## Loading, Saving and Converting

### About Document Conversions in Aspose.Words

The ability to easily and reliably convert documents from one format to another is one of the four main feature areas of Aspose.Words (the other three being: document object model, rendering and mail merge).

Almost any task that you want to perform with Aspose.Words involves loading or saving a document in some format.

The LoadFormat enumeration specifies all *load* or *import* formats supported by Aspose.Words. The SaveFormat enumeration specifies all *save* or *export* formats supported by Aspose.Words. Aspose.Words can convert a document from *any* load format into *any* save format making the total number of possible conversions very large.

Converting from one document format to another in Aspose.Words is very easy and can be accomplished using just two lines of code:

1. Load your document into a Aspose.Words.Document object using one of its constructors. By default, Aspose.Words will even auto-detect the file format for you.
2. Invoke one of the Document.Save(String) methods on the Document object and specify the desired output format.

### Load or Create a Document

The Document class represents a document loaded into memory. Document has several overloaded constructors allowing you to create a blank document or to load it from a file or stream.

##### Creating a New Document

Call the Document.#ctor constructor without parameters to create a new blank document:

{$ExId:DocumentCtor}

The document paper size is Letter by default.

If you want to generate a document programmatically, the most reasonable step after creation is to use DocumentBuilder to add document contents.

{$ExId:DocumentBuilderAndSave}

##### Opening from a File

Pass a file name as a string to the Document.#ctor(String) constructor to open an existing document from a file:

{$ExId:OpenFromFile}

##### Opening from a Stream

Simply pass a stream object that contains a document to the Document.#ctor(Stream) constructor:

{$ExId:OpenFromStream}

##### Opening Encrypted Documents

You can open Word documents encrypted with a password. To do that, use a special constructor overload, which accepts the password string.

{$ExId:OpenEncrypted}

### Save a Document

Use the Document.Save(String) method for saving a document. There are overloads that allow saving a document to a file, stream or ASP .NET **HttpResponse** object for sending to a client browser.There are overloads that allow saving a document to a file or stream. The document can be saved in any save format supported by Aspose.Words. For the list of all supported save formats see the SaveFormat enumeration.

##### Saving to a File

Simply use the Document.Save(String) method with a file name. Aspose.Words will infer the save format from the file extension that you specify.

{$ExId:SaveToFile}

##### Saving to a Stream

You pass a stream object to the Document.Save(Stream, SaveFormat) method. When you save to a stream, you must specify the save format explicitly.

{$ExId:SaveToStream}

##### Sending to a Client Browser

In order to send a document to a client browser, use a special overload that accepts four parameters: file name, save format, save type, and an ASP .NET **HttpResponse** object. Save type is represented by the **Aspose.Words.SaveType** enumeration that determines whether the document being sent to the browser will provide an option to open itself directly in the browser or in an application associated with .doc extension.

{$ExId:SaveToBrowser}

##### Specifying Save Options

Document has a **Document.SaveOptions** property that allows specifying some options that control how the document is saved.

{$ExId:SaveWithOptions}

### How-to: Convert a Document to PDF

The simplest way to convert a document to PDF is to just invoke the **Save** method and specify a file name with the “.PDF” extension.

This is the recommended method for converting to PDF because it uses the new rendering engine built into Aspose.Words and does not use Aspose.Pdf.

{$ExId:Doc2PdfSave}

### How-to: Convert a Document to MHTML and Email

You can download the complete source code of the **SaveMhtmlAndEmail** sample [here](http://www.aspose.com/community/files/51/file-format-components/aspose.words-for-.net-and-java/category1177.aspx).

Aspose.Words for .NET allows to save any document in MHTML (Web Archive) format. This makes it very easy to use Aspose.Words and Aspose.Network together to generate and send email messages with rich content. For example, you can load a predefined DOC, OOXML or RTF document into Aspose.Words, fill it with data, save as MHTML and then email using Aspose.Network.

{$ExId:SaveMhtmlAndEmail}

### How-to: Convert an Image to PDF

You can download the complete source code of the **ImageToPdf** sample [here](http://www.aspose.com/community/files/51/file-format-components/aspose.words-for-.net-and-java/category1177.aspx).

This article shows how to create a PDF document from an image using Aspose.Words. While converting images to PDF is not a main feature of Aspose.Words, this example shows how easy it is to do with Aspose.Words.

To make this code work you need to add references to Aspose.Words and System.Drawing to your project.

The code below allows converting single frame images, such as JPEG, PNG BMP, EMF or WMF, as well as multi-frame TIF images to PDF.

{$ExId:ImageToPdf}

### How-to: Detect the File Format

Sometimes it is necessary to detect the format of a document file before opening because the file extension does not guarantee that the file content is appropriate.

For example, it is known that Crystal Reports often outputs documents in RTF format, but gives them the .doc extension. Therefore, if you are not sure what the actual content of the file is and want to avoid throwing an exception, you can use the **DetectFileFormat** method. This is a static (shared) method that accepts either a file name or stream object that contains the file data. The method returns a **LoadFormat** enumeration value that indicates the file type.

{$ExId:DetectFileFormat}

### How-to: Check Format Compatibility

You can download the complete source code of the **CheckFormat** sample [here](http://www.aspose.com/community/files/51/file-format-components/aspose.words-for-.net-and-java/category1177.aspx).

When you are dealing with multiple documents in various file formats, you may need to separate out those files that can be processed by Aspose.Words from those that cannot. You may also want to know why some of the documents cannot be processed.

If you attempt to load a file into a Document object and Aspose.Words cannot recognize the file format or the format is not supported, Aspose.Words will throw an exception. You can catch those exceptions and analyze them, but Aspose.Words also provides a specialized method that allows to quickly determine the file format without loading a document with possible exceptions.

This article describes how you can check the format compatibility of all files in the selected folder and sort them by file format into appropriate subfolders.

##### Solution

To do this, we will work through the following steps in the code:

1. Get the collection of all files in the selected folder.
2. Loop through the collection.
3. For each file:
   1. Check the file format.
   2. Display the check results.
   3. Move the file to the appropriate folder.

The following files are used in this sample. The file name is on the left and its description is on the right.

###### To test supported file formats:

Test File (docx).docx Office Open XML WordprocessingML document without macros.

Test File (docm).docm Office Open XML WordprocessingML document with macros.

Test File (doc).doc Microsoft Word 97 - 2003 document.

Test File (rtf).rtf Rich Text Format document.

Test File (dot).dot Microsoft Word 97 - 2003 template

Test File (dotx).dotx Office Open XML WordprocessingML template.

Test File (HTML).html HTML document.

Test File (MHTML).mhtml MHTML (Web archive) document.

Test File (WordML).xml Microsoft Word 2003 WordprocessingML document.

Test File (odt).odt OpenDocument Text format (OpenOffice Writer).

###### To test encrypted documents:

Test File (enc).doc Encrypted Microsoft Word 97 - 2003 document.

Test File (enc).docx Encrypted Office Open XML WordprocessingML document.

###### Unsupported file formats:

Test File (pre97).doc Microsoft Word 95 document.

Test File (JPG).jpg JPEG image file.

Test File (XML).xml The Word XML Document.

##### The Code

As we’re dealing with the content in a folder, the first thing we need to do is to get the collection of all files in this folder using the GetFiles method of the Directory class (from the System.IO namespace):

{$ExId:CheckFormat\_Folder}

When all the files are collected, the rest of the work is done by a single method within the Aspose.Words component – Document.DetectFileFormat. The DetectFileFormat method checks the file format, but note that it only *checks* the file format, it does not *validate* the file format. This means that there is no guarantee that the file will be opened even if DetectFileFormat returns that it is one of the supported formats. This is because the DetectFileFormat method reads only partial data of the file format, enough to check the file format, but not enough for complete validation.

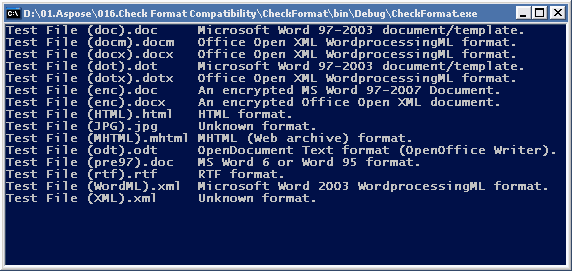
The following code loops through the collected list of files, checks the file format of each file, displays them in the console and moves each file into the appropriate folder:

{$ExId:CheckFormat\_Main}

The files are moved into appropriate subfolders using the Move method of the File class, from the same System.IO namespace.

##### End Result

The sample moves all the files to subfolders and displays the following log:



### How-to: Load and Save a Document to Database

[This article is a draft. It has not yet been reviewed and subject to change.]

You can download the complete source code of the **DocumentInDB** sample [here](http://www.aspose.com/community/files/51/file-format-components/aspose.words-for-.net-and-java/category1177.aspx).

One of the tasks you may need to perform when working with the documents is storing the Document objects into a database. You would definitely need to do this for example if you were implementing some kind of content management system, when you would have to store all previous versions of the document as well as the current version. The ability to store documents in the database is also extremely useful when your application provides the web-based service.

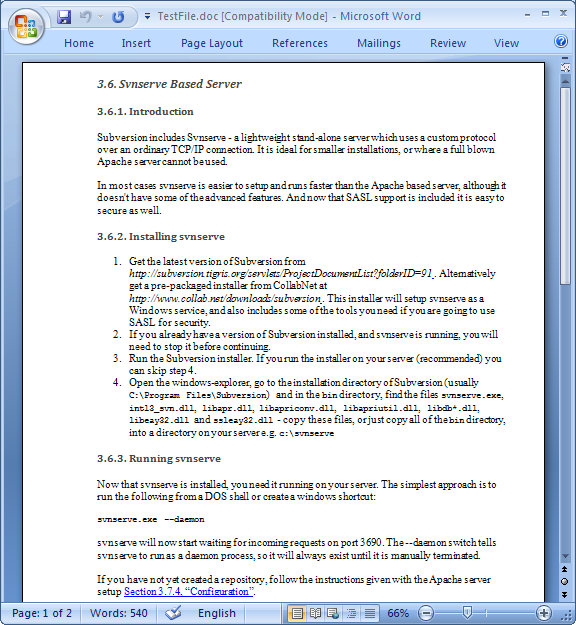
This sample shows how to store a document into a database and then load it back. For the sake of simplicity, the name of the file is used as a key to store and fetch documents from the database.

##### Solution

We will do the following to store, read or delete document in the database:

1. To store a document into a database:
   1. Save the source document into a memory stream. It will allow us to get the content of the document as an array of bytes.
   2. Store the array of bytes into a database field.
2. To read a document from a database:
   1. Select a record that contains the document data as an array of bytes.
   2. Load the array of bytes from the data record into a memory stream.
   3. Create a Document object that will load the document from the memory stream.

The following Word document is used in this sample:



##### The Code

There are three methods implemented in this sample:

* The StoreToDB that stores the Document object into the database.
* The ReadFromDB that reads the Document object from the database.
* The DeleteFromDB that deletes the record from the database.

Let’s start with the StoreToDB method. Here it is:

{$ExId:DocumentInDB\_StoreToDB}

At first, you need to create a connection to the database. In this sample we are using a Microsoft Access .mdb database to store an Aspose.Words document. Then you should create a memory stream and save the document there using the Save method of the Document class. The next step is the most important: specify the commandString, which is an SQL expression that will do all the work. For the storing use the “INSERT INTO” command, specify the table and supply the values of two record fields – FileName and FileContent. To avoid additional parameters, the file name is acquired from the Document object itself. The FileContent field value assigned to the memory stream buffer, which actually contains the binary representation of the stored document. The remaining lines of code are quite simple, as we are simply opening the created connection to the database, executing the command and closing the connection. And that’s it, the work is done, the Aspose.Words document is stored to the DB.

The ReadFromDB method has a lot in common with StoreToDB, although there are several noticeable differences due to the fact that here we need to gather some data from the DB, rather than add data into it.

{$ExId:DocumentInDB\_ReadFromDB}

The first difference is the SQL command. Here you should use the “SELECT \* FROM” expression and fetch the necessary record using the filename. The next point of interest is that due to the fact that we need to collect the data from the database in the DataSet object, to fill the DataSet it is necessary to use an OleDbDataAdapter instance initialized with our SQL expression. Then gather the data with the connection opened and make the necessary checks to ensure that the requested data is actually extracted. In the final steps, the procedure of the document storing is reversed – here you should read the binary buffer, put it into the memory stream and load the Document object from that stream. Then all you have to do is to save the document with the appropriate name to the appropriate folder.

The final method is the DeleteFromDB. This is quite straightforward as there are no manipulations with the Document object.

{$ExId:DocumentInDB\_DeleteFromDB}

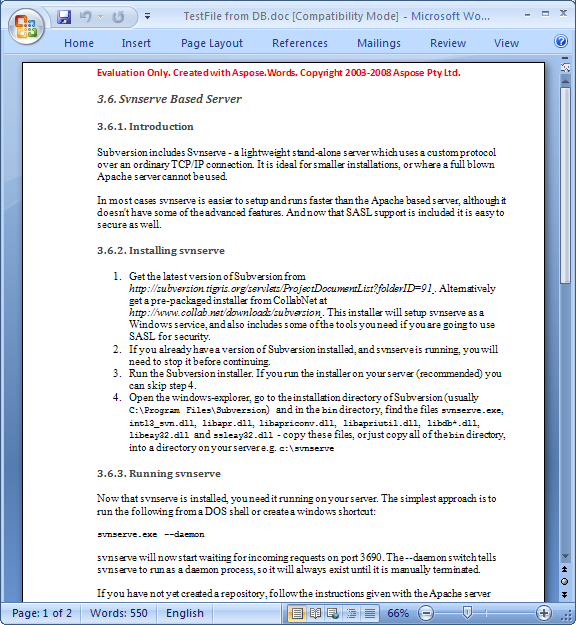
That’s all there is to it: create the connection, specify the SQL command “DELETE \* FROM” using the filename to point out the record, execute the command with the opened connection. Voila, the document is deleted from the database.

To illustrate all three methods we have called them all in a row:

{$ExId:DocumentInDB\_Main}

##### End Result

When the code executed you will see the following document in the Data folder:



### How-to: Load Plain Text (TXT) Files

You can download the complete source code of the **LoadTxt** sample [here](http://www.aspose.com/community/files/51/file-format-components/aspose.words-for-.net-and-java/category1177.aspx)[here](http://www.aspose.com/community/files/72/java-components/aspose.words-for-java/category1377.aspx).

Currently, Aspose.Words supports saving documents as plain text files, but does not directly support loading them. However, it is easy to write your own code to load plain text files into Aspose.Words.

{$ExId:LoadTxt}