PROJECT IN COMMUNICATION SYSTEMS
STOCKHOLM, SWEDEN 2015

# Adding citations to PowerPoint slides from Zotero

*Making referencing in PowerPoint easier*

RUBEN LEJEUNE

**KTH ROYAL INSTITUTE OF TECHNOLOGY**
***INFORMATION AND COMMUNICATION TECHNOLOGY***

# Adding citations to PowerPoint slides from Zotero

*Making referencing in PowerPoint easier*

Ruben Lejeune

2015-08-28

IK2553 Project Report

Examiner and Academic adviser
Gerald Q. Maguire

KTH Royal Institute of Technology
School of Information and Communication Technology (ICT)
Department of Communication Systems
SE-100 44 Stockholm, Sweden

# Abstract

When writing papers, it is frequently necessary to add citations. A frequently used tool to facilitate this process is Zotero. Currently one can add citations using the Zotero Word integration extension to a Word document, however there is not yet a tool to do so in PowerPoint – despite the fact that in many settings this would be very desirable. This project solves this problem.

The first difficulty was that this was the first time I had to analyze a software project of this magnitude (specifically the Zotero and Zotero Word integration software). This meant I had to understand their structure and find out what pieces of the software would be useful to me. The second difficulty was to understand and use some of the tools used by Zotero and Zotero Word integration as they were new to me, specifically: creating add-ins for Powerpoint (in VBA), creating Firefox extensions, and working with C++.

The goal of this project was to enable users to add citations to PowerPoint with an easy to use interface. Additionally, users can now with a single button click automatically generate slides at the end of the presentation with all the references.

## Keywords

Citations, PowerPoint, Zotero, Add-In, Firefox extension, JavaScript

## Sammanfattning

Tidsskrifter och rapporter behöver nästan alltid inkludera referenser till andra verk genom citat och källhänvisningar. Zotero är ett verktyg som ofta används för att förenkla det arbetet. I nuläget existerar en utökad version av Zotero, (Zotero Word integration extension), som ger möjligheten att integrera referenser i Word-dokument. Däremot finns ännu inte möjligheten att göra detsamma i PowerPoint.

Den största utmaningen med mitt projekt var att sätta mig in i och förstå mjukvaran jag jobbade med (Zotero och Zotero Word integration software). Mjukvarornas storlek gjorde det till en komplex uppgift att förstå de interna strukturerna, samt att identifiera vilka delar som skulle vara till nytta i mitt arbete. En annan utmaning var att jobba med för mig helt nya verktyg, som används av både Zotero och Zotero Word integration. Dessa verktyg var: VBA för skapande av add-ins till Powerpoint, skapandet av Firefox extensions, samt arbete i C++.

Projektets mål var att skapa ett användarvänligt verktyg för att inkludera referenser i PowerPoint. Användaren ska också ha möjligheten att på ett enkelt sätt skapa en sammanfattande sida som innehåller alla källhänvisningar i slutet av Powerpoint-presentationen.

### Nyckelord

Citaten, PowerPoint, Zotero, Add-In, Firefox extension, JavaScript

## Acknowledgments

I would like to thank Professor Gerald Q. Maguire Jr. for giving me ideas and feedback during this project.

Stockholm, August 2015
Ruben Lejeune

## Table of contents

## List of Figures

## List of Tables

## List of acronyms and abbreviations

| | |
|---|---|
| ASCII | American Standard Code for Information Interchange (also known as US-ASCII) |
| dll | dynamic-link library |
| (G)UI | (Graphical) User Interface |
| IDL | Interface Definition Language |
| MFC | Microsoft Foundation Class |
| rtf | Rich Text Format |
| Sub | Subroutine |
| VBA | Visual Basic for Applications |
| XML | Extensible Markup Language |
| XPCOM | Cross Platform Component Object Module |
| XUL | XML User Interface Language |

# 1 Introduction

This chapter provides a brief background to set the context of the work and motivate the need for a tool to help authors to add citations to PowerPoint presentations. Section 1.4 describes the goal of this project and Section 1.5 describes the limitations of this project. The chapter concludes with a summary of the structure of the rest of this report.

## 1.1 Background

Adding appropriate citations when writing scientific papers is both common practice and in many settings a necessity. The citation gives credit to the source and helps to support the author's statement(s). Additionally, a citation helps your readers to find additional information to better understand the matter and it helps them to determine if they find the referenced sources sufficiently reliable [1].

## 1.2 Problem definition

Although adding citations to your document is important, citations are often forgotten or absent in presentations. I hope that by facilitating the process of adding citations to reference when creating a presentation that the number of people whom will use appropriate references in their presentations will increase.

## 1.3 Purpose

The purpose of this report is to describe the process of developing the software artifact that has been created. This description has been written in order to help other developers understand more easily what mistakes have been made, what has been tried and what could be improved. This way they can more easily contribute to this software or suggest improvements that could be made to this software.

## 1.4 Goals

The goal of this project is to enable users to easily add citations to their PowerPoint presentations. This has been divided into the following sub-goals:

1. Add citations to the current slide,
2. Add a slide with all of the references used in the entire presentation to the end of the presentation, and
3. Increase the number of people that add citations to their presentation.

## 1.5 Delimitations

The prototype software artefact that was developed during this project serves as a proof of concept. This means that the software is a basic and rough implementation of the concept and is primarily meant to show developers and users that the concept is useful and realisable. Further development is needed to have a more efficient artefact that could be distributed as a product or as open source software.

## 1.6 Structure of the report

Chapter 2 presents the background information about the technologies that were used during this project. Chapter 3 presents the methodology followed to realize a prototype solution. Chapter 4 gives a detailed description of the process of creating the Zotero PowerPoint software. Chapter 5 states a conclusion about this project and suggests future work.

# 2 Background

This chapter provides background information about some of the technologies that were used during the development of Zotero PowerPoint. As Zotero Word integration is an open source project and has a lot of the functionality I also needed for my project, this software was the main starting point for this project. The Zotero Word integration software can easily be downloaded from their GitHub repository [2].

## 2.1 Firefox extensions

In order to understand the structure and files of the Zotero Word integration extension, I needed some understanding of the basic concepts of Firefox extensions.

> *Extensions add new functionality to Mozilla applications such as Firefox and Thunderbird. They can add anything from a toolbar button to a completely new feature.*

Firefox extensions when downloaded are .xpi files, which are actually compressed folders. To reveal the content, the file extension can be changed to .zip, which allows the file to be unpacked [3]. The typical structure of the content within the xpi file is shown in Figure 2-1.

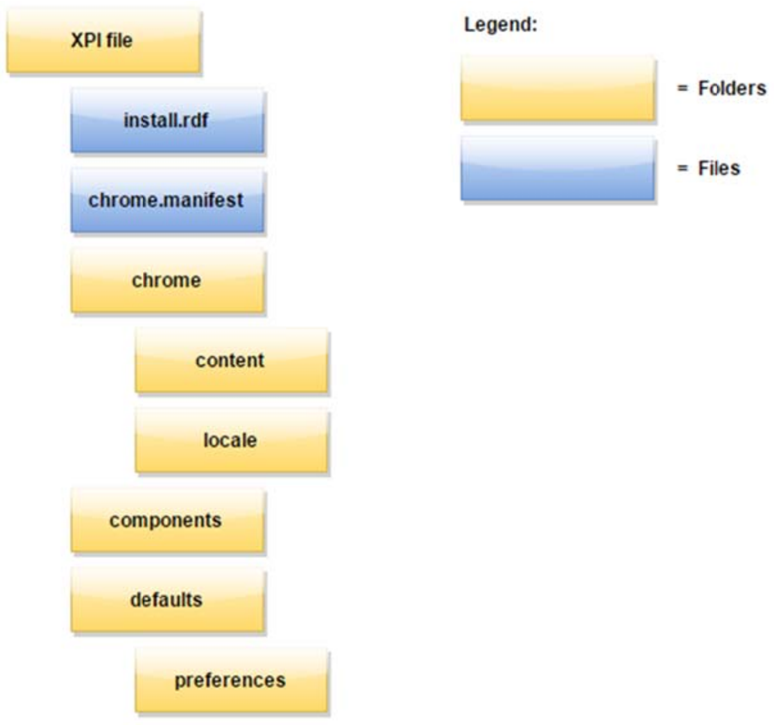


**Figure 2-1 Typical structure within the xpi file**

The included files and folders have the following functionality [4][5]:

| | |
|---|---|
| **Install.rdf** | The installation manifest, written in Extensible Markup Language (XML), provides information about the extension that is being installed (e.g. ID, version). |
| **Chrome.manifest** | Tells Firefox where to look for chrome files. Chrome is the set of user inteface elements of the application window that are outside of a |

| | |
|---|---|
| | windows's content area. Three rules apply when registering content to the chrome manifest:<br>1. Specify the type of the package<br>2. Specify the name of the package<br>3. Specify the location of the package |
| **Content** | Contains the XML User Interface Language (XUL) and JavaScript files of the extension. We use XUL overlays to extend the browser's user interface (UI) by adding or modifying widgets such as buttons, menus, etc. XUL overlays are a way of attaching additional UI Widgets to a XUL document at runtime. |
| **Locale** | Contains the language files. |
| **Components** | Contains the components which will automatically be registered when Firefox runs after the installation of the extension. |
| **Preferences** | Used to store the default preferences. |

## 2.2 The Zotero Word Add-in

The Add-in for Word basically exists out of two parts, the graphical user interface (GUI) and the macros, which are combined in a template file, zotero.dot (found in the install folder within the xpi file).

### 2.2.1 The Graphical User Interface

The GUI provides the buttons that will appear in the Add-in tab. Pressing these buttons activates the macros to initiate the underlying commands.

### *2.2.2* The Macros

When Opening a template, the macros are revealed by pressing ALT + F11.

At the beginning of each macro, a set of functions are declared. These functions are part of a library (e.g. user32) and need to be declared here before they are used;

| | |
|---|---|
| *FindWindow* | Gets the handle of a window outside of the application. A window handle is a unique identifier each window is assigned when it is created in Microsoft's Windows operating system [6] |
| *SendMessage* | Sends a message to a specified window [7] |
| *SetForegroundWindow* | Brings the specified window to the foreground |
| *WideCharToMultibyte* | Maps a wide-character string to a new string [8] |

Whenever a button is clicked a command is executed and the subroutine (sub) that is connected to it will call the *ZoteroCommand* sub with the appropriate arguments, which are the command name and whether to bring the window to the front or not. The different commands included in the Zotero Word integration are included in the table 2-1, found at the Zotero Word integration usage guide. [9]

*Sub stands for subroutine and is similar to a method in programming. However, a sub cannot return a value. If you want a value to be returned in VBA, a function should be used instead of a sub.*

Table 2-1 The Zotero Word integration commands

| **Command** | **Icon** | **Description** |
|---|---|---|
| **Zotero Insert Citation** | | Insert a new citation into your document at the cursor’s location. |
| **Zotero Edit Citation** | | Edit an existing citation. You have to place the cursor inside the relevant citation before pressing this icon. |
| **Zotero Insert Bibliography** | | Insert a bibliography at the cursor’s location. |
| **Zotero Edit Bibliography** | | Edit an existing bibliography. |
| **Zotero Refresh** | | Refresh all citations and the bibliography, updating any item’s metadata that has changed using the information from the relevant Zotero library. |
| **Zotero Set Doc Prefs** | | Open the Document Preferences window. Used to change the citation and/or reference style. |
| **Zotero Remove Codes** | | Remove Zotero field codes from the document. When Zotero inserts a citation or bibliography, it does so by using a field code. This field code allows Zotero to later recognize and automatically update the citation or bibliography (e.g., in numeric styles, citations have often to be renumbered when additional citations are cited). Removing the Zotero field codes prevents any further automatic updates of the citations and bibliographies.<br>Note that removing field codes is **irreversible**, and should usually only be done in the-final copy of your document if it is done at all. |

The *ZoteroCommand* sub tries to find the Zotero or Firefox window (the Zotero window is used for the standalone version of Zotero) by looping through the different possibilities for the window handle. If none of these applications are not found, then an error is shown.

```
ThWnd = FindWindow(appNames(i) & "MessageWindow", vbNullString)
```

Next the pathname to the active document is constructed. The arguments for the SendMessage are modified to have the correct form and finally a message with the corresponding agent, command, and document arguments is sent to the Zotero or Firefox window. An agent is a piece of software that acts on one’s behalf. Here the agent is registered as WinWord and is in this case, as we will see later on, a component that will handle the communication with the Microsoft Libraries on behalf of the Add-on that Zotero Word integration installs in Word.

## 2.3 The Zotero extension

Here the term Zotero extension refers to the main Zotero extension, rather than the Zotero Word integration extension. The reason to mention this extension here is to explain some of the methods and files that will be used later when developing the Zotero PowerPoint software.

As we saw earlier, the macros of the add-in (Section 2.2.2) send a message is to the Firefox or the Zotero-standalone window. The output arguments of the message are shown below:

*firefox.exe –silent–ZoteroIntegrationAgent WinWord –ZoteroIntegrationCommand addCitation –ZoteroIntegrationDocument "C:\Users\Ruben\Desktop\Zotero.dot"*

In order for the command to reach and be processed by the Zotero Firefox extension, something is needed to allow the extension to handle command-line arguments. The Zotero extension adds the following lines to their chrome.manifest file:

```
Component {531828f8-a16c-46be-b9aa-14845c3b010f} components/zotero-service.js
contract @mozilla.org/commandlinehandler/general-startup;1?type=zotero      {531828f8-a16c-46be-b9aa-
14845c3b010f}
category command-line-handler    m-zotero     @mozilla.org/commandlinehandler/general-
startup;1?type=zotero
```

The lines above establish a component from within zotero/components/zotero-service.js as a command-line-handler. Search for this component id in the GitHub repository, we find the handler code, which starts with the following line:

```
function ZoteroCommandLineHandler() {}
```

In this handler, the different arguments are extracted out of the command line string. If the agent is present, then the *Zotero.Integration.execCommand(agent, command, docId)* function is called. This function is located in zotero/chrome/content/zotero/xpcom/integration.js. During the development of Zotero PowerPoint this file proved to be one of the most important files.

The *execCommand* function tries to load the correct Zotero component by first using the agent for the component class, as follows:

```
var componentClass = "@zotero.org/Zotero/integration/application?agent="+agent+";1";
```

If we consider the agent name that was transferred from the macro in this example, we find that this agent name is defined in the Zotero Word integration extension, in the file named build\zoteroWinWordIntegration \zoteroWinWordApplication.h. This file contains the following lines:

```
#define ZOTEROWINWORDAPPLICATION_CONTRACTID
"@zotero.org/Zotero/integration/application?agent=WinWord;1"
```

With this, the correct component (an XPCOM component – see Section 2.4) can be loaded using the following code:

```
var application = Components.classes[componentClass].getService
                    (Components.interfaces.zoteroIntegrationApplication);
```

*The Cross Platform Component Object Module (XPCOM) is a framework which allows developers to break up monolithic software projects into smaller modular pieces. The goal of XPCOM is to allow different pieces of software to be developed and built independently of one another. In order to allow interoperability between components within an application, XPCOM separates the* ***implementation*** *of a component from the* ***interface****.*

*An interface in Mozilla is a definition of a set of functionality that could be implemented by components. Mozilla components implement the code in that does something. Each component implements the functionality as described by their interfaces [10].*

Each object in the components.interfaces represents one of the XPCOM interfaces that can be implemented. These interfaces are created by compiling the file" zoteroIntegration.idl" and they are defined in the header file "zoteroIntegration.h" [11].

The application will be asked to to load the correct document based upon its id, and if this document is not found, then an active document will be selected (using the functions GetDocument and GetActiveDocument in the "zoteroWinWordApplication.cpp") with the following call:

```
document = (application.getDocument && docId ? application.getDocument(docId) :
application.getActiveDocument());
```

As a result of the above call, the method instances of the zoteroWinWordDocument are returned. These zoteroWinWordDocument instances have properties specific to Word (they have been loaded in from the TypeLib MSWORD.OLB).

With these instances the different commands can now be executed. One important command is the addCitation command, which will (as shown in Figure 2-2) activate the interface for the search box based upon the contents of the zotero\chrome\content\zotero\integration\quickFormat.xul file. The code that works together with this XUL overlay is found in the same folder and is called quickFormat.js. This code handles all the events and logic for the overlay. Once the citations have been selected, quickFormat.js fills the array of citation items within integration.js. The quickFormat.xul overlay is shown by calling the displayDialog function after the command has been received at the pptCommand function, as seen in the following code:

```
Zotero.Integration.displayDialog(me._doc, 'chrome://zotero/content/integration/ quickFormat.xul', mode, io);
```

**Figure 2-2 The Zotero search box**

## 2.4 The XPCOM component / build folder

Comparing the GitHub repository and the unpacked xpi file for Zotero Word integration we can see that the files are the same, except for the build folder and the update.rdf which are only found in the GitHub repository. Update.rdf is the update manifest which is pinged by the client software to check if a new version of the code is available.

This build folder contains Interface Definition Language (IDL) files and a folder with C++ code. IDL is an interface definition language. Writing in IDL a programmer describes interfaces in a language- and machine-independent way. These definitions are used to specify interfaces between objects [12]. After these IDL files are compiled they become the xpt files that we can find when we look into the components folder of the Zotero Word integration extension and the header files that are used in the C++ code [13]. These interfaces enable the Firefox extension to communicate with the XPCOM component, which is implemented in C++ code in this folder. Firefox provides an elaborate overview [14] of and guide [15] to XPCOM components.

The build folder is used to generate the zoteroIntegration.xpt, zoteroWinWordIntegrationErrorHandler.xpt, and the zoteroWinWordIntegration.dll files (located in the components-* folders).

In the Zotero Word integration extension, the XPCOM component is used so that a number of Microsoft and Microsoft Office libraries could be used (specifically MSO.DLL, VBE6EXT.OLB, and MSWORD.OLB). These libraries enable the Zotero Word integration extension to get information about the Word document and edit/add text to it.

## 2.5 Summary

The Zotero Word integration software can be split up into three major pieces:

1. The Firefox extension (which in turn installs the two other major components),
2. The Word Add-in, which is the template file with the UI and the macros, and
3. The XPCOM component with its IDL files, which allows the extension to communicate with the required libraries.

# 3 Methodology

This project did not begin with a selection of specific research methodology. However, the research method that was used to develop a software artifact used an empirical method based upon an analysis of the existing Word Intergration extension.

The two most valuable sources of information to realize the Zotero PowerPoint project were the Zotero extension and the Zotero Word integration extension. Small experiments were conducted to observe the function of the existing extension and the code was carefully analyzed in order identify potentially useful pieces of code. This process of identifying code for re-used was very beneficial as these two projects already contained a lot of the functionality I needed.

I evaluated my work by testing the software for bugs and continually updating the software to fix the detected bugs.

My first approach was to modify the Zotero Word integration extension as I thought I could modify this code to work directly from PowerPoint. Therefore during the first phase of development I started out by copying the Zotero Word integration extension file by file and adjusting the contents of these files. This proved to be a bad approach, as there was not enough control of whether the software I was writing worked correctly or not. This occurred because with this method, the software is not gradually developed by expanding its functionality starting with a working base functionality. When making the changes file by file, a lot of the debugging was necessary at the end of the project when the puzzle was supposed to fit together. Shifting all of the debugging to the very end, lead to a lot of time being lost when there are errors. In particular, in several cases I had to start all over again with certain functions.

In the second approach to development I took a more gradual and incremental approach. The Zotero extension and Zotero Word integration extension contained a lot of code, so it was not feasible to try and understand everything all at once. For this reason, I tried to identify the specific parts of the software that I would need to copy and/or change in order to develop Zotero PowerPoint.

In this second approach I first defined which actions I wanted my software to be able to do. If the action that I was trying to implement was already part of one of the existing extensions, then I examined where the specific action that I want to implement in Zotero PowerPoint was initiated and which files were associated with its realization. Once this had been established, I tried to figure out the sequence of functions that were called from the initiation of the action until the end result. Based upon this I started to initiate a similar sequence of function calls in my software, either by copying and adjusting the existing code, or by making use of parts of the existing code. By logging the values of variables and tracing function sequences, I compared these values and sequences to those of the existing extensions. As a result I could immediately see if the results I was getting from my new code were the expected results or not. This allowed me to continuously adjust my software until the expected results were produced.

Whenever I came across technologies that I had not used before or if I wanted something that was not implemented in the extensions, then I looked for tutorials on this topic and then tried to implement this technique in my software.

# 4 Writing the Zotero PowerPoint software

The writing of the Zotero PowerPoint software was split up into two approaches. In the first approach I tried to write the software in a similar fashion as the Zotero Word integration software. However, as was explained in the previous chapter, this did not work out. The second approach was successful. This second approach avoided the use of XPCOM components and an extra Firefox extension.

If as a reader your goal is simply to understand the current code, you can skip Sections 4.1.3 and 4.1.4 as they talk only about code that was not utilized in the final implementation of the prototype.

## 4.1 The first approach

The first steps in this approach were to figure out how to write an Add-In for PowerPoint. To do this, I first created a PowerPoint Macro-Enabled Presentation (Zotero.pptm).

### 4.1.1 The Graphical User Interface

I started to implement the User Interface by creating a UI similar to that of Zotero Word integration. The tool I used for this was the Custom UI Editor for Microsoft Office [16].

I copied the icons, because people are already familiar with them. This was done by making screenshots and stretching them out a bit. To edit the UI, I opened the PowerPoint file within the Custom UI Editor. Using this editor it was easy to edit the XML and to add buttons. A label was added to make the function of each buttons clearer. Finally I added the icons to the buttons to complete the look [17].

An example of the XML associated with a button is the following:

```
<button id="ButtonAddCitation" onAction="ZoteroInsertCitation" showImage="true"
image="addCitation" size="normal" label="Insert Citation" />
```

When a button is pressed the Sub with the same name as defined in the “onAction” attribute is called. These Subs are found in the macros that come with the template. The result is shown in Figure 4-1.

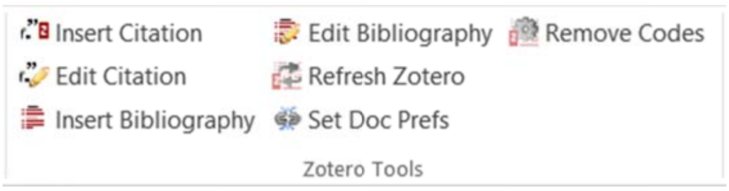
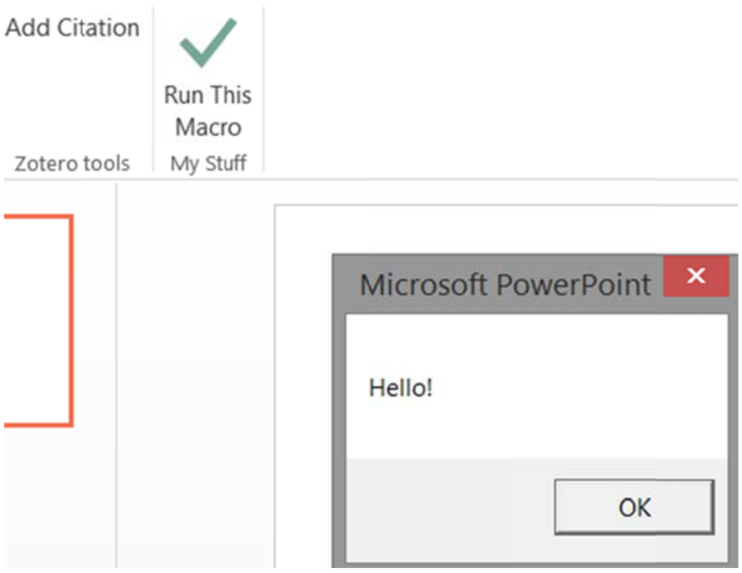


**Figure 4-1 The Zotero PowerPoint GUI**

### 4.1.2 The macros

In order to learn how to write the macros I started with a small test project to see if I could run some code as the result of clicking a button. This simple test project worked, as shown in Figure 4-1.



Microsoft PowerPoint

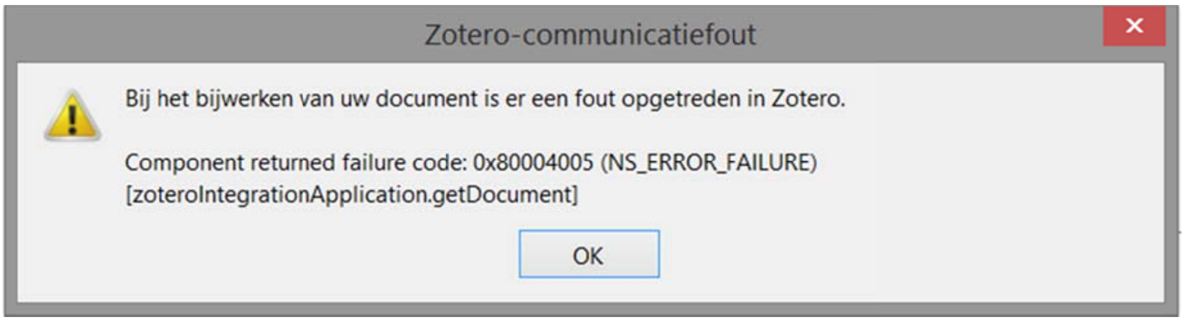
Hello!

OK

**Figure 4-2 The test macro wored**

Next I opnned my Zotero PowerPoint file and pasted the macros from the Zotero Word integration into my macros and made sure they were correctly connected to the buttons. To see what would happen if I invoked a command, I ran the Add Citation command. Because of a problem with the path separator (Application.PathSeparator), which apparently does not work for PowerPoint, I replaced this separator with "/", as this seemed to work (note that "\" did not work correctly as a separator). Invoking the command again, resulted in an error indicating that the ActiveDocument was not defined. Unfortunately, I found that ActiveDocument was specific to Word and that ActivePresentation should be used for PowerPoint [18].

When executing the code after replacing ActiveDocument with ActivePresentation, the Zotero Interface popped up and, and as in Word, I could choose a citation style and select references to add. However, once I had selected a reference to add, this reference was not added to my PowerPoint document, but instead added the reference was added as a citation in a open Word document! I repeated this test with all Word documents closed, then I received the error shown in Figure 4-3.



Zotero-communicatiefout

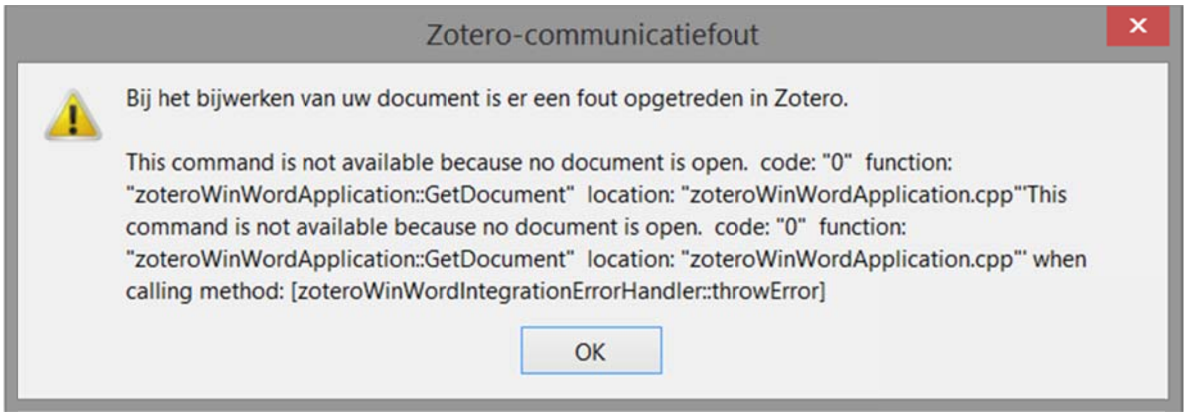
Bij het bijwerken van uw document is er een fout opgetreden in Zotero.

Component returned failure code: 0x80004005 (NS_ERROR_FAILURE) [zoteroIntegrationApplication.getDocument]

OK

**Figure 4-3 The Zotero zoteroIntegrationApplication.getDocument error**

This error message gave me the clue that running Word with the Zotero Word integration extension now automatically activates something called the zoteroIntegrationApplication. If I ran the command again with Word active, but no open document, I received the error shown in Figure 4-4.



Zotero-communicatiefout

Bij het bijwerken van uw document is er een fout opgetreden in Zotero.

This command is not available because no document is open. code: "0" function: "zoteroWinWordApplication::GetDocument" location: "zoteroWinWordApplication.cpp"'This command is not available because no document is open. code: "0" function: "zoteroWinWordApplication::GetDocument" location: "zoteroWinWordApplication.cpp"' when calling method: [zoteroWinWordIntegrationErrorHandler::throwError]

**Figure 4-4 The Zotero ‘no open document’ error**

When analyzing the (Microsoft) Visual Basic for Applications (VBA) code, the only line of code that refers to other installed components was the following:

```
args$ = "-silent -ZoteroIntegrationAgent WinWord -ZoteroIntegrationCommand " & cmd & " -
ZoteroIntegrationDocument """ & name$ & """"
```

This code showed me that I should focus on the extension, as the current macros would suffice for now.

### 4.1.3 Building the Zotero PowerPoint extension

As the Zotero Word integration, which was my main guide (for a while), had its own extension; I started to make an extension for Zotero PowerPoint as well. The first step was to create a very simple extension that installs without any problems. This would serve as the framework to build further upon. To create a Firefox extension, I first used the Mozilla Add-on SDK [19]. However, I felt that I did not have enough control of the extension, thus I shifted to creating an extension in the classical way. To do this, a template for a basic extension was downloaded [20]. I begin by deleting all the example files, except for the chrome.manifest and install.rdf files.

An easy way to set up your project for development is by adding a text file to the extensions directory with the same name as the id in the install.rdf file. The only text this file should contain is the full path to the project directory. Make sure this path ends with a directory separator. On Windows the extensions folder can easily be found by executing Run → %APPDATA% → Mozilla folder → Firefox Folder → Profiles folder → {some id}.default folder → extensions folder.

Once the setup was successful, then the application that the file links to will be installed when you start up Firefox. If the set up was done incorrectly, Firefox will automatically delete the linking file. With this setup you can change your code without having to reinstall your extension all the time. However, a restart of Firefox is required when changes have been made.

With everything in place and thinking that the Zotero Word integration extension only needed modifications in order to work for PowerPoint, I started copying and modifying its files. The first files I wanted to adjust were the essential ones that would give me a working extension that I could install in Firefox as a test. This led me to begin with install.rdf, where I changed only some information about the extension, such as description and author. In the chrome manifest I changed only the following:

```
content     zotero-winpowerpoint-integration chrome/
resource    zotero-winpowerpoint-integration resource/
```

Next I changed the template to install from Zotero.pptm, rather than Zotero.dot to in the installer.jsm file and to change the references from Word to PowerPoint, as in the following lines.

```
var pptm = zpi.getAddonPath(this.EXTENSION_ID);
pptm.append("install");
pptm.append("Zotero.pptm");
```

To test the application as modified thus far, I needed to locate my profile folder → Pressing [Windows key] + [R] and typing %APPDATA% and then OK → locate the Mozilla folder → Locate Firefox/Profiles/<profile_id>.default/ and open the extensions folder in it. In this folder I created a text file named zoteroWinPowerPointIntegration@zotero.org and in the file put the full path to my extension: C:\Users\Ruben\Desktop\Zotero for PowerPoint\Zotero-WinPowerPoint-Plugin-1.0.0

Once I did this, Firefox deleted the file I just created! Only later did I realize that this was probably due to not naming the file the same as the id. Instead I added the file to a zip archive and changed the extension to .xpi. Then I dragged this file to the extensions page, and it installed properly.

Once I changed "WinWord" in the macro to "WinPowerPoint", it gave me the following error: "The needed component for communication with the text editor could not be loaded, make sure the correct extension is installed and try again."

This was a sign that I had to create the XPCOM components for this new extension.

### 4.1.4 Building the XPCOM components

For a long time I was stuck when trying to compile the original Zotero Word integration code. I tried changing the absolute paths, but kept on getting errors about missing tlh files. After a while it turned out that most of these header files were generated with the MFC TypeLib wizard. There were 3 files that generated classes in the project: MSO.DLL, VBE6EXT.OLB, and MSWORD.OLB.

To create the necessary components for Zotero PowerPoint, I created a new C++ project. I began by copying those files that were already in the Zotero Word integration C++ code and then adjusted them file by file to suit PowerPoint. In this process, I learned how a C++ project is structured and how the files work together (i.e., the header files and cpp files).

To figure out exactly where I should start, I checked for those files with the smallest number of includes. For these files, it was easier to see if the project builds correctly. However, it appeared that in the zoteroException file, an include was needed from "zoteroWinWordIntegrationErrorHandler.h", which was a header file produced by one of the IDL filess. Therefore, this file had to be generated first.

The zoteroWinWordIntegrationErrorHandler.idl only needed a change of name (to zoteroWinPowerPointIntegrationErrorHandler) and a change of uuid to make it useable in my build. The uuid was needed to identify my own XPCOM component. This was done in Visual Studio by choosing Tools → Create GUID → (option static const struct GUID) → New GUID and copying the result to the uuid value in the IDL file.

A number of web searches showed that the IDL files could be compiled using midl.exe (located at a path similar to C:\Program Files (x86)\Windows Kits\8.1\bin\x86). However, trying to compile the IDL files gave the error: "cannot find C preprocessor cl.exe". It turned out that C:\Program Files\Microsoft Visual Studio 10.0\VC\vcvarsall.bat had to be run first before compiling. However even after doing so, on every compile there were missing files. When I finally added all needed files, syntax errors started showing up. These were all XPCOM related issues.

I decided to follow another tutorial to see how to write and compile XPCOM components [21]. This tutorial told me I needed to download the Gecko SDK. However, trying to integrate Gecko and

Visual Studio kept on giving errors, so I had to find another way to compile the IDL files to xpt and header files. Eventually I compiled them by adding the C:\Program Files\xulrunner-sdk\sdk\bin; to the Path variables of my computer and ran the compilation with the following commands:

```
C:\Users\Ruben>xpidl -m header -I "C:\Program Files\xulrunner-sdk\idl" -e
"C:\Users\Ruben\Desktop\build\zoteroWinPowerPointIntegration\zoteroWinPowerPointIntegrationErrorHa
ndler.h"
C:\Users\Ruben\Desktop\build\zoteroWinPowerPointIntegration\zoteroWinPowerPointIntegrationErrorHan
dler.idl
```

and

```
C:\Users\Ruben>xpidl -m typelib -I "C:\Program Files\xulrunner-sdk\idl" -e
"C:\Users\Ruben\Desktop\build\zoteroWinPowerPointIntegration\zoteroWinPowerPointIntegrationErrorHa
ndler.xpt"
C:\Users\Ruben\Desktop\build\zoteroWinPowerPointIntegration\zoteroWinPowerPointIntegrationErrorHan
dler.idl
```

I next worked on the zoteroException.cpp and zoteroException.h files. Copying the header did not give any errors. However, for the cpp file there were some files missing, specifically: nsCOMPtr.h and nsServiceManagerUtils.h.

After getting many errors and not finding the right includes, I realized that I had downloaded the xulrunner version for Linux.

While I got the xulrunner version for Windows working, the project still showed a lot of errors. One solution to resolve some of the errors I was getting was due to an undefined char16_t. In order to solved this I needed to add #include <mozilla/Char16.h> to the stdafx.h file. In the preprocessor (located in Project Property Pages -> C/C++) I had to add _CHAR16T to the Preprocessor definitions.

The next errors I ran into were mismatch and linking errors. There was a mismatch between the _ITERATOR_DEBUG_LEVEL, so I had to change the Runtime Library from Multi-threaded Debug to Multi-Threaded. I set up the linking in exactly the same as in the Word version, however in xulrunner there must have been some changes since the time that the Word version had been written. Finally, I resolved the linking errors by linking with nss3.lib and thus I got the compilation to work.

For the zoteroIntegration.idl I changed only the uuid's of the components and repeated the same compilation process. The generated header file contained templates for the other header files. I created the new header files and copied the generated templates into them. I had to replace _MYCLASS_ with the correct class name.

When comparing the folder structure I realized that I was working with the wrong project template. To test this idea, I created a win32 dynamic-link library (dll) file. After testing the different options, the zotero for word looked more like a Microsoft Foundation Class (MFC) dll file. For my next attempt I chose to create a regular dll with MFC statically linked. I again copied the generated header files from the IDLs to the new project's header files and copied the zoteroException file as it does not seem to have Microsoft Word specific items. The build succeeded.

The next header file to create was the zoteroWinPowerPointApplication.h. I this case I only had to change the names and ids. Again the build succeeded.

Next I added the zoteroWinPowerPointDocument, which was generated from the modified version of ZoteroWordIntegration. At this time, I had not yet realized that this was not going to work for PowerPoint. I changed the class names and methods and had to import a few MFC classes with the TypeLib. To add or check which classes are available in the files go to Project -> Class

Wizard and choose MFC Class from TypeLib instead of Add Class. Browse to the correct files. The path should be something similar to:

C:\Program Files (x86)\Common Files\Microsoft Shared\OFFICE15\MSO.DLL
C:\Program Files (x86)\Common Files\Microsoft Shared\VBA\VBA6\VBE6EXT.OLB
C:\Program Files (x86)\Microsoft Office\Office15\MSWORD.OLB

I realized after adding MSPPT.OLB that the fields and document header files were not available for PowerPoint, which meant that I could not easily adapt the extension to work for PowerPoint (as well as Word). This also meant that I would have to rewrite the IDL files and change most of the extension so that it would work for PowerPoint. The Zotero integration file would not be useful, as it works with documents and fields, which meant their integration was basically written only for word processors and could not easily be adapted to work with other software.

Once I realized this, the first step was to see how much I could do if I presented my presentation file as a document to the Zotero integration file. It appeared that even though I might be able to treat my presentation as a document during the first steps, I would still run into problems before being able to execute any command. This occurs because Fields within the document are being used in the Word version, but there is no comparable equivalent in PowerPoint.

Based upon the above analysis this suggested that there were two different solutions:

- Figure out all the different classes and methods I would need for communication between the Integration document and the PowerPoint extension. Create an IDL file to support this. Implement the generated templates. Create the installation files for the extension. Try to access the interfaces used for Zotero Word for Windows integration (e.g. via the add citation interface) and retrieve the information, although this is information was also written specific for Word documents. Add new methods in the integration.js file to process this information for PowerPoint.
- Skip XPCOM by directly expand the Add-ins macro and the integration.js file. As a result there would not be any need to develop an extension, as I could integrate most of the code into the main Zotero software by editing the integration file. In this approach I would use JavaScript to lookup the correct citation, format it, and return it to the macro. Within the macro I would add the citation(s) to the presentation, add slides, etc. In this approach I do not need to rewrite the whole Word extension and would not have to struggle with the XPCOM components.

As the first solution is very complicated and would take a very long time to develop, I decided to explore the second solution.

## 4.2 The second approach lead to the prototype

The following items need to be done to realize this second solution:

1. Enable the user to choose a citation style,
2. Add extra functionality to the Zotero integration file and try to activate the add citation dialog (which normally requires a document file),
3. Get the information back from the add citation dialogue, sending it back either directly to PowerPoint or store it in a temporary file,
4. Read the response and add the citation to the currently active slide, and
5. On activation of another command invoked by clicking an appropriate icon, collect all citations and add them to the references slide(s).

I tried to log data from the zotero-service.js, which handles the command sent from VBA, but whenever I use alert (“some text”) or console.log (“some text”) there, the file would not execute anymore. After a while I realized that it was not really necessary to debug this particular file, as the only functionality was to catch a command and call a function from the Integration.js file. Therefore, I changed the arguments passed the PowerPoint macro's sendMessage function to:

```
args$ = "-silent -ZoteroPowerPoint WinPowerPoint "
```

In the zotero-service.js file I added the following to handle commands with the ZoteroPowerPoint flag:

```
var ppt = cmdLine.handleFlagWithParam("ZoteroPowerPoint", false);
if (ppt) {
    var Zotero = this.Zotero;
    Zotero.setTimeout(function() { Zotero.Integration.pptCommand(ppt) }, 0);
}
```

At first I used the handleFlagWithParam in this if structure, but then I ran into trouble when trying to use it again. I realized that it was important to first give the variable ppt its value first, as the flag and parameter seemed to be removed after using the handleFlagWithParam. In the Integration.js file I wrote a simple function that generates alerts when it receives the message. Later I used this same pptCommand function to initiate the add citation command.

```
this.pptCommand = new function() {
    var inProgress;
    return function pptCommand(ppt) {
        alert("message ppt received");
    }
}
```

This worked, which meant that the command was successfully being sent and handled. It turned out that alert worked here, but console.log commands did not. Next I wanted to display the dialog for adding citations. In the functions used to do this for the Word integration, Word specific values are being used. In order to execute these functions, I needed to make objects that acted as if they were the Word specific values.

In order to display the search box, I started by examining the DisplayDialog function, as this is the general function that can call all sorts of dialog windows. In order to do so, there were two variables that I had to simulate, as the doc and io values are Word specific. This function call is as follows:

```
this.displayDialog = function displayDialog(doc, url, options, io) {
```

When calling this method, first doc.cleanup(); is called, so I created a new function pptDoc() which had a cleanup() function that did not do anything. This seemed to be the only time the doc argument was used. As a test I used null for the io, as io ? io : null was used later in their code, so I thought this variable might be optional. When I execute this, the dialog window opened up, however was not actually working (for example, I was not able to switch to classic mode, create a citation list, or close the dialog). Inside the addCitationDialog I found the following code:

```
io = window.arguments[0].wrappedJSObject;
```

This meant I had to simulate io as well. The variable io is assigned its value in the integration.js file with the following code:

```
var io = new Zotero.Integration.CitationEditInterface(citation, field, me, session);
```

I created my own io function, pptIo(). To create this function I researched what the minimum requirements were for io to make the search function work. To do so I needed to trace what happens with io after it is passed to the XUL file. As can be seen in the code below, it seemed that io was being passed to the newly opened window.

```
Var window = Components.classes["@mozilla.org/embedcomp/window-
watcher;1"].getService(Components.interfaces.nsIWindowWatcher).openWindow(null, url, '', allOptions, (io ?
io : null));
```

An essential task was to get the list of citations when a user types into the search box. For some reason I was convinced that AddcitationDialog.xul was being used for the search box and analyzed this file for a while. However, after some testing with a couple of added alerts, it appeared that quickFormat.xul was actually being used. The addCitationDialog function was for the classical view. So I created a PowerPoint version of the quickFormat.xul and quickFormat.js (quickFormatPpt.xul and quickFormatPpt.js). In the load function I commented out everything using the argument io, as they did not seem important for the first stages of the search box.

To see what the different layout functions were I started commenting out different elements and see what the effect was. When I stripped out all the essentials, it seemed as if I still could not communicate with the event handlers located in quickFormatPpt.js. After a while I realized that the quickFormatPpt.xul was still linked to the old quickFormat.js file. Once I tested the onKeyPress event with the correct include, I was able to get a response.

As the comments in the quickFormat.js file were not clear to me, which caused me trouble when trying to understand what each function did, I put an alert at the start of each function with the name of the function. This way I could follow the actions of the Word version for the functionality that I wanted to put in the PowerPoint version. These alerts enabled me to see which functions were used, when they were used, and why.

The moment I typed something into the textbox, events started to be generated. After typing two letters, a list of possible references showed up. With the alerts at every function in place, the following sequence was found. On typing the first letter the following functions were called:

_onQuickSearchKeyPress → _resetSearchTimer → onKeyPress → _quickFormat → _getEditorContent → _getCurrentEditorTextNode → _updateItemList → _updateCitationObject → _resize

From the second letter on the following functions were called for each leter:

_onQuickSearchKeyPress → _resetSearchTimer → onKeyPress → _quickFormat → _getEditorContent → _getCurrentEditorTextNode → _updateLocator → _getItems → _updateItemList → _updateCitationObject → _buildListSeparator → _buildListItem → _buildItemDescription → _buildListItem → _buildItemDescription → _buildListItem → _buildItemDescription → _buildListItem → _buildItemDescription → _buildListItem → _buildItemDescription → _resize → _openReferencePanel → _updateItemList → (here every list item disappeared again) _buildListSeparator → _updateCitationObject → _buildListSeparator → _buildListItem → _buildItemDescription → (now items started filling the list up one by one again)_buildListItem → _buildItemDescription → this cycle repeated until all elements are in list) → _resize

After _resize the alerts stopped and the list of references was shown. It appeared that all the special keypresses are handled in _onQuickSearchKeyPress, such as enter, backspace, tabs, etc. If the key was simply an alphabetic letter, the _resetSearchTimer method was activated. I did not have to change much here as far as I could see.

I put alerts in the PowerPoint version of _onQuickSearchKeyPress and _resetSearchTimer. The _ resetSearchTimer did not seem to have any calls to other functions or functionality that I would not use, so I did not change anything in this function. Both alerts for these functions showed up when I started entering letters into the textbox.

Next in the sequence the onKeyPress event occurred, however this function simply tests if it is acceptable to exit the window when the escape key is entered. This event was generated every time a key was pressed and was unimportant in the process of showing the list of potential references. However, for some reason, my Zotero window did not want to close on escape, so I added a call to window.close(); when the escape key was pressed. This properly closed the Zotero search bar.

Next I put an alert into the following functions: _quickFormat, _getEditorContent, _getCurrentEditorTextNode , _updateItemsList, _updateCitationObject, and _resize. The expected alerts showed up until _updateItemsList, thus something must have gone wrong within this function. To explore this further, I inserted alerts that output a specific number, spread throughout this method to see how far I got within this function. In this way I was able to pinpoint the line where things went wrong:

```
while(referenceBox.hasChildNodes()) referenceBox.removeChild(referenceBox.firstChild);
```

When I tested whether or not the referenceBox had child nodes, there was no response. So this function was not working properly. When checking what it was initialized with, I found the following line of code:

```
referenceBox = document.getElementById("quick-format-reference-list");
```

I suspected that this was one of the parts of the quickFormatPpt.xul that I commented out, as at that point I was not yet sure of its functionality. Examining this xul file, the element with this id was indeed commented out. It was a panel with the “richlistbox” of citations that I wanted to show.

At this point I decided to uncomment all of the parts that I had commented out in the quickFormatPpt.xul file to see how far I was getting in the sequence. It seemed that for the first letter I got up to _updateCitationObject. Here the citation object is populated, and as I expected, it was a function where io was being used. This was the reason things had gone wrong in this function, as I did not yet have a suitable value for this variable.

As I now needed a suitable value for the variable io, I started to expand my pptIo function, by adding properties when they were needed by the sequence of events. It seemed that io needed a citation element, an array of citationItems, and an array of properties. Within these properties, a Boolean “unsorted” should be added. I gave the unsorted variable the value ‘true’. Later in the method io.sortable is checked, so I added this Boolean to the io elements and gave it a value of false as I did not have any sorting implemented. These changes resulted in the following code:

```
this.sortable = false; // Determines if citation is sortable in current style
this.citation = {"citationItems":[], "properties":{"unsorted":true}};
this.citationsByItemID = {};
```

With these variables in place, I could uncomment some of the code in the load functions of quickFormatPpt.js, especially the initialization of io, which initially I had forgotten to uncomment. To see if the io variable was properly transferred, I tested whether I could generate an alert for the unsorted Boolean within _onQuickSearchKeyPress. This did not work. When I checked if the code after if(io) was executed, the subsequent alert did not appear. So there was something wrong with the passing of my io variable to the quickFormat window.

When testing if there was an argument that came from the dialog window, the test returned true. It was only when the .wrappedJSObject was added, that it did not work anymore. This occurs in the following line:

```
io = window.arguments[0].wrappedJSObject;
```

I must have missed something in io that enabled it to pass the variable as a JSON object. To find exactly what was missing, I examined how the Word version constructed the io variable. It turned out that they initialized io with the following code:

```
var io = new Zotero.Integration.CitationEditInterface(citation, field, me, session);
```

In the function that was called, I saw that they added this.wrappedJSObject = this; to make it work across window boundaries. When testing, I could successfully access the unsorted Boolean within io.

Continuing to follow the sequence, when nodes[i].citationItem was output in the _buildItemDescription function it showed that this value was “undefined”, but this time the _resize function was reached. To see if this value was normally undefined on the first letter, I put the same alert into the Word version. The Word version also that this value was undefined as well, so it seemed as if this was the normal behavior on the first letter. This seems logical as the item list was not yet populated.

Now I could move on to the sequence of event that occur when the second letter is typed. Here I got to the _updateLocator event, which returned a string value to its caller (i.e., _quickFormat). Again, there was a new use of io, which as it did not have the correct value caused a discontinuation of the sequence. It seemed that I needed to implement a io.getItems()function, which, according to the comments in the Word version, gets a list of items used in the current document. As we do not actually have a document, I simply returned an empty array. This seemed to be the only function for io needed in the _quickFormat function.

Unfortunately, it turned out that simply returning an empty array did not work. This revealed that something more complex was going on here. When I checked the getItems function, the return structure was:

```
return this._updateSession().then(function() {
    return me._getItems();
}, function() {
    return [];
});
```

This was a JavaScript promise. Basically this says, if _updateSession was a success, go into the function where the value of me._getItems() is returned, if there was an error, then return an empty array. I created a promise, as it seemed to be needed to correctly return the values to the _quickFormat() function. However, I made the function that was called for the promise always return Q.resolve(true).

```
this._updateSession = function updateSession() {
    return Q.resolve(true);
}
```

I also returned an empty array in getItems, no matter what the promise returned, using the following code:

```
this.getItems = function getItems() {
            return this._updateSession().then(function() {
                return [];
            }, function() {
                return [];
            });
        }
```

When I tested this, I was finally able to show a list of the citations in the search box. The sequences for the first and second letter were exactly the same as for the Word version.

The following step is the selection of an item from the list of citations. Here I used the same method, I clicked on an item in the Word version which produced the following sequence:

_bubbleizeSelected → _getEditorContent → _getCurrentEditorTextNode → _updateLocator → _getCurrentEditorTextNode → _insertBubble → _buildBubbleString → _clearEntryList → _resize → _refocusQfe → _previewAndSort → _refocusQfe

I now tested how far the PowerPoint version got without running into trouble. It appeared that this ran without any problems, so I did not have to change anything. After this I examined what happened when you accept the citation you are adding (by pressing enter). This lead to the following sequence:

_onQuickSearchKeyPress → _bubbleizeSelected → _getEditorContent → _getCurrentEditorTextNode → _accept → _updateCitationObject → _onProgress

The PowerPoint version worked up until _updateCitationObject, where the function io.accept(_onProgress) is called. I copied the functionality of the accept function and added the this.acceptDeferred value to pptIo. By doing this, the whole sequence got executed and the progress bar was filled. This new functionality was implemented by the following code:

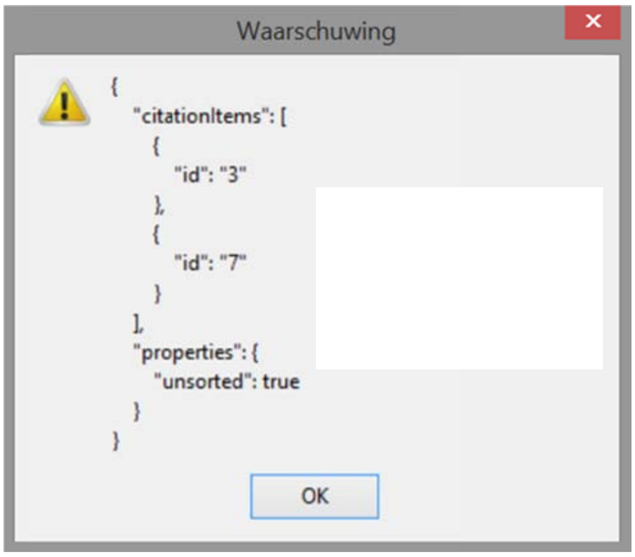
```
this.accept = function accept(progressCallback) {
    if (!this._acceptDeferred.promise.isFulfilled()) {
        this._acceptDeferred.resolve(progressCallback);
    }
}
```

Even though this worked, I still did not know how I could get the citation string, as I did not see any other functions were called. Therefore, I checked if there were any handlers for when _acceptDeferred was fulfilled. This seemed to occur in the Zotero.Integration.CitationEditInterface function. The most important functions that are executed when this happens are the following:

```
me._session.addCitation(me._fieldIndex, me._field.getNoteIndex(), me.citation);
me._session.updateIndices[me._fieldIndex] = true;
```

To get the formatted citation, I needed to know how to output objects. As log was not working, I had to find a way to convert the object into a string and then output it with alert. I found that you can use JSON.stringify(obj,null,4); to get an object as a string with indentation [22].

Figure 4-5 shows the output of the citation after I selected multiple citations in the search box. When outputting various possibilities that could contain the full citation data, I had a hard time pinning down exactly where the citation data was transmitted. After a lot of searching, I did not succeed in finding the exact place where the full citation data is stored into a variable and where it is formatted into the correct bibliographic style. To track this down I put alerts everywhere and then tried to follow the progress step by step.



```
{
    "citationItems": [
        {
            "id": "3"
        },
        {
            "id": "7"
        }
    ],
    "properties": {
        "unsorted": true
    }
}
```

**Figure 4-5** **The structure of the citation variable**

Even though I tried to generate an alert within each called function and output a number of different variables, I could not find the formatted citation. So I started recreating the steps the Word version took and compared the values at each step.

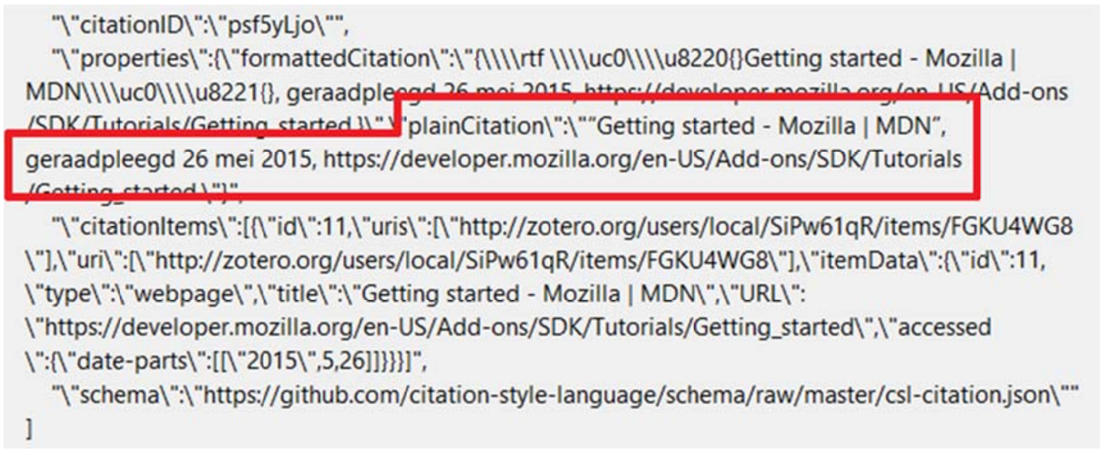
The citation variable before it was sent to the me._session.addCitation(me._fieldIndex, me._field.getNoteIndex(), me.citation) function had the value:

```
{
  "citationItems": [
    {
      "id": "9"
    }
  ],
  "properties": {}
}
```

This was also the same information that I received when outputting citation.citationItems after _updateCitationObject when quickFormatPpt was executed. Next I tried to find out how to format it in the correct bibliographic style. As I was not sure whether the Word version did this within JavaScript or within C++, I searched for the last functions that were executed before I saw the citation appear in the Word document. The sequence of function calls is:

addCitation → lookupItems → updateDocument → updateUpdateIndices → updateCitations → formatCitation → _getPrePost → formatCitation → _updateDocument → getCitationField → getURIsForItemID

I considered which variable I should output in order to check if there was a string approrpriate to the citation style. Inside the getCitationField function I output the field variable and saw the output shown in Figure 4-6.



```
   "\"citationID\":\"psf5yLjo\"",
   "\"properties\":{\"formattedCitation\":\"{\\\\rtf \\\\uc0\\\\u8220{}Getting started - Mozilla |
MDN\\\\uc0\\\\u8221{}, geraadpleegd 26 mei 2015, https://developer.mozilla.org/en-US/Add-ons
/SDK/Tutorials/Getting_started.}\",\"plainCitation\":\"“Getting started - Mozilla | MDN”,
geraadpleegd 26 mei 2015, https://developer.mozilla.org/en-US/Add-ons/SDK/Tutorials
/Getting_started.\"}",
   "\"citationItems\":[{\"id\":11,\"uris\":[\"http://zotero.org/users/local/SiPw61qR/items/FGKU4WG8
\"],\"uri\":[\"http://zotero.org/users/local/SiPw61qR/items/FGKU4WG8\"],\"itemData\":{\"id\":11,
\"type\":\"webpage\",\"title\":\"Getting started - Mozilla | MDN\",\"URL\":
\"https://developer.mozilla.org/en-US/Add-ons/SDK/Tutorials/Getting_started\",\"accessed
\":{\"date-parts\":[[\"2015\",5,26]]}}}]",
   "\"schema\":\"https://github.com/citation-style-language/schema/raw/master/csl-citation.json\""
]
```

**Figure 4-6 the output of the field variable**

From this figure we can see that the variable contained a string called plainCitation which contains the citation in the correct style. The next step was tracing backwards to determine where plainCitation received its value. In the getCitationField function this field is constructed. I expected that properties, a variable in this function, contained plainCitation. When outputting properties it indeed contained plainCitation, as seems reasonable given the code below:

```
var properties = JSON.stringify(citation.properties, saveProperties);
```

Citation was a variable passed to the getCitationField function, which was called from _updateDocument. In _updateDocument I found the following code:

```
field.setText(formattedCitation, isRich);
citation.properties.plainCitation = field.getText();
```

These are calls to C++ methods in the Word version.

Earlier in the function, formattedCitation gets its value from this._session.citationText[i]. I continued exploring this path to find exactly where citationText was changed. In the previous function, which is formatCitation, I saw that the following line of code (which was the code I was looking for):

```
var newCitations = this.style.processCitationCluster(citation, citationsPre, citationsPost);
```

When I outputted the citation variable that is passed to this function I saw the following:

```
{
  "citationItems": [
    {
      "id": 11
    }
  ],
  "properties": {
    "added": true,
    "zoteroIndex": 0,
    "noteIndex": 1
  },
  "citationID": "b3xd5eFw"
}
```

CitationPre and CitationPost were both empty arrays. I needed to find how to initialize this.style so I could use it. As this code was in Zotero.Integration.Session.prototype.formatCitation, the class this.style should be declared somewhere in Zotero.Integration.Session. I found it in the function Zotero.Integration.Session.prototype.setData, as shown in the following code:

```
var getStyle = Zotero.Styles.get(data.style.styleID);
data.style.hasBibliography = getStyle.hasBibliography;
this.style = getStyle.getCiteProc(data.prefs.automaticJournalAbbreviations);
this.style.setOutputFormat("rtf");
this.styleClass = getStyle.class;
```

The Rich Text Format (rtf) format is a Microsoft proprietary format. However, many programs can read and write files in this format.

Outputting the data variable at the beginning of this function showed that data had the value:

```
{
  "style": {
    "styleID": "http://www.zotero.org/styles/chicago-fullnote-bibliography",
    "hasBibliography": true,
    "bibliographyStyleHasBeenSet": false
  },
  "prefs": {
    "fieldType": "Field",
    "storeReferences": true,
    "automaticJournalAbbreviations": true,
    "noteType": "1"
  },
  "sessionID": "FSemvaa4",
  "zoteroVersion": "4.0.26.4",
  "dataVersion": "3"
}
```

Next I searched to find where the data variable was set, as it contained important values to initialize this.style. I started by looking for the call to the setData function. This function was used in the _getSession function and data was initialized by the following code:

```
 data = new Zotero.Integration.DocumentData();.
```

In this code three values were assigned using the following code:

```
this.style = {};
this.prefs = {};
this.sessionID = null;
if (string) {
      this.unserialize(string);
}
```

But these values only get a real value when a string is passed to the DocumentData function. In this case the unserializeXML function is called which gives the variables their true values. This string comes from this._doc.getDocumentData(), which is one of the C++ methods linked to the Word specific document.

To test if I could get the string in the correct citation style with static variables for data and citation, I copied the output for data and used this to try to make this.style to work by adding the following code:

```
var testCitation = {
      "citationItems": [{"id": 11}],
      "properties": {"added": true,
      "zoteroIndex": 0,
      "noteIndex": 1},
      "citationID": "b3xd5eFw"
};
```

```
var newCitations = me.style.processCitationCluster(testCitation, [], []);
```

With the above values, the call succeeded in outputting the following:

```
 [{"bibchange":false,"citation_errors":[]},[[0,"\\uc0\\u8220{}Getting started - Mozilla | MDN\\uc0\\u8221{}, geraadpleegd 26 mei 2015, https://developer.mozilla.org/en-US/Add-ons/SDK/Tutorials/Getting_started."]]]
```

The string is in correct citation style, but not cleaned up yet. In order to test this, I copied this output into a text file and gave it the extension “rtf”.

```
{\rtf1\ansi\deff0
\uc0\u8220{}Getting started - Mozilla | MDN\uc0\u8221{}, geraadpleegd 26 mei 2015,
https://developer.mozilla.org/en-US/Add-ons/SDK/Tutorials/Getting_started.}
```

As expected this showed the correct citation when opened in Word, \uc0\u8220{ and \uc0\u8221{ were the codes for the quotes. I replaced these by an escaped quote (\”) in JavaScript.

For the Zotero for PowerPoint version the user should be able to choose his or her preferred citation style. As the main aim for this project was to add citations and a summary references slide, I decided not to implement all of the functionality that the Word version had. Adding citations, choosing citation style, and adding the references slide were the only UI buttons that I focused on.

To see the available citation styles, I output the available style names and ids with the following code:

```
var styles = Zotero.Styles.getVisible();
var styleList = {"titles":[], "ids":[]};
      for each(var style in styles) {
      styleList.titles.push(style.title);
      styleList.ids.push(style.styleID);
};
alert(JSON.stringify(styleList, null, 4));
```

In this way I could easily copy the name and id of these styles into my VBA script. I made a new form in VBA with just a ListBox for the citation style and a Save/Cancel option for the user to set their preferred citation style.

Upon loading of the form, I initialize two arrays: an array with the name of each of the styles and an array with the ids of the styles. The reason to use 2 arrays instead of one 2D array is that the ListBox was easier to populate this way. It is also easy to request the selectedIndex from the ListBox and thus this index can be used with the other array to retrieve the id (as I arranged them in the corresponding order).

Saving the preferences proved to be an easy task. To save and retrieve the style preferences in the registry, the following methods were used [23]:

```
SaveSetting "ZoteroPowerPoint", "PPTPreferences", "CitationStyle", styleIDs(ListBoxCitationStyles.ListIndex)
GetCitationStyle = GetSetting("ZoteroPowerPoint", "PPTPreferences", "CitationStyle",
"http://www.zotero.org/styles/chicago-fullnote-bibliography")
```

As a default I chose the “Chicago Full Note” style. I added the citation style as an argument to be sent from the VBA script to JavaScript as follows:

```
args$ = "-silent -ZoteroPowerPoint WinPowerPoint -ZoteroCitationStyle " & citationStyle
```

In order to be able to process this in zotero-service.js, I had to extract the citation style from the command line before it is sent to the pptCommand function using the following code:

```
var citationStyle = cmdLine.handleFlagWithParam("ZoteroCitationStyle", false);
```

In JavaScript the citationStyle variable is used when the pptData is initialized. In this initialization the styleID is set to the value of citationStyle. This enables users to use different citation styles. When testing, the “IEEE” and “Vancouver” style output a citation as (respectively): [1] and (1), but would not increment the number when adding additional citations. The “Cell” style gave an error ([CSL STYLE ERROR: reference with no printed form.]). The “Nature” style gave \super 1\nosupersub{} and “Modern Humanities Research Association 3rd edition” had some rtf codes in it. Therefore, I decided to return to these issues in the future.

The next major task was to send the citation string back to PowerPoint. I considered the following methods:

- Use the sendMessage Windows API function to send the data back to the VBA script or
- Write the JavaScript string to a text file and poll for the file in VBA.

The first method was not well documented and I could not find how to communicate with the VBA script when using this method. The second method might be less elegant one, but was selected due to the lack of a better solution.

First I needed to learn how to create a temporary file file [24] and then I wrote the citation to this file. Next I inserted a polling loop into the VBA to read the citation(s) from the temporary text file. To do so, I found a promising method in an Excel tutorial that used Application.OnTime. However, it seemed that this method was not defined for PowerPoint, thus I needed to find another method. I found that a timer that could be started, stopped, and hence I could execute code after a chosen amount of time. In the FileThere sub I used this timer to check every second if the temporary file exists or not, the timer starts from the moment the user clicked on the add citation button. This is done in a sub called PollForDocument [25].

Once the temporary file had been found, I stopped the timer. Now the citation had to be read. To do this I used the insight gained by reading [26] to add the following VBA code:

```
Open myFile For Input As #1
Dim text As String
While EOF(1) = False
    Line Input #1, textline
    text = text & textline
Wend
Close #1
Kill myFile
```

To prevent the file from being read multiple times, I added a Boolean variable: available. Whenever the code where the text file was being read was entered, the variable available becomes false. These his code is no longer executed until the variable becomes true again. Whenever a new command is executed, the timer is stopped and the variable available is again set to true. When the citation has been read, I delete the temp file and call the addCitation(text) sub. At first I worked with an absolute path to the temp file, but had to change this so that it would work on other computers as well. It turned out that this could easily be done by just calling Environ("Temp"), which returns a string with the path to the temp folder [27]. This is done as follows:

```
myFile = Environ("Temp") & "\zoteroCitation.tmp"
```

To format the citations properly, the straight quote symbols had to be replaced with the left and right quotes. So I took a step back and removed the cleaning up from JavaScript and moved it to VBA. In the VBA code I replaced the string ANSI_X3.4-1968 with the proper symbols using the American Standard Code for Information Interchange (ASCII , also known as US-ASCII and standardized in ) code for each symbol, as follows [28]:

```
citation = Replace(citation, "\uc0\u8220{}", Chr(147))
citation = Replace(citation, "\uc0\u8221{}", Chr(148))
```

At first, when adding a citation, I added a textbox that I called citationTextBox to the currently active slide and added the citation text there using the following code:

```
Set pptSlide = ActivePresentation.Slides(ActiveWindow.View.Slide.SlideIndex)
pptSlide.Shapes.AddTextbox(msoTextOrientationHorizontal, (pptLayout.Width * 0.125), (pptLayout.Height -
100), 30, 50).name = “referenceBox”
 pptSlide.Shapes(“referenceBox”).Tags.Add "Text", citationText
```

Next I wanted to make the sortable styles work. Up until now I simply got a string back with the number (e.g. [1]). Now I needed to get the citation as well. I found the string I was looking for in the bib variable in the getBibliography function in integration.js. However, the item’s in style had to be updated first [29]. This was done by:

```
var my_ids = [testCitation.citationItems[0].id];
me.style.updateItems( my_ids, true );
var bib = me.style.makeBibliography();
```

For the summary of references slide(s), I simply looped over all the slides looking for shapes with the name “referenceBox” and added the text from these to one long string with newlines between the citations. After generating a new slide at the end of the presentation [30], I simply insert the text from the long string into the slide, as follows:

```
Set pptSlide = ActivePresentation.Slides.AddSlide((ActivePresentation.Slides.Count + 1), pptLayout)
slideName = "References" & slideCountRef
pptSlide.name = slideName
pptSlide.Shapes.Title.TextFrame.TextRange.text = "References"
```

Up to this point everything was working with a hardcoded test citation. Next I went back to the integration.js file and realized that I only had to change the id of the citationItems to the correct id and then the style.processCitationCluster function would return the correct citation. The other properties were used for the Word version, but for PowerPoint their value did not really matter. With this change, I could now add all the saved citations as follows:

```
testCitation.citationItems[0].id = me.citation.citationItems[i].id;
```

Up until now it was only possible to add a single citation to the slide, but I wanted to be able to add multiple citations in different textboxes to a single slide. To avoid name conflicts, but still be able to loop through the citation boxes for the final reference slide, I gave each of the textboxes the name “citation” & index. However, I had a problem of knowing the correct index. To learn this I had to loop over every shape on the slide, to see if the name contained the string “citation” and get the highest number that follows “citation” from these instances when I want to add a new citation. Another problem was that if any citation is deleted or rearranged, then the indices would be messed up. This would be a problem for sortable styles (e.g. IEEE).

To solve both the highest index problem and the problem with ‘holes’ or incorrect order in the citation numbering, I added an updateIndices function that returns a Long (integer) value. This function has a value “i” and loops over every slide and every shape within this slide. If the name of the shape contains the string “citation” (which would be one of the citation textboxes we added), the name of the shape is updated to “citation” followed by the number “i” and then “i” is incremented. I also added the index of the citation to a tag of the textbox as follows:

```
pptSlide.Shapes(textBoxName).Tags.Add "CitationID", index
```

I thought that this would give the correct names and order, but it did not. When looping over all of the shapes within a slide, VBA did not go through the shapes in the order they were visible in (as I assumed that they were kept in top to bottom and left to right order). The shapes which were added the earliest, were the first ones found, no matter where the shape was placed on the slide.

As I needed to order the citations in order of their position in normal reading order, I had to rethink my strategy. Instead of immediately renaming all the shapes, I added all their names to a string called citationCollection as follows:

```
For Each shp In sld.Shapes
    If (InStr(shp.name, "citation") <> 0) Then
        If (i = 1) Then
            citationCollection = shp.name
        Else
            citationCollection = citationCollection & "," & shp.name
        End If
      i = i + 1
    End If
Next shp
```

Once I got all the shape names, I split the string up to create an array. This array allowed me to use a bubble sorting algorithm, which sorted the citations based upon the distance from the top of the slide and distance from the left side of the slide. Once the array was sorted, I looped over every value and changed the value of the CitationID tag to the index in this array.

Now that all of the citations have received their correct CitationID, I call a sub named UpdateTextBoxes. UpdateTextBoxes loops over every shape in the presentation and updates the name of the textbox to “citation” + CitationID using the following code:

```
shp.name = "citation" & shp.Tags("CitationID")
```

For the sortable styles with numbers, I had to change the text of the textbox as well, so that the correct number would be displayed. This is done with the following code:

```
If (shp.Tags("Sortable") = "True") Then
    If (shp.Tags("Style") = "http://www.zotero.org/styles/ieee") Then
        shp.TextFrame.TextRange = "[" & shp.Tags("CitationID") & "] "
    ElseIf shp.Tags("Style") = "http://www.zotero.org/styles/vancouver" Then
        shp.TextFrame.TextRange = shp.Tags("CitationID") & ". "
    End If
 End If
```

In this way all citations were renamed in the proper order and at the end of the function I return the “i” value, which is the value of the next citation index (as it was incremented one last time after the last citation was renamed). I call this function before adding a new citation box and use the value it returns as the number that should follow the string “citation” in the name of this new textbox, its CitationID tag, and formatted the text of the textbox in the correct bibliographical style. This way all citations have the proper order and there were no name conflicts. To allow the user to update indices when the user has deleted or rearranged citations (as there were no events for this), I added an Update Indices button to the Add-in which simply calls the UpdateIndices function.

Because the function to generate the references slides no longer worked with the new textbox names, I had to make some adjustments. In the beginning of the function I counted the number of citations in the presentation so I could generate an array of the correct length in order to hold all the citations. Next I looped over all the shapes in the presentation and if “citation” was in the name of the shape, I added this name to the array.

To get the references in the right order, I had to bubble sort this array based upon the CitationID. At first the sorting of the references was incorrect for some reason. I found out that I compared the CitationID, but that this was returned as a string instead of an int, which lead to incorrect sorting for numbers with more than one digit. Therefore I cast these strings to integers before comparing them. Once the sorting is completed, I used the names of the references in this sorted array to build a string of the formatted references and used #cit# as a delimiter. This is done as follows:

```
For l = 0 To (citationCount - 1)
    If (sld.Shapes(citationArray(l)).Tags("Sortable") = "True") Then
         citationString = citationString & "#cit#" & sld.Shapes(citationArray(l)).TextFrame.TextRange.text &
sld.Shapes(citationArray(l)).Tags("Text")
    Else
        citationString = citationString & "#cit#" & sld.Shapes(citationArray(l)).TextFrame.TextRange.text
    End If
Next
```

Once I created this new string, I split it up again to create an array that now contains the formatted references in the correct order. As a result, I can just loop through these and add them to the new slide, but if there were a lot of references, the resulting text would not fit on one slide. For this reason I had to find a way to determine how many references I should add on each slide. To do this I calculated how much space one line would take, by adding font size and spacing before and after it. I divided the height of the shape that would be needed to contain these references by the previously calculated space, to get the number of lines I could add to as slide. This is implemented as follows:

```
referenceLength = fontSize + pptSlide.Shapes(2).TextFrame.TextRange.ParagraphFormat.SpaceAfter +
pptSlide.Shapes(2).TextFrame.TextRange.ParagraphFormat.SpaceBefore
itemsPerSlide = pptSlide.Shapes(2).Height / referenceLength
```

I looped over all the citations in the array with the formatted citations and added references to the slide until the number of (itemsPerSlide – 1) was reached. The minus one leaves a bit of space for very long references. If the slide reached this point and there were still references left, a new reference slide is added and the same process of adding references to the slide was repeated until there were no references left.

To update the reference slides, I added some functionality in the beginning of this ZoteroAddReferencesSlide function. First I counted the number of reference slides present in the presentation. With a loop I deleted all the references slides. This way all references slides get cleared up and the new version can be added. The resulting code is:

```
For counter = 1 To slideCountRef
    ActivePresentation.Slides("References" & counter).Delete
Next
```

Some of the styles still did not work, as mentioned earlier in this report. The Citation style “3d edition of Modern Humanities Research Association” gave as reference:

\uc0\u8216{}Zotero | Start\uc0\u8217{} <https://www.zotero.org/start> [accessed 25 May 2015]

When putting these codes into an rtf document, I saw that these were single quotes. Just as with the double quotes, I replaced them with their ASCII character value. For the “Cell” style I found out that this needed to be treated as one of the sortable styles as well, otherwise the following error showed up: [CSL STYLE ERROR: reference with no printed form.]. To treat it as a sortable style I added the id of the style to the styles that need to use the bib variable for their citation in the integration.js file. This was the same case for the “Nature” style.

One problem I noticed while testing was that the “Nature”, “IEEE”, and “Vancouver” styles gave extra newlines when adding them to the references slide. I solved this by adding some code to the integration.js file that removed all newlines from the citation text, as follows:

```
citationText = citationText.replace(/\n/gm, “”);
```

The only thing left was to give the citation a more flexible layout. This is why I added an option to choose the text size for the citations in the preferences form. After doing this I also automatically make the textbox size adjust to the size of the text with the following code:

```
pptSlide.Shapes(textBoxName).TextFrame.TextRange.Font.Size = Val(GetFontSize)
pptSlide.Shapes(textBoxName).TextFrame.AutoSize = ppAutoSizeShapeToFitText
pptSlide.Shapes(textBoxName).TextFrame.WordWrap = msoFalse
```

To make the PowerPoint template installable, I could save it as a ppam or ppa file, wither of which is a PowerPoint Add-in file [31], with the first being a file with macro-enabled add-ins.

## 4.3 Installing Zotero PowerPoint

If you wish to install the Zotero PowerPoint extension, the following steps are required:

1. Download the software from GitHub: https://github.com/lejruben/zotero-powerpoint
2. Navigate to the “zotero-powerpoint-master\zotero-powerpoint-master\fully functional zotero version” folder and copy the zotero@chnm.gmu.eduf folder to whichever folder you want. Copy the path to this folder, as we’ll need it in step 4.
3. Go to the extensions folder by executing Run → %APPDATA% → Mozilla folder → Firefox Folder → Profiles folder → {some id}.default folder → extensions folder and create a text file here with the name “zotero@chnm.gmu.eduf”
4. Open this text file and paste the path to the zotero@chnm.gmu.eduf folder (e.g. C:\Program Files (x86)\zotero@chnm.gmu.eduf\), do **not** forget to end the path with a backslash.

5. Restart Firefox and accept the installation
6. Install the PowerPoint Add-in: Go to File → Options → Add-ins

   At the bottom at Manage, select PowerPoint Add-ins and press go. Click Add New... and navigate to the zotero-powerpoint-master\zotero-powerpoint-master\Zotero.ppam file. Press OKand agree to enabling macros. Close the dialog box and the add-in will have been installed. From now on all the functionality should be available in the Add-ins tab.

Note that the above operations will install the whole Zotero extension together with the added code for Zotero PowerPoint. If you already have the Zotero extension installed, this might lead to a conflict, hence you might need to remove it before adding the new extension.

# 5 Conclusions and Future work

In this chapter I offer my conclusions from this project. Section 5.1 addresses whether the goals were reached or not and what I could have done differently. Section 5.2 discusses the limitations which prevented me from making the project even better. Finally, Section 5.3 suggests some work that could be done in the future.

## 5.1 Conclusions

The goals of creating an extension that could add citations from Zotero and creating the reference slides, were met. I am quite happy with the final result. It was a good experience as I learned a lot about new technologies and am better prepared to handle large pieces of software.

If I had to do the project over again, I would not have started off by copying and modifying the code file by file. Instead I would first make a basic version of the desired application to see that I could make the technologies work together. In this way I could have noticed earlier that modifying the Zotero Word integration was not going to work out and I would not have spent so much time creating extensions or have faced all the C++/XPCOM problems (However, I did learn from this effort). I published the source code on GitHub and made a video about the project to encourage other developers to help improve this software.

GitHub: https://github.com/lejruben/zotero-powerpoint

Video: https://www.youtube.com/watch?v=SP02B2gAG50

## 5.2 Limitations

To make this software safe, reliable, and distributable, more developers would need to be involved. As I was not a specialist in the technologies used, the implementations are sometimes rough and could be improved. I tried a couple of times to get help and feedback from other Zotero developers, but had limited success.

## 5.3 Future work

The following steps would be needed for the Zotero developers to review my code and see whether it is safe and efficient enough to integrate into their software. Although I tried to get some interest in this code, I did not get much of a response. Some of the follow could be improved:

- Passing the citation between JavaScript and VBA could probably be implemented more elegantly than by writing to a temporary file and polling for this file.
- The classical view of the search query has been left out of this software. For a more complete version, this should be implemented.
- The editing of the citation within the search box was not implemented.
- There might be a more elegant solution than hardcoding the citation styles into VBA and JavaScript, thus new citation styles could be added.

If someone continues working on this project, he/she should read this report with the full code next to it. The report might be hard to understand without the code. I would also recommend first going over the code and look at better ways of implementing the operation, before expanding the functionality of the code. Be sure to check the Zotero Word integration extension as well, as the functionality you may want to implement, might already exist.

Finally, I can be reached at the address ruben.lejeune@gmail.com if there are any questions.

## References

[1] Bailey / Howe Library, 'Why Citations Matter.' The University of Vermont Libraries [Online]. Available: https://library.uvm.edu/guides/citation/why.php. [Accessed: 13-Aug-2015]

[2] 'zotero/zotero-word-for-windows-integration', *GitHub*. [Online]. Available: https://github.com/zotero/zotero-word-for-windows-integration. [Accessed: 13-Aug-2015]

[3] 'Extension Packaging', *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/Extension_Packaging. [Accessed: 14-Aug-2015]

[4] 'Building an extension | MDN.' [Online]. Available: https://developer.mozilla.org/en-US/docs/Building_an_Extension#XUL_Overlays. [Accessed: 15-Aug-2015]

[5] 'The Essentials of an Extension', *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/Overlay_Extensions/XUL_School/The_Essentials_of_an_Extension. [Accessed: 15-Aug-2015]

[6] 'Classic VB/API - How can I find a window outside of my program?-VBForums.' 23 August 2005 [Online]. Available: http://www.vbforums.com/showthread.php?356751-Classic-VB-API-How-can-I-find-a-window-outside-of-my-program. [Accessed: 20-May-2015]

[7] 'The SendMessage function.' 5 May 2011 [Online]. Available: http://vb.mvps.org/hardcore/html/thesendmessagefunction.htm. [Accessed: 20-May-2015]

[8] 'WideCharToMultiByte Visual Basic 6 API Function.' [Online]. Available: http://www.ex-designz.net/apidetail.asp?api_id=561. [Accessed: 20-May-2015]

[9] 'word_processor_plugin_usage [Zotero Documentation].' [Online]. Available: https://www.zotero.org/support/word_processor_plugin_usage. [Accessed: 14-Aug-2015]

[10] 'XPCOM Interfaces - Mozilla | MDN.' [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XUL/Tutorial/XPCOM_Interfaces. [Accessed: 27-Aug-2015]

[11] 'Components.interfaces', *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Language_Bindings/Components.interfaces. [Accessed: 17-Aug-2015]

[12] 'What is IDL? - Stack Overflow.' [Online]. Available: http://stackoverflow.com/questions/670630/what-is-idl. [Accessed: 15-Aug-2015]

[13] 'xpcom - Open / Read or Decompile .xpt to idl - Stack Overflow.' [Online]. Available: http://stackoverflow.com/questions/9410856/open-read-or-decompile-xpt-to-idl. [Accessed: 15-Aug-2015]

[14] 'An Overview of XPCOM', *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Guide/Creating_components/An_Overview_of_XPCOM. [Accessed: 16-Aug-2015]

[15] 'XPCOM guide', *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Tech/XPCOM/Guide. [Accessed: 16-Aug-2015]

[16] 'Change the Ribbon in Excel 2007 - 2013.' [Online]. Available: http://www.rondebruin.nl/win/s2/win001.htm. [Accessed: 16-Aug-2015]

[17] 'Ribbon Images & Labels Part II.' [Online]. Available: http://gregmaxey.mvps.org/word_tip_pages/ribbon_images_labels_part_II.html. [Accessed: 16-Aug-2015]
[18] 'In PowerPoint VBA, how do I refer to the ACTIVE slide?' [Online]. Available: http://answers.microsoft.com/en-us/office/forum/office_2007-powerpoint/in-powerpoint-vba-how-do-i-refer-to-the-active/5b4a7daf-9309-4159-b648-586d75895f85?auth=1. [Accessed: 16-Aug-2015]
[19] 'Getting started - Mozilla | MDN.' [Online]. Available: https://developer.mozilla.org/en-US/Add-ons/SDK/Tutorials/Getting_started. [Accessed: 26-May-2015]
[20] 'Firefox Extension Template.' [Online]. Available: http://davidwalsh.name/firefox-extension-template. [Accessed: 16-Aug-2015]
[21] 'Jonas' weblog – Setting up the Gecko SDK with Visual Studio 2010.' [Online]. Available: http://blog.peschla.net/2011/11/setup-gecko-sdk-with-vs2010/. [Accessed: 17-Aug-2015]
[22] 'How can I display a JavaScript object? - Stack Overflow.' [Online]. Available: http://stackoverflow.com/questions/957537/how-can-i-display-a-javascript-object. [Accessed: 20-Aug-2015]
[23] 'Save User Inputs as Default Settings - EnvisionCAD | MicroStation, InRoads, AutoCAD and Civil 3D Services.' [Online]. Available: http://envisioncad.com/tips/save-user-inputs-as-default-settings/. [Accessed: 20-Aug-2015]
[24] 'File I/O', *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/Code_snippets/File_I_O. [Accessed: 20-Aug-2015]
[25] 'Powerpoint run macro on timer.' [Online]. Available: https://social.msdn.microsoft.com/Forums/en-US/9f6891f2-d0c4-47a6-b63f-48405aae4022/powerpoint-run-macro-on-timer?forum=isvvba. [Accessed: 20-Aug-2015]
[26] 'Read Data from Text File using Excel VBA - Easy Excel Macros.' [Online]. Available: http://www.excel-easy.com/vba/examples/read-data-from-text-file.html. [Accessed: 20-Aug-2015]
[27] 'SpecialFolders and Windows Temp folder.' [Online]. Available: http://www.rondebruin.nl/win/s3/win027.htm. [Accessed: 20-Aug-2015]
[28] 'How to use VBA Replace function to replace double quotes in a string?' [Online]. Available: http://forums.esri.com/Thread.asp?c=93&f=992&t=92684. [Accessed: 21-Aug-2015]
[29] 'The citeproc-js Citation Processor.' [Online]. Available: http://gsl-nagoya-u.net/http/pub/citeproc-doc.html#generating-bibliographies. [Accessed: 21-Aug-2015]
[30] 'Slides.AddSlide Method (PowerPoint).' [Online]. Available: https://msdn.microsoft.com/en-us/library/office/ff746586.aspx. [Accessed: 21-Aug-2015]
[31] 'Add or remove add-ins.' [Online]. Available: https://support.office.com/en-gb/article/Add-or-remove-add-ins-64d3d147-98fb-4b82-8833-709d54e3ace1. [Accessed: 21-Aug-2015]

TRITA-ICT 2015:10