Chem4Word Architecture

When we set out to design Chem4Word, we were somewhat restricted in our choice of technologies than we would have been were we writing a standalone software tool. The Microsoft Office ‘technology stack’ imposes restrictions over and above those encountered when designing an application that was targeted solely at the Windows platform. From 2007 onwards, Word documents are XML files residing in a compressed, ‘zipped’ folder structure. Although it is possible to modify these files using simple programming techniques, programming *within* Word (i.e. when the document is also opened for editing by the Word application) precludes this kind of approach.

A decision therefore was taken early on not to try to force-fit the existing set of chemical software tools into the Word programming model. Chem4Word was to be implemented as a Word add-in module, which would allow users to directly access the chemistry from within the Word user interface. This would require that we either interface directly with JUMBO[[1]](#footnote-1) (which was written in Java) or rewrite it to integrate into the add-in. Originally, Word add-ins were developed in an application-specific dialect of Visual Basic, *Visual Basic for Applications (VBA)*. More recently, the introduction of Visual Studio Tools for Office (VSTO) allowed a wider choice of programming languages such as C#, which was a more natural step in the evolution of the JUMBO toolkit due to its similarity to Java.

JUMBO was therefore rewritten from scratch in C# as .Numbo (‘.NET Jumbo ‘). This has proven to be an excellent decision as it allowed us to introduce clean software paradigms into the system, such as declarative programming and stateless systems. These make it easier to navigate the architecture, avoid close-coupling, remove the need for dirty flag management and other synchronisations and also potentially allow developers to modularise and parallelise the code.

Since each function call was stateless, every change to the system would have to be reflected in the underlying XML, this functioning as the only data structure in the system. Every operation takes in the XML, possibly alters it, and emits changed XML. This allows us to know the complete state of the system by simply inspecting the XML, but it can be very much slower for certain operations. Complex analyses, such as determining cyclic bonds, can run up to a million times slower. As this system was designed to run in an interactive manner, this was much less likely to be a factor in *perceived* performance.

We were also keen to use the model-view-controller (MVC) pattern in the application’s design. This pattern imposes a logical demarcation of roles between the components of the system responsible for storing its state (the model), those responsible for interacting with the user (the view), and the remaining functions that keep the two in synchrony (the controller). The MVC pattern implicitly allows multiple views onto the same model, so we would be able to use the same CML for multiple representations of chemistry within the document. Changes to the underlying chemistry in one location would therefore be reflected in all locations. In treating the CML as the authoritative representation of the chemistry, MVC makes the chemistry a first-class object in its own right, as opposed to simply being a ‘backing store’ for graphical representations of embedded molecules. The data itself becomes much more tractable and amenable to mining and indexing inside or outside organization repositories.

Microsoft Word, although, as noted above, limiting the programmer’s approach, nevertheless makes available a rich set of its own components for use in custom solutions. ‘Content controls’ are a recent innovation: these are specially demarcated areas of document content that allow attachment of custom functionality. The available set of content controls allows encapsulation of text (both plain and richly-formatted), and pictures. Chem4Word therefore makes extensive use of content controls to represent and manipulate embedded chemistry. Much of Chem4Word’s logic concerns mediating the flow of information between the content controls and the XML representation of the underlying chemistry.

# The Data Model

The decision to persist all data in an underlying XML format allowed us to exploit several features of the Word document model. Microsoft Word 2007 and later versions store their documents as DOCX files. These are simply folder structures containing XML documents and packaged into a single file using industry standard compression techniques. They can therefore be opened and inspected using a commercial tool such as WinZip.

The structure of a generic DOCX file is shown below.

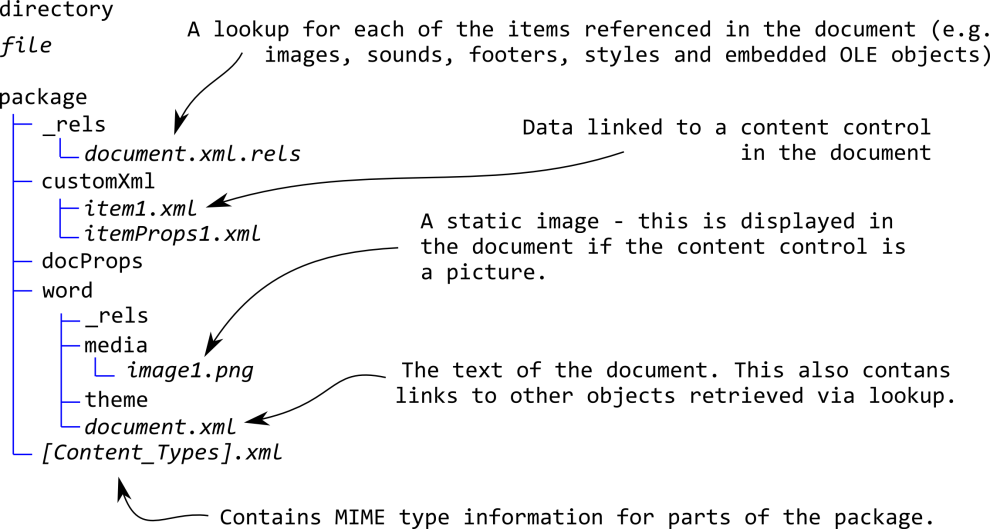


Figure 1: Structure of a DOCX File

The most important feature from our point of view is the *customXml* folder. This stores XML content that is external to the primary *document.xml* and which cannot be manipulated by the Word application itself, known in Word terminology as a ‘custom XML part’. It allows us to store each separate molecule as a CML in its own file. Moreover, each file can be bound directly to a content control.

The *word* folder houses the document itself. It contains a *media* subfolder which stores images. As Chem4Word uses a picture content control to represent two-dimensional chemical structures, we update the picture of the molecule when the chemistry itself is updated and store these pictures in this folder separate from the chemistry itself. This allows users who do not have the add-in installed to still view the chemistry. It also removes the requirement for the application to render chemical structures on loading a chemically-enabled document.

# Chem4Word Services

The Chem4Word add-in calls upon a number of services to realise the above goals. Each service has been carefully designed to provide one kind of functionality and so that logically-distinct functions, such as chemistry and geometry, are kept separate.

The top-level layer is the *Chem4Word add-in*. This is a VSTO component written in C#. When Word is started and the add-in is loaded, it creates single *CoreClass* component. The CoreClass will maintain associations between the document, its content controls and the underlying data model. It will also intercept user actions that change the document in some way, for instance on inserting or deleting a molecule.

The CoreClass is supported within the *Chem4Word.Core* **set** of functions and objects. These in turn call on the services of *Chemically Open Architecture (COA)*. COA serves as a bridge to several other services, namely *CID* (chemical manipulation), *Chemical Intelligence* and Numbo. Numbo itself relies on *Euclid* to provide geometry services.

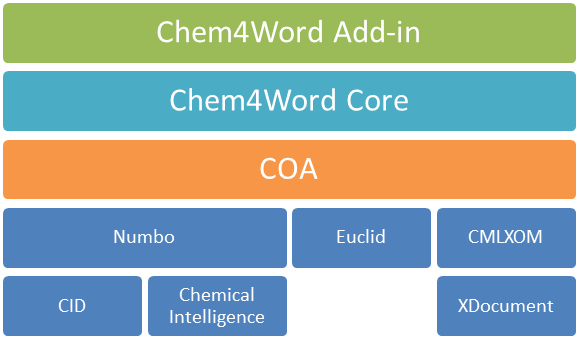


Figure 2: Services within Chem4Word

## Context Objects

On loading a document, the CoreClass will identify all existing content controls and attempt to identify the custom XML part that stores the underlying chemistry. Each underlying chemistry object is then stored in a ‘context object’, an in-memory container for the Chemistry Mark-up Language[[2]](#footnote-2) XML file (CML) that represents the molecule. On importing the XML, it’s important to ensure that it is valid according to the conventions of CML in use within Chem4Word. As CML is a large convention spanning spectroscopy crystal structures and several other domains apart from structural chemistry, we validate the XML against a reduced convention we refer to as *CML-Lite[[3]](#footnote-3)*.

Typically, in response to a user action such as moving an atom or changing its type, a context object is passed to Numbo, and further calls are made to Chemical Intelligence to query the underlying chemistry, and CID to manipulate it. The modified context object is then redisplayed in the document.

The CML itself is persisted in a .NET Framework structure, the *XDocument*.XDocuments are in-memory representations of XML document trees, and have the advantage of being lightweight and easy to program against. Each chemical entity/context object resides in its own separate XDocument, ensuring isolation of distinct chemistry. XDocuments are not manipulated directly but through an intermediate *Chemical Mark-up Language XML Object Model* (CMLXOM). As the Chem4Word uses a stateless programming paradigm, any manipulation of the chemistry by the user is immediately reflected in the contents of the underlying XDocument through CMLXOM.

## Depiction Options

It is important to note that each context object may have more than one representation within the document: there may be more than one content control displaying the object’s chemistry. Each representation may differ from the others: for instance, aspirin may be displayed as a drawn structure, as ‘acetylsalicylic acid’ its IUPAC name ‘2-acetyloxybenzoic acid’, Hill notation C9H8O4, etc. Chem4Word therefore maintains a separate ‘depiction option’ for each representation of the molecule. If the user were to update the structure (and therefore the underlying chemistry), then it is straightforward to refresh all the structural representations of that molecule within the document without disturbing other representations.

Depiction options also allow different elements within the add-in to adopt whichever display convention is most appropriate. The Chemistry Navigator element allows the user to see a list of chemical structures contained within the document: these are displayed as two-dimensional structures captioned by their Hill notation formula, but in the document the chemical may well be referred to by trivial name. Both will refer to the same underlying context object. The diagram below shows how these elements cooperate:



Figure 3: Context objects and Depiction Options

Depiction options are also important for allow the user to pick from a selection of available representations. The View button on the Chemistry Ribbon will display all available depiction options for the selected molecule and allow the user to change it without effectively changing the underlying chemistry. This serves to differentiate Chem4Word from other tools which do *not* preserve the underlying semantic information:

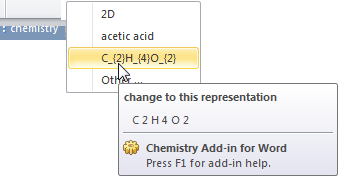


Figure 4: Changing the representation of a chemical entity using the Ribbon

# The Gallery

The Word user interface has, from 2007, used ‘galleries’ to organise sets of complex information that can be chosen at will by users. Galleries are presented as part of the new Fluent User Interface. They are a natural solution to storing and frequently accessing common chemicals. Chemical structures are stored as native CML files in the application’s data folder and identified using a Globally Unique Identifier (GUID). Users can select structures in the document and save these to the gallery.

Behind the scenes, Chem4Word uses a custom Word document template and structures are saved to this as ‘building blocks’. The Gallery displays both name and structure. Moreover, Word allows developers to define ‘smart tags’ – predefined blocks of text – that can be used to automatically recognise the occurrence of a chemical within a document. When a structure is saved to the gallery, a smart tag is created for that structure and if the user inserts this text at a later date, the text is automatically recognised and tagged.

# The Editor

Part of the philosophy behind Chem4Word has been to regard the editing of structures ‘from scratch’ to be a last resort. Users should be encouraged to exploit the ever-growing corpus of publically-available structural information before having to draw their own. The focus has therefore been very much on querying and retrieving structures from public sources.

Nevertheless, when a structure has been imported, Chem4Word allows a degree of editing. Atoms can be repositioned, atom and bond types can be changed and isotope labels assigned. Clicking on an atom brings up a ‘ring menu’ showing the possible operations that the user can perform. Currently, editing options are currently more restricted than one would expect from other commercial and open-source chemical editors. One can delete bonds and atoms, for instance, but not add new ones.

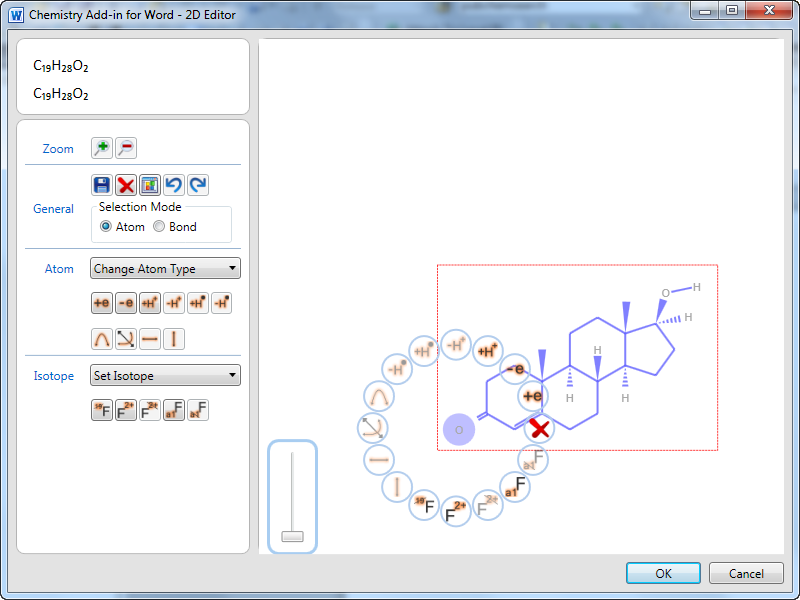


Figure 5: Chem4Word's editor

We have constructed the Editor component with the elements of *Windows Presentation Foundation (WPF)*, a Microsoft framework for the construction of modern, elegant and highly responsive user interfaces. Developers can use a specialised dialect of XML to rapidly construct and prototype WPF interfaces.

Development in VSTO (rather than VBA) allows access to the full gamut of WPF services. The ‘declarative’ nature of WPF programming, where one tends to specify the outcome one desires rather than programming the steps involved, leads to simpler and more compact code that is easier to maintain. WPF also makes heavy use of ‘data binding’, where the developer simply specifies relationships between interface elements and the underlying data model and relies upon the framework to update one in light of changes to the other.

WPF’s sophisticated graphics facilities also enhanced the perceived performance: much of the graphic rendering is done in hardware and this makes the user experience more responsive. It also provides a wide selection of graphics objects that can be easily harnessed to displaying and updating atoms and bonds. Chem4Word displays static 2D representations not as a collection of these primitive objects but as single PNG images (since rendering structures whenever the view was changed would be prohibitively expensive). WPF makes these images straightforward to generate as the items are simply arranged on a *Canvas* object, and this object itself is rendered (with its contents) in one step. This would typically occur when a structure is inserted or updated. Changes to the backing CML are also written back to the underlying context object during this process.

# Importing Structures

Most of the structures in a Chem4Word document will have originated from an external repository. Chem4Word provides two ‘out-of-the-box’ connections to *PubChem[[4]](#footnote-4)* and OPSIN[[5]](#footnote-5) (a Cambridge University structural server). These services, and services like them, will allow interrogation through HTTP interfaces, and require the user to supply the name of a compound. In the case of *PubChem*, the ‘response ‘feed’ consists of XML containing information in a proprietary format. However, as this format is essentially XML then it is tractable to being converted to CML using *Extensible Stylesheet Language Transformations (XSLT).* An XSLT stylesheet (which is itself a simple XML document) specifies the mapping between source and destination formats and allows the backing CML to be directly generated from the result of a search.

We hope that as more public structural repositories come on-line, we will be able to accommodate their search results using this simple approach or, better still, that they directly use CML to publish structural information.

1. <http://www-pmr.ch.cam.ac.uk/wiki/Jumbo> [↑](#footnote-ref-1)
2. <http://www.xml-cml.org/> [↑](#footnote-ref-2)
3. <http://www.xml-cml.org/schema/cmllite.html> [↑](#footnote-ref-3)
4. <http://pubchem.ncbi.nlm.nih.gov/> [↑](#footnote-ref-4)
5. <http://opsin.ch.cam.ac.uk/> [↑](#footnote-ref-5)