Architecture and Design

# Web application

The Velodoc XP Edition web application follows the standard architectural guidelines of a typical ASP.NET web application with three specificities which we describe below:

* An http module for streaming the upload of large files and reporting on progress;
* An http handler for providing resumable downloads;
* ASP.NET Ajax web controls for building professional UIs with upload functionality.

# Upload module



The browser issues a GET request to load the ASP.NET page which is rendered as HTML. This page contains two server controls which are MultiUpload and ProgressReport. The upload module is not bound to the MultiUpload or ProgressReport server controls, but they are shown here to explain how they work together.

The user selects files to upload and submits the page. The browser issues a POST request to the same ASP.NET page. The POST request is intercepted by the Upload module which checks that the ENCTYPE is “multipart/form-data” instead of “application/x-www-form-urlencoded”.

If this is the case, the upload module looks for uploaded files in the request and streams them through the storage provider. As the upload module receives data and streams files, it updates an UploadData object maintained in context cache and identified by a unique identifier called muid with “real-time” data about the size of the request, how many bytes have been processed, the number and names of files, ...

At the same time, a ProgressReport control hosted on the page issues periodic GET requests to an upload handler, passing the muid in query string, in order to obtain server feedback about the progression of the upload. The upload handler retrieves the upload data identified by the muid in context cache and returns progress data which is displayed by the ProgressReport control.

# Download handler

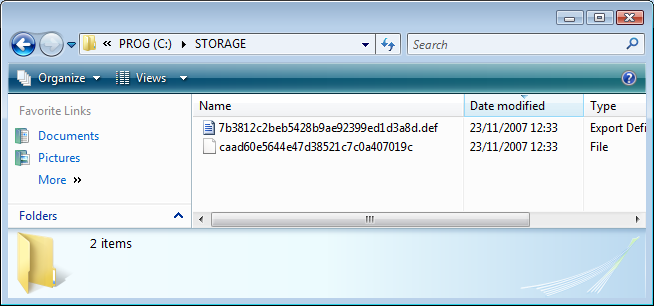
When a page with files to upload makes a multipart/form-data POST request to the server, files are saved to a storage directory whether using the upload http module or not.

If you want your storage to be secure, you at least need to:

1. Remove the file extension, so that the files cannot be easily executed;
2. Rename the files with random names which users never see.

If your stored files are named like caad60e5644e47d38521c7c0a407019c (key), how are you going to restore their original name and extension for users to download them? You need a way to map the key, under which they are named and identified in the storage, the original file name and the content type.

You can use a database like in Velodoc Enterprise Edition or a file which contains the mapping, which is the solution implemented in Velodoc XP Edition.



Such files have the .def extension and their Xml content is similar to the following:

<?xml version="1.0" encoding="utf-8"?>

<File xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.memba.org/2006/attachments">

<Guid>7b3812c2-beb5-428b-9ae9-2399ed1d3a8d</Guid>

<FileName>Important Document.pdf</FileName>

<ContentType>application/pdf</ContentType>

<Key>caad60e5644e47d38521c7c0a407019c</Key>

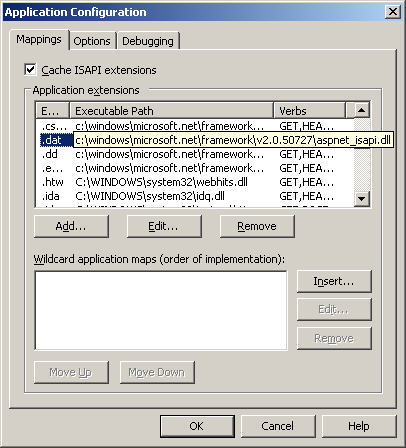
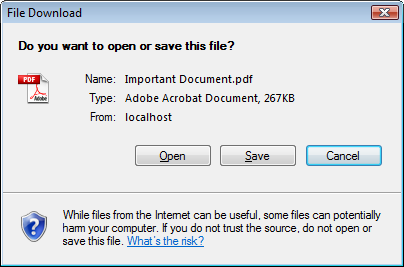
<Size>274308</Size>

<HashValue>474ab5ca8ee6909be1eef19508cde94f9240d924</HashValue>

<CreatedOn>2007-11-23T12:33:45.7382102Z</CreatedOn>

</File>

When the browser emits a GET request for http://<server>/VelodocXP/7b3812c2beb5428b9ae92399ed1d3a8d.dat where 7b3812c2beb5428b9ae92399ed1d3a8d is the public identifier of the file to download:

1. The .dat extension triggers the execution of the download handler because:
   * There is a script map recorded in IIS to trigger aspnet\_isapi.dll on .dat extension:  
     
   * The download handler is configured in web.config to get activated on .dat extension:  
     <add verb=”GET,HEAD” path=”\*.dat” validate=”false” type=”Memba.FileDownload.DownloadHandler, Memba.FileDownload.XP, Version=1.0.0.0, Culture=neutral, PublicKeyToken=a4ae091aa8097a5a” />
2. The download handler reads 7b3812c2beb5428b9ae92399ed1d3a8d.def which contains the content type, the original name and the private name of the file, which in our example is caad60e5644e47d38521c7c0a407019c.
3. Finally, the download handler transmits the file under its original name and content type:  
    

As regards supporting Etag and Accept-Ranges http headers to provide resumable downloads, the download handler is a modified C# implementation of the download handler described by Joe Stagner in an MSDN Magazine article dated September 2006 and entitled [Web Downloads: Build Smarter ASP.NET File Downloading Into Your Web Applications](http://msdn.microsoft.com/msdnmag/issues/06/09/WebDownloads/). Modifications include:

1. Raising Http exceptions instead of response status codes to automatically redirect to custom error pages;
2. Adding web events to instrument the handler;
3. Fixing an issue with Unicode file names;
4. Implementing storage providers to access files.

You can also find interesting information at:

* <http://www.devx.com/dotnet/Article/22533/0/page/1>
* <http://www.devsource.com/article2/0,1759,1877272,00.asp>

# UI components

The solution includes 3 UI components (not including the InfoBox) built with ASP.NET Ajax extensions:

1. The MultiUpload component is used to select files to upload;
2. The ImageList component is used to display files as icons with a remove button;
3. The ProgressReport component is a progress bar to display the status of a pending upload.

Regarding the architecture and design, please refer to:

* [Creating custom ASP.NET Ajax client controls](http://www.asp.net/ajax/documentation/live/tutorials/CreatingCustomClientControlsTutorial.aspx);
* The developer tutorial which you should have received with this document.

# New to version 1.1: WCF streaming service

Version 1.1 now features a WCF streaming web service to upload and download files using either:

* The Memba Velodoc Outlook Add-In;
* The client API which comes with the add-in;
* The .NET proxy which you can [generate using svcutil.exe](http://msdn.microsoft.com/en-us/library/ms734691.aspx).

The use of the Memba Velodoc Outlook Add-In and Client API is described in the relevant documentation which you can find with the source code at <http://www.velodoc.com/download>. For more information regarding the generation of a.NET proxy see the “Developer Tutorial”.

The WCF streaming service is designed according to the guidelines developed at:

* [How to enable streaming](http://msdn.microsoft.com/en-us/library/ms789010.aspx);
* [MSDN streaming sample](http://msdn.microsoft.com/en-us/library/ms751463.aspx).