate fields of the SharePoint document library. |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | ParseFileName |
| **Parameters** | fileNameToParse As String |
| **Return Type** | String() |
| **Code** | Protected Overrides Function ParseFileName(ByVal fileNameToParse As String) As String()  Dim fileNameParts As String() = MyBase.ParseFileName(fileNameToParse)  \_programId = fileNameParts(1)  Dim tempVendorId As Integer  If Not Integer.TryParse(fileNameParts(2), tempVendorId) Then  ThrowFileNameExceptionInvalidType(fileNameToParse, VenueContractFieldName.VendorId, "Integer")  End If  \_vendorId = tempVendorId  \_documentYear = ExtractDocumentYear(\_programId)  Return fileNameParts  End Function |
| **Summary** | Parses the specified fileNameToParse to ensure that it contains the required fields needed for the document and that the fields that it does contain are of the correct type. If it meets these requirements then these fields in the fileNameToParse are assigned to the fields of the document. |
| **Comments** |  |

### DocumentBroker

In the DocumentBroker class, ‘Create’ functions need to be created.

The ‘Create’ functions should implement the three main constructors of the document as public functions. These should be overloaded and named Create<ClassName>.

|  |  |
| --- | --- |
| **Name** | CreateVenueContract |
| **Parameters** | programId As String  vendorId As Integer?  Contents As Byte()  fileExtension As String  company As CompanyCode |
| **Return Type** | VenueContract |
| **Code** | Public Function CreateVenueContract(ByVal programId As String, ByVal vendorId As Integer?, ByVal contents As Byte(), ByVal fileExtension As String, ByVal company As CompanyCode) As VenueContract  Return New VenueContract(programId, vendorId, contents, fileExtension, comp, \_spDocs)  End Function |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | CreateVenueContract |
| **Parameters** | fileName As String  contents As String  company As CompanyCode |
| **Return Type** | VenueContract |
| **Code** | Public Function CreateVenueContract(ByVal fileName As String, ByVal contents As Byte(), ByVal company As CompanyCode) As VenueContract  Return New VenueContract(fileName, contents, comp, \_spDocs)  End Function |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | CreateVenueContract |
| **Parameters** | filePath As String  company As CompanyCode |
| **Return Type** | VenueContract |
| **Code** | Public Function CreateVenueContract(ByVal filePath As String, ByVal company As CompanyCode) As VenueContract  Return New VenueContract(filePath, comp, \_spDocs)  End Function |
| **Summary** |  |
| **Comments** |  |

## In The Database

### DocumentType

All of the Document tables are located in the Events database. There are two main tables you will need to deal with when adding a new Document: DocumentAccessControl, DocumentType. We will start with DocumentType.

The DocumentName & DocumentAcronym columns are self-explanatory. The only thing to note here is the DocumentAcronym column has a UNIQUE constraint on it; name accordingly.

The FolderName column’s value is the name of the folder that you gave the document on the file system. In our case this would be VenueContract.

The ClassName column’s value is the name of the class that you gave the object in the MEI\_SP\_Documents project. In our case, it is VenueContract. During automated upload and possibly searching, methods are called by name and the document classes are instantiated by class name. This reduces the need to update so much code every time you want to add a document.

The five bit fields beginning with ‘Search’ are for searching purposes. Their names are self-explanatory really. If a document can be searched by ProgramId, i.e. it has a ProgramId column setup in SharePoint, then its value would be set to true. These fields enable the grouping you see on the Document Search page in the intranet.

The ‘IsGetVersions’ bit field specifies whether or not a search on this document type should also return the versions of that particular document.

The ‘IsDisabled’ bit field is self-explanatory. A document with this field set to true will not be able to be searched on.

The ‘IsInFaxNames’ bit field is used on the ‘MEIIntranet FaxNames’ web page. If this field is set to true for a document, then that document will be populated in the dropdown on the FaxNames page.

### Permissions

Permissions for each document are setup in the database as well. These are setup in the [DocumentAccessControl](#_DocumentAccessControl) table. There is also a hierarchy system that can be setup for permissions.

The DocumentType column links to the ID field of the [DocumentType](#_DocumentType) table and the Privilege column links to the ID field of the [DocumentPrivilege](#_DocumentPrivilege) table. It is the Username field that we need to worry about. Many types of usernames can go in to this field. For intranet use, you can put the name of an Active Directory group such as ‘Domain Admins’ or ‘IS’. You can also put individual user names, like ‘MEIDOMAIN1\jnichols’. Say you are using this on a client website and you want to give permission to certain documents once you have decided that the user logged in is a speaker. You put a username of ‘ClientSpeaker’ in the database and use that as your username.

When using the SP\_Documents class to access documents, a username is required as a parameter in the constructor for the class. This is where you would enter ‘ClientSpeaker’ or ‘MEIDOMAIN1\jnichols’. If you enter a ‘MEIDOMAIN1’ user (such as you would on the intranet), all of that users’ Domain Groups will be loaded from the Domain Controller as well as the username into a list. That list will automatically be used when checking for permissions to documents.

If you need to add a new permission type, you must be sure to make entries in all of the tables. These tables are [DocumentPrivilege](#_DocumentPrivilege), [DocumentPrivilegeGroup](#_DocumentPrivilegeGroup), and [DocumentPrivilegeGroupPrivilege](#_DocumentPrivilegeGroupPrivilege). Once you create one in [DocumentPrivilege](#_DocumentPrivilege), you must either associate it with an already existing group in [DocumentPrivilegeGroupPrivilege](#_DocumentPrivilegeGroupPrivilege), or create a new group in [DocumentPrivilegeGroup](#_DocumentPrivilegeGroup), and then make that connection. It is in [DocumentPrivilegeGroupPrivilege](#_DocumentPrivilegeGroupPrivilege) that you can setup a hierarchy for permissions (i.e. ViewOriginal trumps ViewWatermarked). 1 is the highest. This ensures you always have the highest permissions possible.

# Adding a New Client SharePoint Site

## Creating the Site

To begin this process, you will want to log in to the SharePoint Central Administration webpage. This can be done by going to <http://vs-sharepoint1:11684>

Now, go to the Application Management tab. Under the SharePoint Web Application Management sub-heading, go to ‘Create or extend Web application.’ 🡪 Create a new Web application.

Please use meaningful descriptions on fields and folder names. Looking for a folder by a client name rather than a port number is much easier.

For this example we will use the company AstraZeneca.

Create a new IIS Web site Yes  
 Description SharePoint – AstraZeneca  
 Port Whichever one is provided is usually fine  
 Path Change the port number folder to the name  
 i.e. …\VirtualDirectories\27357 to  
 ….\VirtualDirectories\SPAstraZeneca  
 Note: The full path is still required.  
 Authentication Provider NTLM  
 Allow Anonymous No  
 Use Secure Sockets Layer (SSL) No  
 URL Leave as-is  
 Create new application pool Yes  
 Application pool name SharePoint – AstraZeneca  
 Configurable Yes  
 Username MEIDOMAIN1\sharepointservice  
 Password good.4you2  
 Restart IIS Manually Yes  
 Database Server SQL1  
 Database Name WSS\_Content\_AstraZeneca  
 Windows Authentication Yes  
 Search Server VS-SHAREPOINT1

Once you click OK and you get the confirmation that the site has been created, email IT and inform them that there is a new SharePoint database that needs to begin being backed up; WSS\_Content\_AstraZeneca.

Back on the Application Management tab, under the SharePoint Site Management sub-heading, go to ‘Create site collection’

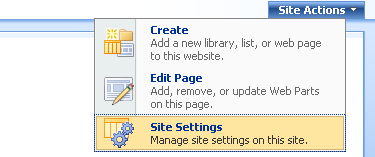
Be sure your new site is selected in the drop down list and your new settings will most likely be as follows:

Title AstraZeneca  
 Description Not necessary but you may put one if desired  
 Web Site Address Simply use the root /  
 Template Selection Team Site  
 Primary Site Collection Administrator MEIDOMAIN1\jnichols   
 Secondary Site Collection Administrator Someone else  
 Quota Template No Quota

You should now be finished. Go ahead and go to your new AstraZeneca page.

## Site Permissions

Access to all document libraries across every client site should be given to the MEIDOMAIN\MEIDocs user.

On your new default AstraZeneca site, in the top-right hand corner of the page go to Site Actions 🡪 Site Settings  
 

On the Site Settings page, under the Users and Permission sub-heading, go to ‘People and Groups.’

Then on the toolbar, go to New 🡪 Add Users

Your settings should be:

Users/Group MEIDOMAIN1\MEIDocs  
 Add Users to A SharePoint group <SiteName> Members [Contribute]  
 Send E-Mail No

Permissions are finished.

## Document Libraries

This part will take a while and hopefully will be changed soon. You will have to go though and manually create each document library. The easiest way to do this is to open another client site and just look at the settings one by one. In a later date, I hope to have a site template setup so that the creation of the libraries and the permission setup is not necessary. Until then, have fun.

## In the MEI\_SP\_Documents Project Code

### CustomConfig Entries

Follow the guidelines under the [Adding a New Document in the MEI\_SP\_Documents Project Code](#_In_the_MEI_SP_Documents_1) about adding the GUID’s for your new document libraries in the CustomConfig.xml. Yours would look like:

<add key="AstraZeneca-BEO-DocLibGuid" value="{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}" />   
<add key="Search-AstraZeneca-BEO-DocLibGuid" value="XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX"/>

Near the top of the CustomConfig.xml, you will find a group of keys with the suffix ‘BaseSPDocumentLibSite.’ You need to add a new ‘AstraZeneca-BaseSPDocumentLibSite’ key here. Be sure you have the ‘/’ at the end of the path. It is important that you keep the prefix ‘AstraZeneca’ on your GUID key names, the prefix ‘AstraZeneca’ on this key, and all subsequent spellings and capitalizations the same. Example:

<add key="AstraZeneca-BaseSPDocumentLibSite" value="http://vs-sharepoint1:XXXX/" />

At the bottom of the CustomConfig.xml you will find the database connection strings. You will need two entries here, one called ‘CurrentAstraZeneca’ and one called Intranet-CurrentAstraZeneca. Be sure you test these and they work in both situations. Example:

<add key="CurrentAstraZeneca" value="server=SQL1;database=astrazeneca2010;integrated security=true;connect timeout=90;" />

<add key="Intranet-AstraZeneca" value="server=SQL1;database=astrazeneca2010;Trusted\_Connection=Yes;" />

In the MEI\_SP\_Documents ‘Types’ folder, open the Company.vb file. Here you will need to add the AstraZeneca enum, as well as add the functionality to the two-shared functions on the page. Be sure to keep your spelling the same.

### Web Services

Searching and uploading is taken care of by three separate web services. The Lists web service is one provided by SharePoint Services, the second is a custom upload web service but you are not required to add this one for each client. The MEIUniversal one is used for all clients. Finally, the Version web service is used to get all of the versions of a particular file.

Right click on the Web References folder in the project and go to ‘Add Web Reference’. We will assume the port number for the new site is 22222.

The URL you’ll type in first is <http://vs-sharepoint1:22222/_vti_bin/Lists.asmx>. Once you click ‘Go’ you should see a list of all the functions discovered. Rename the web reference to SPAstraZeneca\_Lists and click ‘Add Reference’.

The URL you’ll type in second is <http://vs-sharepoint1:22222/_vti_bin/Versions.asmx>. Once you click ‘Go’ you should see a list of all the function discovered. Rename the web reference to SPAstraZeneca\_Versions and click ‘Add Reference’.

### Code Additions

In the SP\_Documents.SubmitBarcodeEntry function, you’ll need to add AstraZeneca to the Select Case of database names to associate with the barcode entry.

Your next set of changes will occur in the DocumentBroker class. You’ll need to ‘Dim’ a new instance of the Lists service and an instance of the Versions service for AstraZeneca.

|  |  |
| --- | --- |
| **Name** | \_AstraZenecaList |
| **Code** | Private \_AstraZenecaList As MEI.SPAstraZeneca\_Lists.Lists |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | \_AstraZenecaVersion |
| **Code** | Private \_AstraZenecaVersion As MEI.SPAstraZeneca\_Versions.Versions |
| **Summary** |  |
| **Comments** |  |

You will then want to create a property for both of these fields. This property, will allow the services to be lazily loaded, i.e. not loaded until they are needed.

|  |  |
| --- | --- |
| **Name** | AstraZenecaList |
| **Code** | Private ReadOnly Property AbbottList As SPAbbott\_Lists.Lists  Get  If \_AbbottList Is Nothing Then  \_AbbottList = New SPAbbott\_Lists.Lists()  \_AbbottList.Credentials = \_spDocs.SPCredential  End If  Return \_AbbottList  End Get  End Property |
| **Summary** |  |
| **Comments** |  |

|  |  |
| --- | --- |
| **Name** | AstraZenecaVersion |
| **Code** | Private ReadOnly Property AbbottVersion As SPAbbott\_Versions.Versions  Get  If \_AbbottVersion Is Nothing Then  \_AbbottVersion = New SPAbbott\_Versions.Versions()  \_AbbottVersion.Credentials = \_spDocs.SPCredential  End If  Return \_AbbottVersion  End Get  End Property |
| **Summary** |  |
| **Comments** |  |

Unless you misspelled column names when creating the document libraries, you should be finished with code additions. If you did misspell columns, you will need to go in to those document objects and add special cases for the XML in the [Initialize](#MethodInitialize) function of the document.

## In the Database

There are multiple stored procedures that must be added to a clients SQL1 database in order for it to function on the intranet site. This list is as follows and should be updated whenever one is added

Note: the WEBSVR1$ user needs to have Execute permissions on all of these stored procedures

* usp\_GetSpeakersByName
* usp\_GetSpeakerNameBySpkrCounter
* usp\_GetProgramIDByRegion
* usp\_GetProgramIDByDistrict
* usp\_GetProgramExpenses
* usp\_GetSpeakerListBySpkrCntr
* usp\_GetSpeakerNameByLName
* usp\_GetLikeTerritories
* usp\_GetLikeProgramID
* usp\_GetLikeRegions
* usp\_GetLikeDistricts
* usp\_GetProgramsByTerritory
* usp\_GetPIFInfoForProgram
* usp\_GetProgramInfoByProgram
* usp\_GetProgramsBySpeakerCounter
* usp\_GetProgramsByVendorID
* usp\_GetProgramsByDateRange
* usp\_GetProgramsByPayTo
* usp\_GetProgramsByCityState
* usp\_GetLikeExpenses
* usp\_GetVendorsByVendorID

# SharePoint Automatic Document Uploader

The SharePoint document uploader goes through each of the documents ‘ConvertedDocs’ folders and tries to upload the files in them to the correct site and document library. If you have followed all the naming conventions listed above and have all your database entries setup correctly, no code changes will be needed when adding a client or document. The only changes that will need to be made will be the parameters you pass the program.

## Parameters

Parameters are used to specify which companies to scan and which documents to scan for those companies. Examples of these parameters are listed below. The company name is what you setup in the MEI\_SP\_Documents CompanyHandler class in the Shared Functions. Be sure they are the same.

SharePointConsoleUploader.exe –Abbott:W9,TI,BEO –Solvay:ALL –AstraZeneca:PIF,ER,PVM

It basically goes company name, colon, comma delimited document acronyms with no spaces, then repeat for the next company.

## Configuration Settings

In the app.Config file, there are only two keys you need to worry about, the BasePath, and the LogPath. Each time the uploader is run, text logs are written to this path. The BasePath is where all the Documents folders are located.

# Permissions

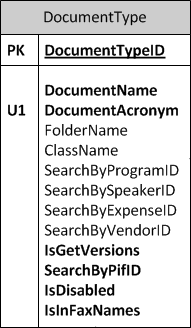
If a user wants access to view a document, they must specifically be given permission to do so. This can be done as described above in the [Permissions](#_Permissions) section above.

There are three main permissions we currently use: Search, ViewOriginal, and ViewWaterMarked. A user will not be able to even see the document folder in the search at all unless they have Search permissions. If a user has the ViewWaterMarked permissions, that document will be presented to them with a VOID watermark stamped across. The ViewOriginal permission allows documents to be viewed as is and trumps the ViewWaterMarked permission should the user be assigned both.

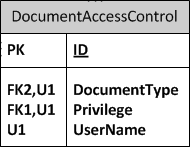
# Event Database

## Schema

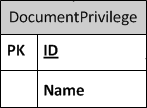
### DocumentType



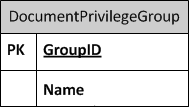
### DocumentAccessControl



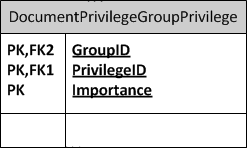
### DocumentPrivilege



### DocumentPrivilegeGroup



### DocumentPrivilegeGroupPrivilege



# Logging

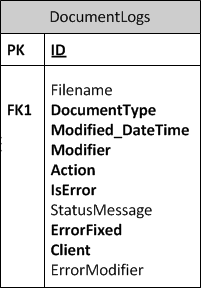
The reason there are so many specifications on access modifiers and permissions is because we want to keep tight logging on all document activity.

No one besides the MEIDocs user (and the IS folks of course) should have direct access to the document libraries of any client besides the MEIUniversal site. Any access of documents should be done through the intranet. The MEI\_SP\_Documents library should be used to write any code to access these documents. This is because permissions and logging are built directly into this project.

Any time a document is uploaded or attempted to be uploaded and failed, a record is made of the operation. These records are stored in the Events database in the [DocumentLogs](#_DocumentLogs) table.

## Schema

### DocumentLogs



# Web Services

SharePoint comes with many web services built in. Unfortunately, there are some bugs in those web services, namely the document upload web service. This is why there is a custom SharePoint web service running on vs-sharepoint1. The project for this can be found in H:\Database\Development\VB\DocLibUpload\_Service.

Should you need to make a change to this web service and redeploy it, there are some special tasks you must complete in order for it to be seen by SharePoint.

This Channel 9 video should explain most of the points of deployment: <http://channel9.msdn.com/posts/kirke/SharePoint-for-Developers-Part-6-Custom-web-services/>

MSDN Walkthrough: Creating a Custom ASP.NET Web Service

<http://msdn.microsoft.com/en-us/library/ms464040.aspx>

The basic rundown of the edit process is

1. Modify the code in the project as needed.
2. Ensure that the markup in the .asmx file follows this pattern:

***<%@ WebService Language="<Language>" Class="<FullyQualifiedClassName>, <NamespaceName>, Version=<Version>, Culture=<Culture>, PublicKeyToken=<PublicKeyToken>" %>***

-So, for this example, it would look like this:

***<%@ WebService Language="VB" Class="SPDocLibUpload\_WebService.SPDocLibUpload, SPDocLibUpload\_WebService, Version=1.0.0.0, Culture=neutral, PublicKeyToken=b89a229a437dc98" %>***

To obtain a Public Key Token:

1. Open the 'Visual Studio Command Prompt'
2. Type

***sn.exe -Tp <PathToDLL>***

, and press 'Enter'.

So, for this example, it would look like this:

***sn.exe –Tp "C:\Documents and Settings\jnichols\My Documents\Visual Studio 2008\Projects\DocLibUpload\_Service\SPDocLibUpload\_WebService\SPDocUpload\_WebService\bin\SPDocLibUpload\_WebService.dll"***

1. Ensure that the project is signed with a strong name.

To sign the project with a strong name:

1. Right-click the project in the Solution Explorer and click 'Properties'
2. Select the tab titled 'Signing' on the left
3. Check the box in the bottom titled 'Sign the assembly'
4. In the dropdown titled, 'Choose a strong name key file:' select the option titled '<New...>'
5. Type the name of the file that this key will be saved as in the box titled 'Key file name'
6. Uncheck the box titled 'Protect my key file with a password' and click 'OK'

else check the box titled 'Protect my key file a password' and enter a password in the box titled 'Enter password' and the confirmation password in the box titled 'Confirm password' and click 'OK'

If you choose the password protected option, then the password must be entered every time the project is opened

1. Save the project properties
2. Build the project
3. Add the built assembly to the Global Assembly Cache (GAC) in one of two ways:
   1. First way
      1. Directly copy the assembly file into the GAC. That is, copy the assembly file into the directory '***C:\WINDOWS\assembly***'
   2. Second way
      1. Open the 'Visual Studio Command Prompt'
      2. Type

***gacutil.exe /if <PathToAssembly>***

, and press 'Enter'

So, for this example, it would look like

***gacutil.exe /if "C:\Documents and Settings\jnichols\My Documents\Visual Studio 2008\Projects\DocLibUpload\_Service\SPDocLibUpload\_WebService\SPDocLibUpload\_WebService\bin\SPDocLibUpload\_WebService.dll"***

1. Copy the .asmx from the project directory into the '\_vti\_bin' directory of the SharePoint site that the web service will be located at.

So, for this example, the '\_vti\_bin' virtual directory for the 'MEI Universal' SharePoint site points to the physical directory,

***C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\ISAPI***

1. Generate the disco and wsdl files associated with the .asmx file
   1. Open a regular command prompt
   2. Navigate to the directory containing the 'SPDev.exe' application.

The website for this tool is: <http://blog.crsw.com/spdev/>.

It can be downloaded from CodePlex here: <http://spdev.codeplex.com/>.

* 1. Type

***SPDev.exe -o GenWS -url <BaseUrlOfSharePointSite> -asmx <PathOfCopiedFileInAboveStep>***

, and press 'Enter'

So, for this example, it would look like,

***SPDev -o GenWS -url http://vs-sharepoint1:17074 -asmx "C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\ISAPI\SPDocLibUpload.asmx”***

The newly generated disco and wsdl files will be in the same directory as the 'SPDev.exe'

1. Copy the disco and wsdl files into the ‘\_vti\_bin’ directory of the SharePoint site that the web service will be located at.

So, for this example, the ‘\_vti\_bin’ virtual directory for the ‘MEI Universal’ SharePoint site points to the physical directory,

***C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\ISAPI***

Note:

Extra tools that may help with this process:

Visual Studio External Tools

1. Public Key Token
   * Title: Get SN Token
   * Command: C:\Program Files\Microsoft SDKs\Windows\v6.0A\bin\sn.exe

If unsure of the location of 'sn.exe', open a 'Visual Studio Command Prompt' and type 'where sn.exe' and press enter.

* Arguments: -Tp $(TargetPath)
* Use Output window: Checked

1. Visual Studio Command Prompt
   * Title: Cmd
   * Command: %comspec%
   * Arguments: /k "C:\Program Files\Microsoft Visual Studio 9.0\VC\vcvarsall.bat" x86
   * Initial directory: $(ItemDir)

**-This is the old way of creating the disco and wsdl files before the finding of the SPDev.exe app**  
*Generate the .disco and .wsdl files using the Visual Studio command prompt   
 cd “C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\TEMPLATE\LAYOUTS  
 disco http://vs-sharepoint1:53090/SPDocLibUpload.asmx  
Rename the .disco and .wsdl files to .aspx  
 So say you had Test.asmx, Test.disco, & Test.wsdl, they would become Test.asmx, Testdisco.aspx,   
 Testwsdl.aspx  
Replace any reference to an address in the disco and wsdl files to a dynamic function.  
 <% SPHttpUtility.AddQuote(SPHttpUtility.HtmlEncode(SPWeb.OriginalBaseUrl(Request)),Response.Output); %>  
 The video will be able to explain exactly where  
There are also some page directives that go at the top of the pages, they can be found below.*

*Copy the three files into C:\Program Files\Common Files\Microsoft Shared\web server extensions\12\ISAPI\*

*Here is a copy of what the discovery node and page directives in the disco file should look like:*

*<%@ Page Language="C#" Inherits="System.Web.UI.Page" %>   
<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>   
<%@ Import Namespace="Microsoft.SharePoint.Utilities" %>   
%@ Import Namespace="Microsoft.SharePoint" %>  
<% Response.ContentType = "text/xml"; %>  
  
 <discovery xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap.org/disco/">*

*<contractRef ref=<% SPHttpUtility.AddQuote(SPHttpUtility.HtmlEncode(SPWeb.OriginalBaseUrl(Request) + "?wsdl"),Response.Output); %> docRef=<% SPHttpUtility.AddQuote(SPHttpUtility.HtmlEncode(SPWeb.OriginalBaseUrl(Request)),Response.Output); %> xmlns=" http://schemas.xmlsoap.org/disco/scl/ " />*

*<soap address=<% SPHttpUtility.AddQuote(SPHttpUtility.HtmlEncode(SPWeb.OriginalBaseUrl(Request)),Response.Output); %> xmlns:q1="SPDocLibUpload\_WebService" binding="q1:SPDocLibUpload" xmlns=" http://schemas.xmlsoap.org/disco/soap/ " />*

*<soap address=<% SPHttpUtility.AddQuote(SPHttpUtility.HtmlEncode(SPWeb.OriginalBaseUrl(Request)),Response.Output); %> xmlns:q2="http://tempuri.org/" binding="q2:ServiceSoap12" xmlns="http://schemas.xmlsoap.org/disco/soap/" />*

*</discovery>*

*The changes in the wsdl file occur in one of the last nodes of the file*

*<%@ Page Language="C#" Inherits="System.Web.UI.Page" %>   
<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral, PublicKeyToken=71e9bce111e9429c" %>   
<%@ Import Namespace="Microsoft.SharePoint.Utilities" %>   
<%@ Import Namespace="Microsoft.SharePoint" %>  
<% Response.ContentType = "text/xml"; %>*

*<wsdl:service name="SPDocLibUpload">*

*<wsdl:port name="SPDocLibUploadSoap" binding="tns:SPDocLibUploadSoap">*

*<soap:address location=<% SPHttpUtility.AddQuote(SPHttpUtility.HtmlEncode(SPWeb.OriginalBaseUrl(Request)),Response.Output); %> />*

*</wsdl:port>*

*<wsdl:port name="SPDocLibUploadSoap12" binding="tns:SPDocLibUploadSoap12">*

*<soap12:address location="http://vs-sharepoint1:42745/\_layouts/SPDocLibUpload.asmx" />*

*</wsdl:port>*

*</wsdl:service>*

That is it. You should be able to see this web service from any site in SharePoint. When adding a reference in Visual Studio, your link would be:   
[http://vs-sharepoint1:<port>/\_vti\_bin/SPDocLibUpload.asmx?WSDL](http://vs-sharepoint1:%3cport%3e/_vti_bin/SPDocLibUpload.asmx?WSDL)

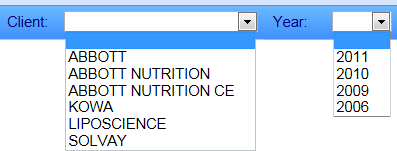
Where ‘<port>’, is the port number of the SharePoint site where the web service is located. So, in this example, the port would be 17074 for the MEI Universal SharePoint site.

# Intranet Site

At the time of writing this document, the intranet site used for these documents is located at <https://development.meintl.com/MEIIntranet/>

## Client / Year Dropdowns

The intranet site is a central location for managing all documents across all MEI clients (Abbott, Ross, etc.). Access to the clients is controlled by two dropdowns located in the top right hand corner of the site.



Essentially, this just changes the database you hit when doing queries and such. If you select ‘Abbott’ and ‘2009’ from the database, then you will grab the key ‘Abbott2009’ out of the web.config and use it. That web.config entry should be a connection string to the Abbott2009 database.

The clients that are in the Client dropdown are dynamic in that they will be based on the Active Directory groups that the user is a member of.

## Menu, Sitemap, & Breadcrumbs

The left-hand site menu, the sitemap page, and the breadcrumbs are all built from the Web.sitemap file in the root of the website.

The breadcrumbs will automatically be built from the Web.sitemap as the pages are navigated to.

The Web.sitemap file is made up of SiteMapNodes. These nodes can be nested in any fashion and the menu on the left of the master page will automatically reflect it. Each node can consist of a URL, title, and description. The URL, if given, is the location that will be navigated to if that particular node is clicked. The title is the text that will be displayed on the node. The description is the text that will be displayed as a tooltip if the cursor pauses over the node.

The Sitemap web page is essentially a dynamic menu. Using the Web.sitemap file, it builds an unordered list type representation of the sitemap with each item acting as a hyperlink. These items, when clicked, function in the same manner as the menu on the left of the master page. That is, they use their respective URL’s from the Web.sitemap to perform the navigation. A special feature of the Sitemap web page is that a ‘GroupName’ can be passed to it in the query string. When the Sitemap web page loads, it will use this ‘GroupName’ to build its representation of the sitemap. That is, the ‘GroupName’ specified will filter the nodes that are displayed. It will cause only the nodes that are under it to be displayed. The ‘GroupName’ must be the title of one of the top level nodes in the Web.sitemap file.

Notice that the top level nodes in the Web.sitemap all have URL’s that point to the Sitemap web page. But, individually they all pass their respective title in the querystring as the ‘GroupName’ so that the sitemap generated on the Sitemap web page will only display nodes which are descendents of their self.

# Business Data Columns

Business data column setup can take a bit of extra work at first, but they can be very useful in centralizing data across SQL Server & SharePoint. We’ll use the example of SpeakerCheck 🡪 CheckType to explain the process.

## Making SharePoint Aware of SQL1

Problem: We only want to allow a set list of values for the CheckType column in the SpeakerCheck document library. The first solution that may come to mind is to use the ‘Choice’ datatype and set the value list in SharePoint. That will work perfectly fine for one site. However, now we have 4 or 5 client SharePoint sites, all with SpeakerCheck document libraries, all with CheckType columns, and all with SEPARATE CheckType lists we have to maintain. The process of adding a CheckType to that list across all the sites is a tedious process.

Solution: Use the Business Data Catalog. We will create a CheckType table in the Event database on SQL1, then make all of the CheckType columns across all the sites, and reference that table on SQL1.

Using the Business Data Catalog (BDC) involves the process of making SharePoint aware of databases on SQL1 and defining methods that SharePoint can perform against tables. Luckily, this process can be automated using a couple of different tools.

The first tool is free and comes with the MS Office Server SDK and is called the BDC Definition Editor. At the time of writing this document, I haven’t been able to get this product installed (but I hear once you do it’s kind of nice). The second tool, and the one I’ve been using with great success, is ‘BDC Meta Man’. A company called LightningTools produces it. The free developer version comes with limitations, but they don’t matter too much if you can manually combine XML files correctly.

We’ll start with the process of making SharePoint aware of the CheckType table on SQL1. Open up BDC Meta Man and select SQL Server from the ‘Connect to data source:’ dropdown. Once you get connected, you should see all of the databases appear in the list below the dropdown. Go to the Event database and drag the CheckType table into the workspace on the right. A green play button arrow should appear on the toolbar above that ‘Connect to data source’ dropdown. Click this button to create the XML file describing the CheckType table. You’re done in Meta Man so go ahead and exit.

Now, go open the .XML file you just created, there is a field we need to change. If you go to the path //LobSystem/LobSystemInstances/LobSystemInstance/Properties/Property[@Name]=’AuthenticationMode’, you will want to change the value from PassThrough to RevertToSelf. This will allow SharePoint to have access to SQl1.

Open SharePoint Central Administration and under Shared Services Administration heading on the left hand menu, go to SharedServices2, Under the Business Data Catalog sub-heading go to Import application definition. Note: You may go to View applications and delete the application if it already exists (i.e. if you’ve already added with database once)

Your settings will most likely be:

File Type Model  
 Resources to Import Localized Names | Properties

You should now be able to View the application, go to CheckType entity and view its profile. From there you can send CheckTypeID in the query string and see information straight from SQL1.

## Creating the Document Library Column

Go ahead and open up a SpeakerCheck document library and go to its settings page 🡪 Create column. You settings will need to be something along the lines of:

Column Name CheckTypeBDC  
Column Type Business Data  
Column Settings Type CheckType  
Display field for Type CheckTypeName  
Display Actions Menu False  
Link to profile page False  
Add to Default View True

Once you type in, CheckType, for the column settings type, you may need to click the ‘check document’ icon next to it to get you display field choices.

What you are saying here by selecting these column settings is that the underlying value for the CheckTypeBDC column will be the CheckTypeID (this is defined in the XML we uploaded) column, and the field to actually display is the CheckTypeName column.

Once you click OK, you are finished creating the column

If you are manually adding an item to the document library, you’ll need to enter the CheckTypeID value for the CheckTypeBDC column. SharePoint will query the table itself to get the display value.

## Adding a BDC Column to the MEI\_SP\_Documents Project

When inserting or updating an item in a document library programmatically that contains a BDC column, special values must be used. In reality, when you add a Business data column, SharePoint adds more columns that you don’t see, and you have to take all these columns into account when editing an item.

Finding the name for the hidden column that you need, can be a bit difficult, but it does follow some standards so you can try these first, and find alternative means if this doesn’t work,

Since the title of our application entity is CheckType, a column is created called dbo\_x002e\_CheckType\_ID. The \_x002e\_ is just SharePoint’s interpretation of a period. Now that you have the column name, you have to properly format the data going into it. You may think that you simply need to pass in the CheckTypeID column from the table in the Event database. Well, you’re partially right.

There is a function in the DocumentBroker class called GetBDCColumnProperID. You first need to pass the CheckTypeID column into this function and the result is the value you’ll need to pass to SharePoint. If you want to know what this does, look it up in the upload web service I created and Google it.

In addition to using the dbo\_x002e\_CheckType\_ID column, you’ll also want to use the normal CheckTypeBDC column. Remember earlier when you were creating the column that you set the display column to the CheckTypeName column in the CheckType table on SQL1. This CheckTypeName is the value you’ll want to use for the CheckTypeBDC column.

For a more visual reference, check out the GetUploadXML & GetSearchXML functions of the SpeakerCheck object in the MEI\_SP\_Documents Project.

# The MEI\_SP\_Documents Project

When you first get the project out of source control, be sure to also get the latest version of the ‘Third Party’ Team Project. This is due to the fact that all of the references to third party libraries in this project relatively point to the libraries in the ‘Third Party’ Team Project and this will ensure that the project can resolve these relative paths.

All functionality for creating document objects, and searching document libraries, and uploading documents is routed through the DocumentBroker. When you instantiate a SP\_Documents object (which is your first step in using the project) you will have access to the DocumentBroker.

# Known Issues

There is one issue I know about with the SharePoint server but it’s sporadic and I haven’t figured out how it happens. Sometimes, processes become deadlocked and frozen when calling the web services on vs-sharepoint1. This can cause the SharePoint sites to not load, and essentially disable the uploading and searching of documents. I have only seen this happen once during the stint of the project, however.

There is an article on MSDN that deals with possible causes and fixes for this problem. It will be worth looking over, as precautions may have to be taken in the future. <http://support.microsoft.com/default.aspx?scid=kb;en-us;821268>

# Document File Name Standards and Column Names

#### Abbott Document Search Document

Columns: DocumentSearchDocumentTypeId

DocumentTitle

UploadUserName

FileName: ADSD\_\_<DocumentTitle>\_\_<DocumentSearchDocumentTypeId>.pdf

#### Abbott Document Search Slidekit

Columns: DocumentSearchProductId

SlidekitTitle

UploadUserName

VersionNumber

FileName: ADSSK\_\_<SlidekitTitle>\_\_<VersionNumber>.pdf

**Banquet Event Order**

Columns: Program ID  
VendorID

FileName: BEO\_\_<Program ID>\_\_<VendorID>.pdf

**Budget Worksheets**

Columns: Program ID

FileName: BudgetWorksheet\_\_<Program ID>.pdf

**Check Request**

Columns: Program ID  
SpeakerCounter  
ExpCounter

FileName: CheckRequest\_\_<Program ID>\_\_<SpeakerCounter>\_\_<ExpCounter>.pdf

#### Credit Card Authorization

Columns: Program ID

FileName: CCAuthorization\_\_<Program ID>.pdf

**Expense Report**

Columns: Program ID  
SpeakerCounter

FileName: ExpenseReport\_\_<Program ID>\_\_<SpeakerCounter>.pdf

**PIF**

Columns: Territory  
PifID

FileName: PIF\_\_<Territory>\_\_<PifID>.pdf

**Program Vendor Menu**

Columns: Program ID  
VendorID  
MenuType  
Annotations

FileName: ProgramVendorMenu\_\_<ProgramID>\_\_<VendorID>\_\_< MenuType>.pdf

**PSA**

Columns: Agreement ID  
SpeakerCounter  
SpeakerName  
Program ID <= Not supposed to be in filename

FileName: PSA\_\_<Agreement ID>\_\_<SpeakerCounter>\_\_<SpeakerName>.pdf

**Receipt**

Columns: Program ID  
ExpenseCounter

FileName: Receipt\_\_<ProgramID>\_\_<ExpenseCounter>.pdf

**Reservation Checklist**

Columns: Program ID

FileName: ReservationChecklist\_\_<Program ID>.pdf

**Sign In Sheet**

Column: Program ID

Filename: SignInSheet\_<ProgramID>.pdf

**SpeakerCheck**

Columns: Program ID  
SpeakerCounter  
Expense Counter  
CheckTypeBDC

FileName: SpeakerCheck\_\_<Program ID>\_\_<SpeakerCounter>\_\_<ExpCounter>\_\_<CheckTypeBDC>.pdf

The CheckTypeBDC value is the CheckTypeAcronym value from Event.CheckType on SQL1.

**Thank You Letter**

Column: ProgramID  
 SpeakerCounter

FileName: ThankYouLetter\_\_<ProgramID>\_\_<SpeakerCounter>.pdf

**Travel Itinerary**

Columns: Program ID

FileName: TravelItinerary\_\_<Program ID>.pdf

**Vendor Menu -** This document is only setup in MEI Universal

Columns: VendorID  
StartDate  
EndDate  
MenuType  
Annotations

FileName: VendorMenu\_\_<VendorID>\_\_<StartDate>\_\_<EndDate>\_\_<MenuType>.pdf

MenuType is a list of characters representing the actual designations

Breakfast = B  
Lunch = L  
Dinner = D  
Holiday = H  
Drink = K  
Dessert = T

So, a menu that was classified as a Breakfast, Lunch, Dinner, & Drink, menu would have a MenuType value of `BLDK`.

The Date components here should be in the format `mmddyyyy`

5/24/2007 = 05242007

**Venue Contract**

Columns: ProgramID  
VendorID

FileName: VenueContract\_\_<ProgramID>\_\_<VendorID>.pdf

**W9**

Columns: SpeakerCounter  
Year  
TIDSSN

FileName: W9\_\_<SpeakerCounter>\_\_<Year>\_\_<TIDSSN>.pdf