Imperial College London

BEng Individual Project

Imperial College London

Department of Computing

Integrating Quantitative and Computational Frameworks

Author:
Kavya Chopra
kc2320

Supervisor:
Prof. Paul Bilokon

Second Marker:
Dr. Konstantinos Gkoutzis

October 15, 2023

Abstract

Excel is a spreadsheet program owned by Microsoft used primarily for business applications, in the sense that it enables users to format, organize and calculate data in a spreadsheet.

Many quantitative and computational frameworks exist to integrate programming languages with Excel, but they are often difficult to integrate in a way that allows exportation of common datatypes and objects from one framework to another.

In this project we establish the common datatypes across some widely used languages such as Python and Java, and create a library **Zend** that integrates multiple quantitative frameworks together using sockets and a networking protocol built upon TCP (Transmission Control Protocol). A protocol in this context is defined as a set of rules that governs communication [23]. In this way, disparate frameworks can be integrated into a single distributed system.

For example, this system marshals common datatypes like integers, strings and NumPy arrays from Python, as well as similar datatypes from Java and Typescript. These extensions support the effective connection between quantitative frameworks.

Contained in this report are a detailed explanation of the implementation of the extended Excel Add-In and a formal specification of the API (Application Programming Interface), as well as the architecture of the **Zend** library itself.

Acknowledgements

I would like to extend my gratitude to my academic supervisor, Professor Paul Bilokon for his consistent advice and support through the process of developing this project.

Additionally, I would like to thank my second marker, Dr. Konstantinos Gkoutzis for his feedback during the crucial parts of the project, as well as my friends and colleagues for their advice and unconditional help.

Finally, I would like to thank my family whose unending support has always been an integral part of my progress.

Contents

List of Figures

Chapter 1

Introduction

Microsoft Excel is one of the most substantial and important business applications in the world. It is possibly the most suitable tool for synthesizing many different interdisciplinary concepts, which mostly pivot around business administration topics, ranging from Accounting, to Banking, Corporate Finance, Management, Strategy, Marketing. [55]

Despite its global popularity and widespread support, there are numerous tasks that one uses Excel for that can be monotonous, repetitive and tiring. Additionally, there are various other drawbacks to using pure Excel when trying to get more accurate results or handle larger datasets, which are discussed in the next section of this report. This makes it important to introduce integrating frameworks that automate such processes.

1.1 Objectives

This project aims to build a new library **Zend** that implements a framework with functionality that allows a user to write user-friendly client code in programming languages: Python and Java, to be able to transfer objects or data to and from Excel, and perform operations on these objects, as well as use functions directly defined in the library on data in Excel or the integrated languages. It builds upon the library "sydx" [19]. To achieve this goal, the aim can be broken down into the following objectives:

- Write a framework that integrates Excel, Python, and Java such that the user can send objects between programs in Java and Python, and an Excel worksheet.
- Design a protocol over TCP that decides how to establish a connection between a server and client, send requests and objects, and serialize those objects between programming languages.
- Write a WebSocket proxy implementation in Typescript to allow Excel to communicate with the other frameworks using regular sockets.
- Extend the functionality of the "sydx" library to demonstrate the value of being able to communicate between languages and Excel.
- Make the interface intuitive and user-friendly to enforce abstraction of the actual library code.

1.2 Challenges

The main challenges faced during the building of this library are listed below:

- **Familiarising with the sydx library** - This project is built upon the example of the sydx library (discussed in the next section) which attempts to integrate Python with Excel. The

first challenge was familiarising myself with the C# syntax and code written in the library, as well as navigating Visual Studio which was an unfamiliar software to use at the time to build and use the library.

- **Serialization and Deserialization** - The third main challenge was researching and determining the best serialization method and the libraries supporting cross-platform serialization/deserialization of objects. This challenge involved considering the difference in semantics in different programming languages and deciding how to translate between datatypes, as well as deciding the type-safety of the library.
- **Maneuvering type-safety** - The original sydx library had only designed the Python component of the project to run with the intended protocol, and was not a typed document. Hence it was a challenge to decide the constraints to put on the object datatypes that could be sent cross-platform to bridge the difference in language semantics (for example, Java can have lists that only contain one datatype, while Python does not need to specify a type for its objects).
- **Debugging in Visual Studio** - There were many instances where Visual Studio was difficult to get accustomed to. Debugging the Excel add-in was one of these challenges, which forced the project to change directions and switch to building the add-in in Typescript as opposed to C#. Learning the syntax and semantics of Typescript from scratch was an additional challenge.
- **Sockets Design** - A significant section of the code has been written to work using TCP sockets. However, Excel add-ins run on a browser environment and hence require web-sockets for communication over an HTTPS connection as opposed to regular TCP sockets. This required a choice to be made between switching back to the C# VSTO add-in, changing all the code to use web-sockets, or finding an alternate route. I settled on writing a proxy which is discussed later in this thesis.
- **Good design and object/type converters** - Other smaller challenges included having to determine what "good quality design" meant for this library both in terms of code and the user experience. Additionally, as this library handles sending objects from coding languages to Excel, another factor to consider was how to convert types from across platforms in a way that they are understandable and communicable to each other (for example: converting a Python list to a cell range in Excel).

1.3 Contributions

In light of the objectives listed above, the wider contributions of the project are as follows:

- Integrate multiple languages with Excel (as compared to other similar libraries which stop at integrating one language), which provides a more multi-purpose and user-friendly platform.
- Handle large datasets and mutable objects cross-platform in order to be able to compose features from multiple different languages to modify and update the data, without being confined to the features of one language's support.
- Perform more complex computations on data in Excel worksheets through the use of custom functions written in other languages.
- Potentially speed up various quantitative processes (example: analysis of data on large datasets) due to the faster execution and development time of coding languages.

Chapter 2

Background

This chapter explores the need for integrating computational frameworks with Excel, and some major concepts and libraries that were researched in the process of building this library. Additionally, this chapter also discusses the **"sydx"** library that this library was built upon and modified for the purposes of the project.

2.1 Excel limitations

"Microsoft Excel is a spreadsheet software application used to store, organize, and analyze data."[40] Its popularity has grown in various fields such as Marketing and Product Management, Finance and Accounting, Human Resource Management and many more. "Budgeting, analysis, forecasting, spotting trends, reporting"[40] are some of the major processes Excel is used for. However, a lot of these processes involve repetitive and monotonous tasks for the data management, which can be abstracted away by making it autonomous using an integrated framework. Such a library could also help make the process more user friendly and intuitive.

Some of the advantages of integrating Excel with other programming languages are as listed below:

- Programming languages are much more capable of handling both structured and unstructured data. Data manipulation is much easier in other languages, as compared to Excel where it can be time-consuming and complex.[12]
- Automating and re-running processes are much more feasible in other programming languages. For example, if one were required to run the same analysis on different data sets repetitively, Excel would require you to manually re-enter all the formulae in every new workbook. At the same time, a language like Python would allow you to import a script that re-runs the process on whatever data you provide.
- Excel takes more time for analyzing data compared with other direct statistical analysis software, especially when the data is numerous.[43]
- It is much easier to reproduce results by re-running a process in programming languages as compared to Excel.[12]
- Errors in coding languages are much easier to debug due to the error messages provided, as well as the possibility to comment out your code.
- Languages like Python have the capability to produce much more complex visualisations for data and even have more advanced statistical capabilities that allow the user to even create machine learning models.[12]

There are a number of frameworks and libraries that exist to integrate Excel with other coding languages, some of which are discussed in the **Related Works** chapter.

2.2 Server and Sockets

A server is a software or hardware device that accepts and responds to requests made over a network. The device that makes the request, and receives a response from the server, is called a client.[15] The client-server model, or client-server architecture, is a distributed application framework dividing tasks between servers and clients, which either reside in the same system or communicate through a computer network or the Internet. The client relies on sending a request to another program in order to access a service made available by a server.[50]

The client-server relationship communicates in a request–response messaging pattern and must adhere to a common communications protocol, typically TCP or IP.[50]

A TCP Server is an application that awaits TCP connections from TCP clients. TCP protocol maintains a connection until the client and server have completed the message exchange and determines the best way to distribute application data into packets that networks can deliver, transfers packets to and receives packets from the network. It manages flow control and retransmission of dropped or garbled packets.[50] Once a connection is established, both the server and the client can send a receive data in any order.[34] This is the type of server being used in the sydx library as well.

To create a TCP Server, we first create a first TCP socket and associate it to a particular port, on which it listens and waits for a connection. When a client request has been detected, we build a second socket for interaction with the client and exchange data with the client. Meanwhile, the first connection socket can continue to wait for a new connection, and even create new sockets to exchange data with other clients. Once the interaction with the client has completed, we close the client interaction socket. Finally once all interaction sockets have ideally be closed, we close the initial connection socket to kill the server and stop accepting client connections. [34]

One thing to note is that the client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.[11]

2.2.1 Creating servers in Typescript

Typescript follows the Single Threaded with Event Loop Model[31]. In other words, "Node JS Platform doesn't follow the Multi-Threaded Request/Response Stateless Model."[31] This means that a server does not create new threads for each connection, but rather uses this event-driven, non-blocking I/O model to handle them "concurrently" [31].

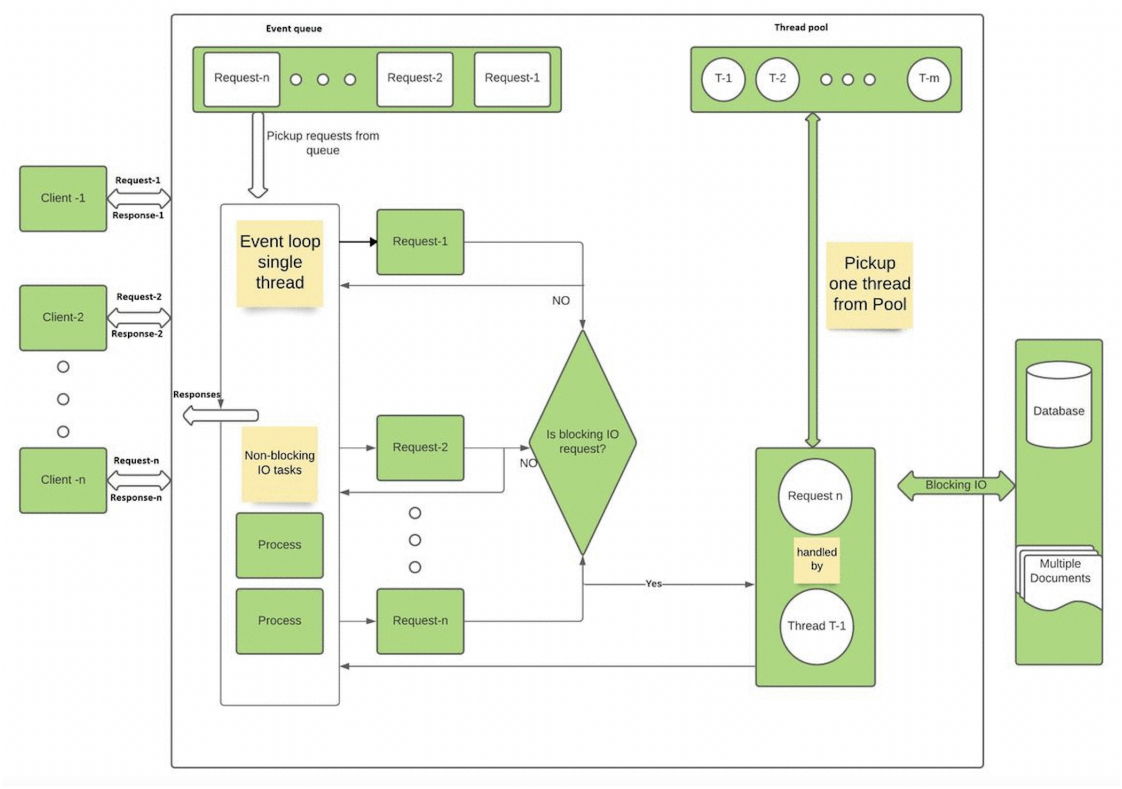
The Javascript runtime provides a call stack, where executable contexts are stacked to be executed (using the Last In First Out principle)[38]. The Web API and callback queue are provided by the browser and are used to handle requests sent by clients to the server in Typescript. When the call stack is empty, the event loop processes events or callbacks in the queue (which are added to the queue when the request is received by the server), and are executed accordingly. Hence multiple connections are handled without blocking.[31][25][38]

The reduction of the use of threads, memory, and resources is a major advantage to the event loop. This also means that it can handle more connections with ease.[31] A visual example of this event loop can be seen in [Figure 2.1] below.

2.3 WebSockets

"WebSocket gives you a bidirectional, full-duplex communications channel that operates over HTTP through a single socket."[35] With the WebSocket API, one can "send messages to a server and receive event-driven responses without having to poll the server for a reply"[10]. In other words, WebSockets keep the connection between the server and client open from which they can send messages back and forth rather than sending requests at random intervals.[35] In terms of security, it also "operates over Transport Layer Security (TLS) or Secure Sockets Layer (SSL)"[35].

Figure 2.1: Visual of the Event Loop[31]



WebSockets use the **"ws://"** or the **"wss://"** URL scheme, where the **wss** scheme is used for secure connections over SSL or TLS and the **ws** scheme is used for insecure connections[10]. SSL certificates secure the connection using encryption and authentication, therefore preventing interception of the connection[30].

For the purposes of this project, we use the `WebSocket` library API in Typescript order to make the creation and connections using WebSockets much easier. When a WebSocket sends a connection request to a server, it initially starts with a standard HTTP request that requests an upgrade to a websocket.

As web sockets are event-driven, the server listens for certain events once opened. The common events include:

- **onopen** - an event that is triggered when the WebSocket connection is successfully established.[8]
- **onmessage** - an event that is triggered when the WebSocket receives a message or data from the connection/server.[8]
- **onclose** - an event triggered when the connection is closed.[8]

Alternatively, the **addEventListener()** method is used to add an event listener to any of the events: "message", "open", "error", or "close".

2.4 Proxies

A proxy is an intermediate server that forwards requests and responses between a client and a server. In general cases, a proxy can be placed between a client and server for caching, security, and load-balancing purposes.[56]

A more specific application for proxies is to act as a bridge between different protocols. For example: bridging the gap between a WebSocket Server and TCP Server to allow communication

of data between WebSocket clients and TCP clients. The proxy would have the role of transferring data or requests that are sent to the WebSocket server from its client, in a way that is interpretable by the TCP server, so it can be appropriately sent to the relevant TCP clients.

There are two main types of proxies: forward proxies and revers proxies. A forward proxy "provides proxy services to a client or a group of clients."[47] It can anonymize the client (i.e. user) IP address, regulate traffic and increase security for users connecting to the server.[56]

A reverse proxy on the other hand, acts on behalf of the server.[47][53] This means that it accepts requests from external clients on behalf of the server that the proxy is protecting or is stationed in front of. This provides higher security for the server as the proxy can avoid overloading and enhance safety for the server by adding security layers and even SSL encryption if desired.[56]

For the purposes of this project, a reverse proxy was used (as described in the example above) in order to communicate between the different types of protocols and clients.

2.5 Centralized Systems

"Centralized systems are systems that use client/server architecture where one or more client nodes are directly connected to a central server."[27] They "follow a client-server architecture that is built around a single server with large computational capabilities. Less powerful nodes connected to the central server can submit their process requests to the server machine rather than performing them directly."[39]

For example, if one node or client wishes to send some data to all the nodes in the system, it can do so by sending that data to the proxy along with a request to be sent across the network, rather than doing that work itself. In this case, the main point of failure is the server, as the execution or working of the whole system is mainly dependent on that one component.[46] Additionally, client nodes can be easily added or removed without detrimental effects on the system as a whole.[39]

Additionally, one must be careful when choosing or designing algorithms for the server to handle client requests and responses. Inefficient algorithms for wait times and scheduling may lead to starvation.[39]

The alternative to a centralized system is a distributed system that treats all the components as nodes rather than having a central server to which all the components connect.

2.6 Serialization and Deserialization

The process of converting a data structure or object state into a storable format is referred to as serialization. The data is converted to a stream of bytes and then stored to a data stream. This allows for the encryption of data before sending it over a network. The resurrection of the stored data in the same or another computer environment is referred to as deserialization. In other words, the data stream (containing the stream of bytes) is converted back to a clone of its original data form (object, class, etc.).[17]

Binary Serialization, on the other hand, is converting the object in binary format as opposed to a stream of bytes, and being able to store it in a storage medium.[17] Serialization is also known as marshaling or deflating.

Many programming languages provide interfaces for serializing which can be implemented by the object classes that we want to serialize. Serialization and deserialization can hence be attained by writing codes for converting a class object into any format which we can save in a hardware and produce back later in any other or the same environment. [17]

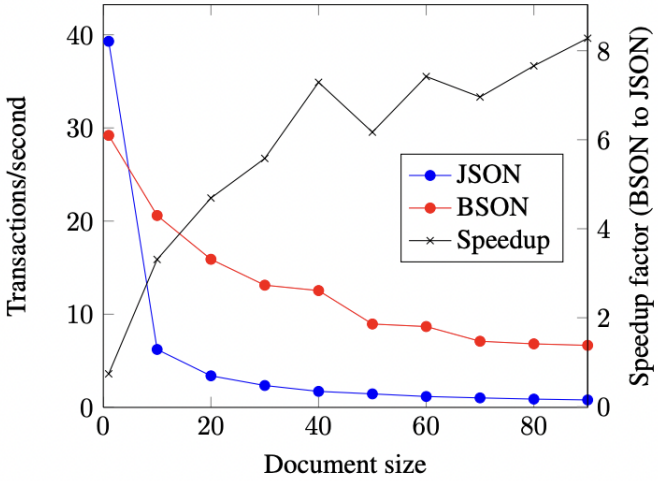
A common way of serializing data is creating a JSON file that stores the serialized data as a string (i.e. as simple text). Meanwhile, if Binary Serialization is carried out, then the data is stored in a BSON file. We choose to serealize to BSON in this project due to certain advantages over serealizing to JSON, as discussed below.

JSON documents are stored in human-readable text format in the database—without any opti-

mizations for efficient storage or fast traversal. This approach enables documents to be easily passed to clients and end users but does not necessarily enable the performant exchange to clients. BSON is a lightweight binary encoding for JSON-like documents, which is specifically designed for speedy traversal compared to the human-readable JSON format. BSON also has support for advanced data types.[57] For example, BSON documents may contain Date or Binary objects that are not natively representable in pure JSON. Second, each programming language has its own object semantics. JSON objects have ordered keys, for instance, while Python dictionaries (the closest native data structure that's analogous to JavaScript objects) are unordered, while differences in numeric and string data types can also come into play. Hence, BSON supports a variety of numeric types that are not native to JSON.[51]

However, on the other hand, the binary stored in BSON is only machine readable while JSON is also human readable.[57] However this minor disadvantage is well compensated for by the difference in performance speed, an example of which is demonstrated in [Figure 2.2][57] below.

Figure 2.2: Relative performance of BSON and JSON for increasing document size over a constant number of tuples



MongoDB is a popular document database that has libraries that allow serialization both to JSON and BSON. It stores the data in JSON-like files, however, it stores data in BSON format both internally, and over the network. This allows the user to enjoy the faster performance time of BSON even when they serialize to JSON files.[51]

2.7 Excel Add-ins

Excel Add-ins are extensions that allow a user to extend or add functionality to Microsoft Excel. There are two types of Excel add-ins: VSTO (Visual Studio Tools for Office) add-ins and Office Web add-ins. These add-ins can be used for automating tasks, enhancing user interface, integrating with other frameworks, or adding useful tools to Excel.[44]

Excel VSTO add-ins are specific to the Windows Excel desktop app (and can only be built and run on Windows using the Visual Studio IDE). It requires local code to be running on each user's device and a local install (with appropriate permissions and authorities enabled), which is "susceptible to environmental variables and potential conflicts with other add-ins."[36] Meanwhile, Web add-ins can be built and run cross-platform (like Windows, iOS, Mobile phones, and tablets on web browsers) without a local install, as it is web or cloud-based.[36] They essentially contain a web page embedded within the application[36], and a user can build it using either the Visual Studio IDE or the Visual Studio Code IDE.[1]

VSTO add-ins are built using C# or Visual Basic .NET and developed using the .NET framework. On the other hand, Web Add-ins are developed using Javascript or Typescript, HTML and CSS.[1]

Although VSTO add-ins provide deeper integration with the Excel application, it also requires

more resources for deploying, updating and maintaining than a Web add-in.[36]

Excel web add-ins can now run on Windows with a Microsoft 365 subscription and retail perpetual Office 2016 or later, as well as on Mac and on the Web. However, VSTO custom Excel add-in functions are not supported on Office on iPad, or on volume-licensed perpetual versions of Office 2019 or older.[26]

2.8 sydx

sydx[18] is a base example library containing a VSTO add-in written in C# that worked similarly to an Excel add-in, as well as a Python interface that sends simple marshaled objects between two or more Python programs. It contains some functions in the Excel component, which can be called and used in Excel when the add-in is enabled. This project is based around the example library, in terms of the protocol used for communicating between Python programs and the library's method of using TCP sockets, as well as building the add-in with the required API.

This library is built and run on Visual Studio, which allows for easier building of Excel add-in solutions and even templates for the same purpose. When the solution of the library is built, there are a number of .xll and other add-in related files that are added to the Excel component of the library, which allow for the add-in to be run on a local Excel workbook.

The main file in the Excel component is called Sydx.cs, which is written in C#. It contains code for a Server, which allows one to send and receive data or objects from other interfaces or components, as well as some functions which can be used as part of the add-in. The library also contains code that defines a Storage, that can be used to store values entered into Excel using a certain function and can be accessed later. The three main classes of this file are discussed in more detail below.

2.8.1 Sydx.cs - Server class

This class defines a simple TCP server. The Server class creates a socket and binds it to the specified port (specified by caller when creating the object), then enters an infinite loop to listen for incoming connections. There is an **AcceptCallback()** method called when a connection is made, that creates a new socket to handle the connection. The **ReadCallback()** method is called when data is received from the client. This method reads the data line-by-line and stores it in a buffer. The **ReadCallback()** method checks for an end-of-file tag in the client-sent message (denoting the end of the messages). If found, it calls the **Send()** method to send a "Thank you" response back to the client. Finally, the **SendCallback()** method is called when the response has been sent, to close the socket to end the connection.

This however, is different from the Python component as the Python component (sydx.py file) uses a different protocol for communication, which is also used in this project (i.e. the Zend library) and is discussed later in this report.

2.8.2 Sydx.cs - Storage class

The Storage class is a simple in-memory storage that can store an object by using a key and retrieve it later on. It contains a Dictionary object, a **Put()** method that can add an element to the Dictionary, and a **Get()** method that can retrieve an object from the Dictionary, using the key value associated with the object required. Each Dictionary element contains a key (or index) to identify the element, and a value (or intended object to be stored).

The Python component has a similar class with similar functions; however, the only difference is that it serializes all its objects before storing them in the storage so that the complete storage and its values are ready to send over the network.

2.8.3 Sydx.cs - ExcelFunctions class

This class contains a number of functions which can be called from Excel and used when the add-in is enabled. These functions are as follows:

- S_Port(): Takes a port number as argument and opens a port connection on given port number.
- S_Version(): Return the version of the Integral AddIn as a string
- S_Put(): Add an object to the storage, given a key and value
- S_Get(): Retrieve an object from the storage using the given key
- IntegralHello(): Takes the user's name as String and returns "Hello user name"
- S_Identity(): Return the same value that was passed as an argument
- S_Describe(): Takes an object and returns the datatype of that object.

The Python component of the sydx library has two main files in its **src** directory. These are the Sydx.py and converters.py files. The converters.py file defines functions to marshal (serialize) Python objects to JSON-like dictionaries, so that they can be sent over a network (via sockets) to other components or interfaces.

The Sydx.py defines a number of classes that form a simple message-passing system. The **Connection** class represents a connection to a remote host. It takes arguments for the host's IP address, remote process ID, and local port number. The **Connections** class is a container for Connection objects, and it has a method to send a message to all connected hosts.

The **Storage** class is similar to the Storage class written in the Sydx.cs file, in that it stores key-value pairs in a dictionary object, and has methods for storing and retrieving data.

The **Interpreter** class is responsible for interpreting incoming requests and dispatching them to the appropriate method. It receives a JSON string and deserializes it to a Python dictionary using the **json.loads()** method. It then checks the request type and calls the required method.

Chapter 3

Related Works

Most existing frameworks have limited functionality in terms of integrating multiple languages with Excel. This section discusses other libraries and works that have already been published that are similar in concept to this project or have components that have inspired it. Some examples of these frameworks include: xlwings, openpyxl, xlsx amongst others. Some of these pre-existing projects are discussed below, along with details about their functionalities and limitations.

3.1 xlsx - integrate with R

The xlsx package is one of the more basic packages that allows the user to directly manipulate an Excel workbook or sheet using the programming language R. For example: a user can set colours, fonts, add or remove sheets etc.[16] This package uses a subset of the Apache project which is a library written in Java [52]. Some of the functionality of this project has been integrated into the xlsx package.

An example code snippet using xlsx to write data into an Excel sheet is as follows[16]. This code creates an Excel workbook, adds a sheet named "addDataFrame1" to the workbook, creates a list of lists with specified values, adds the data to the sheet, sets font attributes for certain cells and then saves the workbook to the specified file location, finally producing the output in the Excel sheet:

```
#create Excel workbook
wb <- createWorkbook()
#create Excel sheet
sheet <- createSheet(wb, sheetName="addDataFrame1")
#data to add to the sheet
data <- data.frame(mon=month.abb[1:10], day=1:10, year=2000:2009,
       date=seq(as.Date("1999-01-01"), by="1 year", length.out=10),
       bool=c(TRUE, FALSE), log=log(1:10),
       rnorm=10000*rnorm(10),
       datetime=seq(as.POSIXct("2011-11-06 00:00:00", tz="GMT"),
       by="1 hour",
       length.out=10))
#set font attributes for cells of columns in the workbook
cs1 <- CellStyle(wb) + Font(wb, isItalic=TRUE) # rowcolumns
cs2 <- CellStyle(wb) + Font(wb, color="blue")
cs3 <- CellStyle(wb) + Font(wb, isBold=TRUE) + Border() # header
#add data to the sheet; add sheet to workbook
addDataFrame(data, sheet, startRow=3, startColumn=2,
             colnamesStyle=cs3, rownamesStyle=cs1,
             colStyle=list(`2`=cs2, `3`=cs2))
#save the workbook
saveWorkbook(wb, file)
```

Although this library houses various functionalities that allow a user to manipulate Excel through R, it has limited functionality in terms of going beyond the features that Excel already provides. Mostly this package is used to read data directly from Excel sheets, and write to them, as well as change superficial attributes like fonts, borders, and text colour. On the other hand, this package does not have the functionality to send or receive coding language-style objects or unstructured data and doesn't specify additional functions that go beyond simple modifications, apart from those defined in the Apache project.

Finally, in terms of security, xlsx does not provide any strong protection for its data and its manipulation.

3.2 openpyxl - integrate with Python

openpyxl is an open source Python library to read/write Excel 2010 files with the suffixes xlsx/xlsm/xltx/xltm.[22] This library has more functionality than the xlsx package, but is still limited to helping lessen the manual work involved in writing to and reading from Excel sheets.

It also works with the popular Python libraries - Panda and NumPy, and provides the functionality to create various charts for visualization (for example: bar charts, pie charts, other 2D and 3D charts etc.) based on the data provided. Additionally, the user can manipulate Excel spreadsheets by adding/reading data, adding/removing rows and columns, adding figures and pictures, parsing formulas, adding filters and sorts, and even has a type for supporting dates and timestamps. Similarly to the xlsx package, openpyxl has the feature to modify cell styles as well.[29]

The client code for utilizing this package is comparatively more intuitive and user-friendly. For example, the following code snippet produces a similar output as the xlsx code snippet in the previous subsection:

```
import openpyxl
#create Excel workbook
wb = openpyxl.Workbook()
#create Excel sheet
sheet = wb.active
sheet.title = "addDataFrame1"
#add data to the sheet
#can replace direct addition of data here using for-loops
data = [
    ["month", "day", "year", "date", "bool", "log", "rnorm", "
    datetime"],
    [month[1], 1, 2000, "1999-01-01", True, log(1), 10000*rnorm(1),
    "2011-11-06 00:00:00"],
    [month[2], 2, 2001, "2000-01-01", False, log(2), 10000*rnorm(1)
    ,
    "2011-11-06 01:00:00"],
    #similarly produce remaining row data until row 10 as below
    [month[10], 10, 2009, "2008-01-01", False, log(10), 10000*rnorm
    (1),
    "2011-11-06 09:00:00"]
]
#add data to the sheet
for row in data:
    sheet.append(row)
#set attributes for cell and text style
bold_font = openpyxl.styles.Font(bold=True)
blue_font = openpyxl.styles.Font(color="0070C0")
italic_font = openpyxl.styles.Font(italic=True)

for i in range(1, sheet.max_row+1):
    sheet.cell(row=i, column=1).font = italic_font
```

```
#set header as bold text
for j in range(1, sheet.max_column+1):
    sheet.cell(row=1, column=j).font = bold_font

for i in range(2, sheet.max_row+1):
    sheet.cell(row=i, column=2).font = blue_font
    sheet.cell(row=i, column=3).font = blue_font
#save workbook
wb.save("file.xlsx")
```

Although the features of this package are more advanced than those of xlsx, openpyxl still does not provide a way to directly convert between unstructured data and a type that is readable in Excel. Additionally, this package mostly abstracts away monotonous work involved in writing or reading data to excel (and creating read-only workbooks for security reasons or to handle large datasets that slow down the processing), but does not add any major functionality to Excel.

Meanwhile, in terms of security, openpyxl provides the functionality to create a read-only spreadsheet of data to avoid its manipulation.[22]

3.3 xlwings - integrate with Python

xlwings is a Python library used to call Excel from Python and Python from Excel. It is comparatively more popular and has a much more vast functionality compared to the other libraries listed in this report. It can read from and write to Excel workbooks from Python, include Matplotlib figures as pictures, has convenience functions (such as range expanding), has powerful converters to handle Python objects like NumPy arrays and Pandas DataFrames (in both directions from Python to Excel and vice versa), and even has the functionality to write User Defined Functions (UDFs). (UDFs are only supported on Windows) Additionally, xlwings has its own custom add-in functions that are written in Python and can be called from Excel, provided the add-in has been installed on that particular machine. xlwings also supports multi-threading and multi-processing (multi-processing is supported on Windows).[13]

Comparatively, this library has a much more user-friendly interface. For example, writing and reading from Excel can be done in two simple lines of code[13]:

```
#write to excel cell 'A1'
sheet['A1'].value = 'Foo 1'
#read value contained in cell A1
sheet['A1'].value
#Output: 'Foo 1'
```

Additionally, xlwings provides a number of advanced features such as converters (which allow a user to read a value differently from its default converted datatype or form), customized debugging, extensions and even supports threading and multiprocessing (multiprocessing is only supported on Windows). Their convenience functions range from returning the transpos of the table to making it easy to format the ranges in Excel.[13]

To demonstrate the simplicity of using this library, the code below provides the same output as the example discussed previously in the openpyxl and xlsx subsections (we assume rnorm and month.abb functions are defined in the same context as they are previously used):

```
import xlwings as xw
# create a new workbook
wb = xw.Book()
# create a new sheet
sht = wb.sheets.add("addDataFrame1")
# define the data to be added to the sheet
data = {
    'mon': month.abb[1:10],
    'day': range(1, 11),
```

```
    'year': range(2000, 2010),
    'date': list(range(1999, 2009)),
    'bool': [True, False],
    'log': [log(i) for i in range(1, 11)],
    'rnorm': [10000*rnorm(10)],
    'datetime': list(range(2011, 2021))
}
# add the data to the sheet starting at cell A1
sht.range('A1').value = data
# define font attributes for cells of columns in the workbook
bold = xw.Font(bold=True)
italic = xw.Font(italic=True)
blue = xw.Font(color='blue')
# apply the font attributes to the appropriate cells
sht.range('A1:A10').api.Font = italic
sht.range('B1:B10').api.Font = blue
sht.range('C1:I1').api.Font = bold
# save the workbook
wb.save('temp.xlsx')
```

Alternatively, we can also use the pandas library in Python to create the dataframe instead, as seen below:

```
# create a pandas dataframe
data = pd.DataFrame({'mon': month.abb[1:10], 'day': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                     'year': [2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009],
                     'date': pd.date_range('1999-01-01', periods=10),
                     'bool': [True, False]*5, 'log': np.log(1:10),
                     'rnorm': 10000*np.random.randn(10),
                     'datetime': pd.date_range('2011-11-06', periods=10, freq='H')})
```

Therefore, xlwings is certainly more advanced by the standards of other existing libraries (to our knowledge). Its main disadvantage is its inability to support other programming languages interfaces.

3.4 Apache POI and JExcel - integrate with Java

The Apache POI[3] library is a comparatively complex library for working with Excel files, that supports both .xls and .xlsx files. It provides the **Workbook** interface for modeling an Excel file, and the **Sheet, Row and Cell** interfaces that model the elements of an Excel file, as well as implementations of each interface for both file formats. It contains the **XSSFWorkbook, XSSFSheet, XSSFRow** and **XSSFCell** classes that support the .xlsx files.[52]

Java methods for modifying and reading from an Excel file are comparatively wordy when compared to Python libraries. Due to having strong typed constraints, the Apache library even has separate methods to read different datatypes from cells in Excel. For example, when the cell type enum value is STRING, the content will be read using the **getRichStringCellValue()** method of Cell interface:[52]

```
data.get(new Integer(i)).add(cell.getRichStringCellValue().
    getString());
```

Given below is an example of a Java main method utilizing the Apache POI library to create a simple table with 2 columns and 2 rows: a header row with topics "Name" and "Age", and second row consisting of certain values corresponding to their headers. (code from [52])

```
//create workbook
Workbook workbook = new XSSFWorkbook();
//create sheet and set column widths for the first and second
    columns
Sheet sheet = workbook.createSheet("Persons");
sheet.setColumnWidth(0, 6000);
sheet.setColumnWidth(1, 4000);
//create row in the table
Row header = sheet.createRow(0);
//create a cell style: solid lightblue foreground
CellStyle headerStyle = workbook.createCellStyle();
headerStyle.setFillForegroundColor(IndexedColors.LIGHT_BLUE.
    getIndex());
headerStyle.setFillPattern(FillPatternType.SOLID_FOREGROUND);
//create custom font for column headers: Ariel, height 16 points,
    bold
XSSFFont font = ((XSSFWorkbook) workbook).createFont();
font.setFontName("Arial");
font.setFontHeightInPoints((short) 16);
font.setBold(true);
headerStyle.setFont(font);
//create header cells of the columns
Cell headerCell = header.createCell(0);
headerCell.setCellValue("Name");
headerCell.setCellStyle(headerStyle);
headerCell = header.createCell(1);
headerCell.setCellValue("Age");
headerCell.setCellStyle(headerStyle);
CellStyle style = workbook.createCellStyle();
style.setWrapText(true);
//create second row and add value "John Smith" to first column
Row row = sheet.createRow(2);
Cell cell = row.createCell(0);
cell.setCellValue("John Smith");
cell.setCellStyle(style);
//add value "20" to second row second column, i.e. age of John
    Smith
cell = row.createCell(1);
cell.setCellValue(69);
cell.setCellStyle(style);
//write above values to temp.xlsx file in the current directory
File currDir = new File(".");
String path = currDir.getAbsolutePath();
String fileLocation = path.substring(0, path.length() - 1) + "temp.
    xlsx";
FileOutputStream outputStream = new FileOutputStream(fileLocation);
workbook.write(outputStream);
//close workbook
workbook.close();
```

JExcel is a comparatively simpler library that supports .xls files but not .xlsx files. The **Workbook** class represents the entire collection of sheets. The **Sheet** class represents a single sheet, and the **Cell** class represents a single cell of a spreadsheet. Given below is a code example that executes the same output as the code example above for the Apache POI library, but instead using JExcel [52]:

```
//create file in current directory
File currDir = new File(".");
String path = currDir.getAbsolutePath();
```

```
String fileLocation = path.substring(0, path.length() - 1) + "temp.
    xls";
WritableWorkbook workbook = Workbook.createWorkbook(new File(
    fileLocation));
//create sheet in workbook
WritableSheet sheet = workbook.createSheet("Sheet 1", 0);
//create custom header font
WritableCellFormat headerFormat = new WritableCellFormat();
WritableFont font
  = new WritableFont(WritableFont.ARIAL, 16, WritableFont.BOLD);
headerFormat.setFont(font);
headerFormat.setBackground(Colour.LIGHT_BLUE);
headerFormat.setWrap(true);
//create header "Name" in first row first column
Label headerLabel = new Label(0, 0, "Name", headerFormat);
sheet.setColumnView(0, 60);
sheet.addCell(headerLabel);
//create header "Age" in first row second column
headerLabel = new Label(1, 0, "Age", headerFormat);
sheet.setColumnView(0, 40);
sheet.addCell(headerLabel);

WritableCellFormat cellFormat = new WritableCellFormat();
cellFormat.setWrap(true);
//add data below each header on next row respectively
Label cellLabel = new Label(0, 2, "John Smith", cellFormat);
sheet.addCell(cellLabel);
Number cellNumber = new Number(1, 2, 69, cellFormat);
sheet.addCell(cellNumber);
//write data to workbook
workbook.write();
//close workbook
workbook.close();
```

Chapter 4

Overview

This chapter provides an overview of the overall idea behind the functioning of the library and the technologies used to build it.

4.1 Technologies used

This section discusses the technologies and software used in the building of this project, which substantially affected the building process.

4.1.1 Programming Languages

For the purposes of building the Excel add-in, the sydx library contained a C# component that created a VSTO add-in for Excel. However, the server was unable to communicate with the Python component due to differences in the design of their protocols.

As the project progressed, it was discovered that building an Excel Add-in in Typescript was more in line with the objectives of this library and easier to debug as it ran on a browser environment. Hence, Typescript was adopted for writing the Excel component, as well as the proxy that bridged the connection between the WebServer on the Excel side and the TCP server that connected to external components.

The sydx library contained a Python component as an external framework that defined a protocol over TCP to connect to other components. Hence, this component was built upon to be integrated into the Zend library.

Finally, Java was chosen for the third and final component of the Zend library as it is easy to compile and debug and allows you to create modular programs and reusable code due to its object-oriented nature.[21]

Although the C# component was developed through the project, it became redundant after the Typescript component was used instead.

4.1.2 C# and Typescript

Although the **sydx** library originally used C# for creating the VSTO add-in, it was later decided in the project to use Microsoft's Typescript API for building a custom Excel add-in. At this time, the Visual Studio IDE was being used to build and test the add-in, as VSTO add-ins contain various components specific to this IDE (such as the **.sln** solution file). This made the writing and debugging of the add-in inflexible as preferred or familiar IDE's could not be used, and when Visual Studio was not compatible with the given platform or operating system, or did not perform as expected, there were insufficient tools for support and to fall back on.

Upon switching to Typescript and the Visual Studio Code IDE, the project was built using a "Shared runtime", i.e. a browser runtime. This kind of project contains a task pane as well which

is auto-updating (as the code changes) due to it being part of a webpage embedded into the Excel application. Having a shared runtime also means that the add-in is closed (i.e. the program stops running) when the Excel application is closed.[9]

Debugging at this stage of the project was done in a number of steps with certain tools; for example: the live TCP dump provided information about the passing of requests and responses between components, and gave hints as to where the communication issues lay; after clearing the Office application's cache[4] (using a script **clear-excel-cache.sh** written in the repo. Additionally, the "Inspect Element" tool of the "Safari Web Inspector" was used to set breakpoints in the code and thus debug the add-in by getting a deeper understanding of the code's effects.[20]

Going forward, this thesis includes information about both the C# and Typescript components that were written through the progression of the project even though the C# component is now redundant. This is to ensure that light is brought to the effort, ideas, and process of building both components and the differences reflected in their design choices.

4.1.3 Important libraries

Some of the important libraries used in this project are:

- **net:** This Typescript library was used to create TCP servers in the Typescript component for communication with external components.
- **promise-socket:** This Typescript library was used to create TCP clients in the Typescript component.
- **ws:** This library was used for its WebSocket API and for creating WebSocket servers and clients for communication between the Excel component and proxy.
- **bson or BSON:** The "bson" library in Python and "BSON" library in Typescript were used for serializing and deserializing objects to and from BSON.
- **org.mongodb.bson:** Similar to the purpose of the "bson" library in Python, this library was also used for serialization/deserialization purposes. Additionally, the **org.mongodb.Document** class was used to store serialized objects in the Java component.

4.2 The outcome

As described before, the aim of this project was to create a library containing an Excel add-in such that it can integrate or communicate with programs written in other languages (i.e. with other frameworks). This section discusses the high-level working of the library that allows the communication between components, and the ideas behind the way it was built.

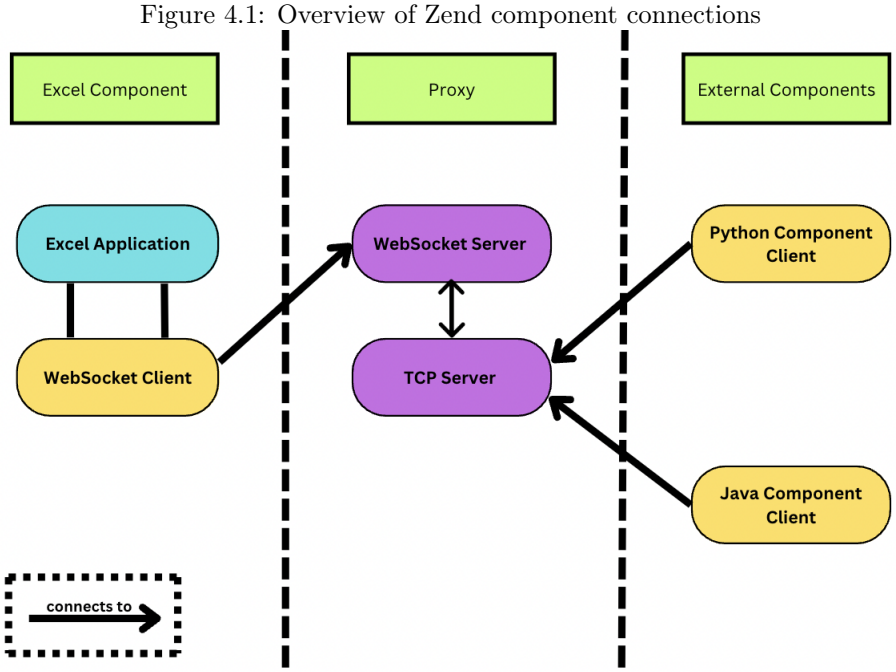
The library's main component the Excel add-in, starts with an Excel worksheet that has the add-in of this library enabled. The user can then use the `PUT()` and `GET()` methods from the add-in to add or retrieve values from the local storage of the library. There is a `PORT()` method that connects the add-in to a running proxy, so that the add-in can communicate with other external frameworks connected to that proxy. A depiction of the overall architecture and the connection of components in the library is portrayed in [Figure 4.1] below.

Before calling the `PORT()` method in the add-in, the user must manually start the proxy (which allows connection between the add-in and the other components of the library) which is compiled using **webpack** and run using **node.js**.

When the `PORT()` method is called by passing in the port number that the TCP server must be opened on, the Excel component (written in Typescript) connects to the proxy. It, therefore, opens the connection for the Excel component to communicate with any other frameworks connected to the proxy.

The user can similarly call the `openPort()` (Java)/`open_port()` (Python) method with an unused port number, followed by the `connect()` method with the TCP server port number as a parameter

Figure 4.1: Overview of Zend component connections



in the Java and Python components, to be able to communicate with each other and with the add-in if it is connected.

In this mechanism, we treat the Add-in (hidden behind the proxy) as the main connection point (i.e., the server) to which other frameworks can connect. This ensures that the Add-in receives any requests from the frameworks first before they are sent to the other connections so that its storage values remain up to date.

As evident from the working of the library, this project is based upon a centralized system architecture. All the components are intended to be treated as clients and used such that they directly connect to the Excel component, whose proxy contains the server. When a user "put"s a value in one component's storage, that value is sent to the Excel component (via the proxy) to be synced, and that value is in turn sent to all the other connections of the proxy to sync all the components' storage values.

On the other hand, if the external components are connected to each other without also being connected to the proxy, the syncing of all components cannot be guaranteed. For example, if the Java component connected to the Python server and only the Python server was connected to the proxy, if a user added something to the Java storage, this value would be sent and added to the Python storage but not to the proxy (as external components do not re-distribute values they receive on their server).

Once the components are connected, the user can use `PUT()`/`put()` and `GET()`/`get()` methods to pass or add, and retrieve supported datatypes across the Excel add-in and other frameworks respectively.

The library API, architecture chosen for the library and the design and functioning of the code written, as well as a specification of the chosen protocol in order to make the library work as intended are discussed in the following sections.

Chapter 5

The Zend API

This chapter discusses the functions that can be called by the user for each platform. These functions are located in a Zend.java class for the Java component, as export functions in the Typescript component's `functions.ts` file, and as global functions in the Python component. Any differences in these functions across platforms are also discussed below.

5.1 put() method

This method is called by the user to add a key-value pair to the storage of that framework. The key must be a string and the value can be any object type supported by that respective framework. Hence, the parameters taken by this function are the string `key` and the object `value`. When this method is called, a **Put Request** is sent to all the connections of that framework so that the key-value pair is added or updated in the other connected frameworks, and the value is also added to the framework's storage.

In the Excel Add-in, the **=PUT()** method in an Excel worksheet, takes as a parameter a double-dimensional array of `any` type so that it can accept a range or table of Excel cells.

5.2 get() method

This method is called using **=GET()** in the add-in enabled worksheet when the user wishes to retrieve the object value associated with a certain string key in the storage. It takes a string `key` as a parameter, searches the storage for that framework, and returns the associated value if it exists. There is an optional second boolean parameter that allows the user to print the range of values column-wise rather than row-wise. If the second parameter is set to **TRUE** then it prints column-wise. If it is set to **FALSE** or is not present in the set of parameters entered by the user then it prints row-wise.

In the Excel component, this method returns a two-dimensional array (of `any` type) to depict that it returns a range of cells.

In the external components, this method directly prints whatever value is associated to the string key.

5.3 openPort() method

This method is called to open a server port to listen for incoming connections on any component. It is present in the **Zend.java** class for the Java component, in the **ExcelCustomFunctions** class in the C# component, as a global function `open_port()` in the Python component, and in the proxy.ts file in the Typescript component for communication with external frameworks. It takes

the `port` number as a parameter and opens a server for that framework on the specified port. A user cannot open more than one port for a given framework.

The Java component also provides a `closePort()` method that closes the server port and thus ends its connection to all its clients.

In the Excel component, the user can call the **=OPEN_PORT()** method passing in as a parameter the port number on which the TCP server must open and start listening for connections. When this method is called, the add-in component in Typescript opens a WebSocket that connects to the proxy and sends an **Excel Open Port Request** to the proxy. It `await`s the **Excel Open Port Response** from the proxy and returns the string field **result** from the response.

Any requests or responses being sent or received over the WebSocket connection are serialized and deserialized to or from BSON, similar to the way the TCP connections are handled in this library.

In the case that the **id** field of the message being received by the add-in is present and exists in the add-ins Map **pending responses** (which maps from numbers **id**s to resolve functions), then the **id** is removed from the map and the corresponding response is resolved.

If the **id** field is not present in the map (or in the response) then this indicates that the message is an **Update Message** rather than a response to a specific request from the Add-in. This response is handled by updating the relevant cells in the Excel worksheet, and relevant values in the storage in accordance with the values sent in the response.

5.4 connect() method

This function is not present in the Excel add-in, as the aim of the library is to treat the Excel worksheet as the true center point of connection (i.e. the server and handler of all clients/connections) between all the frameworks. In the Java and Python components, this method creates a Client instance and connects to the specified `host`**(: string)** - the server hostname, and the `port`**(: Integer)** parameters specified by the user in the method. It returns the connection handle to prove the success of the connection. The server it connects to is intended to be the TCP server running in the proxy, to be able to connect to the Excel component.

Chapter 6

The Communication Protocol

This section discusses the protocol that the library uses for its components to pass information between each other. There are three types of requests that a component can send to its connections, each of which triggers a certain response back to the sender. A notable design choice in terms of handling these requests and responses is that whenever data is sent over the network, the sender first sends an integer denoting the length of the BSON byte array of the request or response that is being sent, followed by the actual request. This is so that it is much easier to read the request given the size of the request to be read. We know that an integer sent over the network would be 4 bytes in length, so we read those 4 bytes first; followed by the request to the length specified by the integer. The requests that the library can send between frameworks are listed below:

6.1 Between Proxy and External Frameworks

The requests illustrated below denote the protocol used between frameworks and the proxy. Due to the nature of the Excel Add-in, it must first communicate with the proxy using WebSockets to send requests and receive responses (using HTTP requests). Since both the proxy and add-in are written in the same framework (i.e. Typescript) and this communication is different and not indicative of the protocol used above, there is a separate protocol used to communicate between the Add-in API and the proxy (which in turn communicates with the other frameworks using TCP sockets).

6.1.1 Handshake Request

When a connection request (i.e. a Handshake Request) is made to the server, the framework creates a string **UUID** (Universal Unique Identifier)[24] and a Connection by including data from the request, and adds them to the `Connections` instance **"connections"**.

This request has the structure:

```
{
    request_type: "HANDSHAKE_REQUEST",
    host: string, //<string host-name of client>
    pid: int, //<integer process id>
    local_port: int //<integer port number of client>
}
```

When a server receives this request, it sends back a **Handshake Response** of the structure:

```
{
    response_type: "HANDSHAKE_RESPONSE",
    connection_handle: string //<string UUID to identify
                              // connection>
}
```

6.1.2 Sync Storage Request

Following the Handshake Request, once the client receives a Handshake Response from the server that confirms the connection, the client sends a `Sync Storage Request` in order to ensure that their global **storage** values are synced (i.e. contain the same key-value pairs) and updated.

The Sync Storage Request has the structure:

```
{
    request_type: "SYNC_STORAGE_REQUEST",
    storage: Map<string, serializedObject> // <a map of the
             // storage keys to its serialized values>
}
```

On receiving this request, the server puts all the values in the client's storage snapshot within its own storage (in addition to its current storage) and sends back a `Sync Storage Response` with its own updated snapshot of its values, so that the client too can update its storage with the new server storage values.

The Sync Storage Response has the structure:

```
{
    request_type: "SYNC_STORAGE_RESPONSE",
    storage_snapshot: Map<string, serializedObject> // <a map
                      // of the storage keys to its serialized
                      // values>
}
```

6.1.3 Put Request

This is the request that is sent to all the connections of a server when a user uses a specific framework's `put()` method to put a key-value pair into its storage. The request includes the string key put into the storage and the serialized value associated with that key.

The Put Request has the structure:

```
{
    request_type: "PUT_REQUEST",
    name: string, // <the key>
    value: serialized_object // <the serialized value>
}
```

Upon receiving this request, the client sends back a response that indicates the success of putting the value into their storage.

A Put Response has the structure:

```
{
    response_type: "PUT_RESPONSE",
    result: "success"
}
```

6.2 Between the Excel Add-in and Proxy

The protocol used for communication between the Add-in and the proxy is described below:

6.2.1 Excel Put Request

This request is sent to the proxy when the user uses the =**PUT()** method in the Excel worksheet using the add-in. It is similar to a **Put Request** such that it contains the name and value, but this

request also contains an **id** field that gives the request a unique ID by which it can be identified. This allows the proxy to send back a response using the same ID so that the Add-in component knows which request that response belongs to. Having this mechanism allows for multiple requests to be sent without having to wait for the response to each request before the next one is sent.

The Excel Put Request has the structure:

```
{
    id: number, //<the unique request ID number>
    request_type: "EXCEL_PUT_REQUEST",
    name: string, // <the key>
    value: ZendType // <the serialized value>
}
```

Upon receiving this request, the proxy sends back a response indicating the success of the request.

The Excel Put Response has the structure:

```
{
    id: number, // <the unique request ID number>
    response_type: "EXCEL_PUT_REQUEST",
    result: "success"
}
```

6.2.2 Excel Get Request

This request is sent to the proxy when the user uses the =**GET()** method in the Excel worksheet using the add-in. It contains the **name** field that is entered by the user into the add-in function, so that they may retrieve the value associated with that key.

The Excel Get Request has the structure:

```
{
    id: number, // <the unique request ID number>
    request_type: "EXCEL_GET_REQUEST",
    name: string // <the name key for which the value is to be
                 // returned>
}
```

This response triggers a response containing the value associated with the given key in the request.

The Excel Get Response has the structure:

```
{
    id: number, // <the unique request ID number>
    request_type: "EXCEL_GET_RESPONSE",
    result: ZendType // <the value to which the name was
                     // associated>
}
```

6.2.3 Excel Open Port Request

This request is sent to the proxy when the user uses the =**OPEN_PORT()** method in the Excel worksheet using the add-in. The request contains the port on which the proxy is running, so that the Add-in can connect to the proxy and therefore communicate with any other components or frameworks connected to that proxy.

The Excel Open Port Request has the structure:

```
{
    id: number, // <the unique request ID number>
    request_type: "EXCEL_OPEN_PORT_REQUEST",
```

```
        port: number // <the port on which the proxy is running>
    }
```

This request triggers an Excel Open Port Response containing a string field that indicates the success or failure of the connection request.

The Excel Open Port Response has the structure:

```
    {
        id: number, // <the unique request ID number>
        request_type: "EXCEL_OPEN_PORT_RESPONSE",
        result: string // <success or failure of the connection>
    }
```

6.2.4 Excel Update Message

This message is sent from the proxy to the Excel add-in component when the proxy receives a **Put Request** from one of its external connections. It contains the key name and serialized value to be added or updated in the **storage** in the add-in component.

The Excel Update Message has the structure:

```
    {
        response_type: "EXCEL_UPDATE_RESPONSE",
        name: string, // <the key>
        value: ZendType // <the serialized value>
    }
```

6.2.5 Excel Bulk Update Message

This message is sent from the proxy to the Excel add-in component when the proxy receives a **Sync Storage Request** from one of its external connections. It contains the map of all keys to serialized value pairs to be updated in the add-in storage.

The Excel Bulk Update Message has the structure:

```
    {
        response_type: "EXCEL_BULK_UPDATE_RESPONSE",
        storage: ZendMap // <the key-serialized value pairs>
    }
```

Chapter 7

The Library Architecture

This chapter discusses the classes that are present in every component of the library (i.e. C#, Java, Typescript and Python), the differences and similarities in the design choices made during the process of writing the components and the reasons for those choices. The roles of the different classes included in each component of **Zend** and how they work to fit into the total architecture are reviewed below. Note that the Java and Python (and C#) components share a similar architecture and method of passing requests and information.

7.1 Server class

This class acts as the server, which has functions to open a server port (port number being specified by the user) and to listen to clients.

For the purposes of this project, a user must also be aware of which port numbers can be used to open a server or connections to the server. IANA (Internet Assigned Numbers Authority) maintains a list of the reserved, registered, and ephemeral ports that are in use in the Service Name and Transport Protocol Port Number Registry [5]. Registered ports are of the range 1024 to 49151 and are assigned by IANA upon application and approval [28]. This is the general range of ports for a user to choose from when opening a server or client connection. If the port number is too high, it may overlap with the range for ephemeral ports, which are transient or are open for a short amount of time, and therefore not recommended to be used.

7.1.1 Java, Python and C# components

- **Constructor:** In the Java, Typescript, and C# components, the Server class is structured to have a constructor that takes `host`**(: string)** - the hostname on which the server opens, `port`**(: int)** - the port on which the Server listens for clients, `storage`**(: Zend.Storage)** - the storage used globally through the program so that the server can sync the storage across its clients, `connections`**(: Zend.Connections)** - a list of other clients (i.e. connections) connected to this server.

  In the Python component, the Server class constructor takes parameters for the `host` and `port` similar to as described above and also takes a `request`**(: Zend.Interpreter | Zend.Request)** parameter that takes an instance of the `Interpreter` class that can process requests that are received from the clients to determine the appropriate response. Since the `storage` and `connections` instances are already globally present in the Python program and not only in a main class, these instances can be directly referred to, used, and altered in all of the classes without having to pass them in as parameters.

- **listenToClient() method:** The Server class also contains a `listenToClient()` method that is called in an infinite while-loop until the server is closed, so that multiple clients can be accepted and listened to by the server. in the Java, C#, and Python components, a new thread is assigned to each client that is accepted by the server, so that all clients can be

listened to simultaneously. This process is done by first receiving an integer denoting the length of the request, followed by the BSON request and deserializing it. Then the request is processed using the Interpreter or Request class, and the resultant response is serialized into a BSON byte array before being sent back to the client preceded by the size of the response BSON byte array.

7.1.2 Typescript component

- **Constructor:** The Typescript component has a similar `ZendServer` class that takes the same constructor parameters as described above. However, there is an additional class variable `ws`**(: WebSocket.WebsSocket)** that is set using the `setWebSocket()` method. This variable is required in order to be able to process the request that is being received from the other components.

- **listen() method:** the Typescript component cannot start multiple threads to handle multiple clients as Typescript is a single-threaded language. Hence, we handle multiple clients by making the `listen()` method return a `Promise<void>`. This method creates a server using the `net.createServer()` method, that contains two lambda functions:
  `handleReceivedRequest()` to process the request received over the connection and send the appropriate response, and `processData()` to read the request.

  When data is received over the connection, the `on("data")` event is triggered, which calls `processData()` on the buffer received. This method first determines whether the data received is the length of the request or the actual request by checking if the `requestLength` variable is null and if the buffer received has a size larger than 4. If this is true, then the data is read and deserialized as an Integer using `readInt32BE()` and set to the **requestLength** variable. If the **requestLength variable** is not null then the request is read and the `handleReceivedRequest()` method is called on it.

  The `handleReceivedRequest()` method deserializes the Buffer, processes it using the `processRequestFromExternal()` method that generates the response, serializes it then sends it back over the network using the `socket.write()` method.

7.2 Client class

This class acts as a client that can subscribe to or connect to the server of another framework (or another program of the same programming language) so that they can exchange data and information.

7.2.1 Java, Python and C# components

- **Constructor:** The Client class constructor takes a `host`**(: string)** - the hostname of the server to connect to, `port`**(: int)** - the port number of the server to connect to, `local port`**(: int)** - the port on which the client is opened, `storage`**(: Zend.Storage)** - the global storage so that the program that has the client port open can sync with the listening server storage.

- **sendRequest() method:** The `sendRequest()` method is called when the client needs to send a `request` to the server and consequently receive a response. The method takes the request that is to be sent to the server as a parameter. First, the method checks if there is already a socket connection open with the server. If the **socket** is null then it creates a socket that is connected to the given server **host** and **port**. Then, the request is serialized to BSON and sent over the network, preceded by the length of the request. Knowing the request will trigger a response on the server side, the method receives the serialized response, deserializes it, and returns the response.

- **connect() method:** The `connect()` method is called to connect the client to the given server **host** and **port** taken in the constructor. To do this, the method first creates a

`Handshake Request` that is sent to the server using the `sendRequest()` method, and receives the appropriate `Handshake Response` from the server. Then, the client's `handle`**(: string)** is assigned to the connection handle received from the server, and the client proceeds to send a snapshot of the program's current `storage` instance, so that the server and client **storage**s can be synced. Once the server sends back its **storage snapshot**, the client updates the program's current storage with the values of the server's storage.

7.2.2 Typescript component

- **Constructor:** The Typescript component has the same constructor parameters as the other components but additionally assigns a new instance of the `PromiseSocket()` class from the **"promise-socket"** library to a class variable called `socket`, as well as a `socketOpen`**(: boolean)** value that denotes if the socket connection is open.

- **initializeSocket():** This method assigns a new `PromiseSocket()` instance to the **socket** class variable. and connects it to the given **host** and **port**. The **socketOpen** bool is set to true, indicating that the connection is open. When the "end" event of the socket is triggered (i.e. when the socket is closed), then the **socketOpen** bool is set to false.

- **sendRequest() method:** In the Typescript component, the `sendRequest()` method returns a `Promise` of the **response** as opposed to the resolved response, as the method is marked **async** due to its requirement of being synchronous in nature. "The async keyword allows functions to be defined in a sequential, synchronous manner (although the execution will be asynchronous)."[49] It uses the `await` keyword when connecting the socket to the specified host and port, when it is writing to the TCP connection, when it reads from the TCP connection, and when the socket is closed. The `await` keyword "suspends the execution until an asynchronous function return promise is fulfilled and unwraps the value from the Promise returned"[48] and thus "makes it easier to handle the asynchronous nature of code."[49]

  First the method checks if **socketOpen** is set to true, then serializes the request, records the length of the resultant byte array, then creates a Buffer out of the integer length and the request itself using the `Buffer.from()` method. First the length is written over the socket connection, followed by the request. Furthermore, the response is received from the connection using the `read()` method, deserialized, and returned from the method once the socket connection is closed using the `end()` method. When receiving the data, it is done in a lambda function `receiveAll()` (taking a `number` parameter denoting expected message length) containing a while loop that continues to receive data until all the packets are received (so that it handles packet-splitting and instances where all the data is not sent altogether). Each iteration checks that the length of the data buffer received is less than the expected length of the request, and if so, then the received data is concatenated to the buffer.

  If the **socketOpen** variable is set to false, then the `initializeSocket()` method is called to create a new connection. This helps re-set or retry a connection when the attempt times out on the other side of the connection, or if the connection is closed due to any error or reason on either sides.

- **connect() method:** This `async` method returns a `Promise<void>`. It creates a **Handshake Request**, sends it using the `sendRequest()` method, and awaits the **Handshake Response** returned by the method. Then, the **handle** class variable is assigned to the **handle** field in the returned response.

- **syncStorage() method:** This `async` method takes a `storage`(: Zend.Storage) parameter, creates a **Sync Storage Request**, sends it using the `sendRequest()` method, awaits the **Sync Storage Response** and adds all the values sent back in the response to the **storage**. This method also returns a `Promise<void>`.

7.3 Request or Interpreter class

The role of this class is to receive a request from the **Server** class, process it, and produce the appropriate response to send back to the **Server** class, which will in turn serialize the response and send it to the appropriate client.

7.3.1 Java and C# components

In the Java and C# components, we define a superclass `Request`, which has three subclasses, corresponding to the different types of requests that can be sent over the network: `HandshakeRequest`, `PutRequest` and `SyncStorageRequest`. The **Request class** has two main methods: a `getResponse()` method and a `processRequest()` method, and its constructor takes the serialized Request that is meant to be processed.

- **Constructor:** The public constructor takes a **Document** parameter in the Java component, and a **BsonDocument** in the C# component. These parameters denote the **request** being passed into the class to be processed. There is an additional empty constructor that is declared `protected` so that it can only be called from the sub-classes of **Request**. This is so that the sub-classes can call an empty `super()` constructor method in their own constructors without having to pass in a Document.
- **getResponse() method:** This method determines the type of request (i.e. a **Handshake Request, Put Request** or **Sync Storage Request**), then accordingly creates a sub-class instance of that type, passing in the required parameters from the given request, calls `processRequest()` on that instance, then at the end returns the developed `response`.
- **processRequest() method:** This method is an empty method intended to be overridden by each of the sub-classes of `Request`. Each sub-class has its own parameters and method of processing the request, as well as creating the corresponding response.
- **HandshakeRequest sub-class:** The `HandshakeRequest` sub-class constructor requires the **host(: string)** - client hostname, **pid(: int)** - process ID, and **localPort(: int)** - port that client is hosted on, sent by the client. When `processRequest()` is called on an instance of this class, it creates a unique connection `handle`**(: string)** and adds a new `Connection` to the `Connections` class, using the **handle** as the **connection** identifier, the parameters entered into the constructor and the current date and time. At this point, it sets the `Client` as **null** in the `Connection` instance. Then the method returns a `HandshakeResponse` containing the **connection handle**.
- **PutRequest sub-class:** The `PutRequest` sub-class constructor requires the `name`**(: string)** - key of the value to be put into the **storage** and the `value`**(: serialized Object)** - value associated to the key. When the `processRequest()` method is called on the instance of this class, it adds the given key and value to storage, then returns a `PutResponse` that indicates the success of the request.
- **SyncStorageRequest sub-class:** The `SyncStorageRequest` sub-class constructor takes a `storageSnapshot`**(: Map of string to serialized Object)** - the snapshot of the **storage** received from the client, deserialized into a **HashMap (in Java)** or **Dictionary (in C#)** type, mapping from string keys to serialized Object type values. When the `processRequest()` method is called on an instance of this class, it first adds all the key-value pairs for the snapshot into the program's own **storage** instance, and then sends back a `SyncStorageResponse` with the program's current storage snapshot.

7.3.2 Python component

The Python component calls this class an **Interpreter** class. The `Interpreter` class constructor only takes the **storage** and **connections** objects to be able to add key-value pairs or connections

to them. It has a single `interpret()` method that takes the request to be deserialized and processed as a parameter. Instead of having sub-classes containing a `processRequest()` method, the `interpret()` method handles all the processing by reading the required values from the request like a Python **dictionary** type, and having a **switch-case** on the different types of requests once the request-type is determined. The response is formed and returned depending on the **case** that the request fits into.

7.3.3 Typescript component

The Typescript component must handle requests from both: the Excel add-in and from external connections. Hence there are two `export` methods contained in a **interpreter.ts** file to handle the different types of requests:

- **processRequestFromExternal() method:** This method acts similarly to the `processRequest()` method in the Python component. However, it takes an additional `ws`**(: WebSocket.Websocket)** parameter along with the **storage(: Zend.Storage)**, **connections(: Zend.Connections)** and **request(: ZendRequest)** parameters. Specifying the types of the accepted parameters also make the Typescript component more type-safe compared to the Python component. The processing itself is similar and done using an if-else chain based on the **request_type** field of the **request** parameter. This method handles **Handshake Request**s, **Put Request**s, and **Sync Storage Request**s and returns the appropriate **response(: ZendResponse)**.
- **processRequestFromExcel() method:** This method handles **Excel Put Request**s and **Excel Get Request**s, and returns a **Promise** of an **ExcelResponse**. It accepts parameters for the **request**, **storage**, and **connections**. If the **request_type** field of the **request** parameter is `"EXCEL_PUT_REQUEST"` then the method puts the `name` and serialized `value` contained in the request, into the **storage**, forwards the same **Put Request** to all the **connections** and returns an **ExcelPutResponse** indicating the success of the request.

  Alternatively, if the **request_type** field of the **request** parameter is `"EXCEL_GET_REQUEST`, then the method gets the serialized **value** associated with the `name` field of the request, and returns an **ExcelGetResponse** containing that serialized **value**. If the key **name** does not exist in the storage, then the method returns an empty string.

7.4 Storage class

This class acts as the local storage for each instance of a framework, but is meant to be synced with the other connected frameworks. When an instance of the `Storage` class is created, it does not take any parameters into its constructor, but rather initializes an empty **Map** or **Dictionary** type, which maps from **string** keys to serialized **Object** values.

The `Storage` class contains different versions of two main methods: `put()` and `get()`. There are two polymorphic types of the `put()` method: one which takes a `name`**(: string)** and `value`**(: serialized object)** as parameters and one which takes a `name`**(: string)** and `value`**(: any object)** as parameters. The first type directly puts the key-value pair into the dictionary, while the other type first serializes the value using the `serialize()` method of the `Converters` class.

In the Typescript component, we rename the first type of `put()` method to `putSerialized()` as different methods with the same name are not allowed to have different implementations. The first type is called in the `processRequest()` method of the **PutRequest** class as the value sent over the network is already serialized and thus can be directly stored in the map. Meanwhile, the second type is called by the user through the main `Zend` class, so that they may add something to the **storage**, but the serialization of the object is abstracted away.

The other versions of the `put()` method are the `putAll()` method and the `putAllSerialized()` method. The `putAll()` method takes a **Map** of key(string) to value(Object) pairs as a parameter,

and adds all the key-value pairs to the storage dictionary after serializing each of the values. Moreover, the `putAllSerialized()` method takes a **Map** of key(string) to value(serialized Object) pairs as a parameter and adds all the key-value pairs to the storage dictionary directly without serializing any values (as they are already serialized).

The `get()` method takes a `name`**(: string)** parameter that searches the **dictionary** and returns the value that corresponds to that key/name, after deserializing it. Similarly, the `getSerialized()` method takes the same parameter but returns the value as it is, i.e. in its serialized form.

The other versions of the `get()` method are: `getAll()` which returns the entire storage **dictionary** after deserializing each of the values, so that it returns a **Map** from string keys to Object values, and `getAllSerialized()` which returns a copy of the **dictionary** as it is (i.e. a **Map** from string keys to serialized Object values).

7.5 Converters class

This class serializes and deserializes data that is to be sent over the network or stored in the global **storage**. When stored within the program, the object values of the **storage** are stored as dictionary JSON-type values, for user readability.

The `Converters` class comprises of two main `static` or `export` methods: `serialize()` and `deserialize()`. The Java and C# components also comprise of an additional method called `getTypeMarker()`.

The `serialize()` method takes a value of any datatype and returns the serialized version back. In Python, the **"serialized object"** is a **dict** object containing the **"value_type"** and actual **"value"**. In Java, we store the serialized Objects in a data structure called `org.bson.Document` which is discussed in more detail in the **"Serialization and Deserialization"** chapter. In Typescript, we have created `types` for each Typescript datatype that could be serialized, as listed below:

- A `number` is serialized into a **ZendFloat**
- A `string` is serialized into a **ZendString**
- A `list` is serialized into a **ZendList**
- A `Map` is serialized into a **ZendMap**

The **ZendInt** type also exists in the library to ensure that the Typescript component recognizes values that have been sent from Python, Java or C# components that are tagged as **"int"** or **"int32"**.

An example of how objects are serialized in the library: assume we called the `Converters.serialize()` method on an `ArrayList` of `Integers` like:

```
List<Integer> list = new Arraylist<>();
list.add(1) // list = {1}
list.add(2) // list = {1, 2}
```

Then, the value returned by the Java `serialize()` method would be an `org.bson.Document` that, when converted to **JSON**, looks like:

```
{
    value_type: "list",
    value: [ { value_type: "int",
               value: 1 },
             { value_type: "int",
               value: 2 } ]
}
```

The serialization is done in the method by first determining the type of Object that has been passed into the function as a parameter, then using a `switch-case` or a list of `if-else if` statements to determine how to serialize the value. If the value is a primitive type (i.e. an integer, float or string value, then the value is serialized into a single-valued JSON-like object:

```
{
  value_type: <string representing the datatype>
  value: <the value passed into the serialize() function>
}
```

However, if the value is a compound data structure (i.e. a list, array, Dictionary/Map or tuple), then the `serialize()` method is recursively called to serialize each individual element or key-value pair in the object. Lists, arrays and tuples are serialized into JSON-lists while Dictionaries/Map are serialized into JSON-like embedded objects. An example of list serialization is given above, and an example of Map serialization is as follows:

```
// let Map<string, Integer> in Java = {"hello": 1, "world": 2}
// serialization output:
{
    value_type: "dict"
    value: {
            "hello": {value_type: "int", value: 1},
            "world": {value_type: "int", value: 2}
           }
// note that the dict keys are not serialized and are strictly
// strings
}
```

We can also have nested Maps/Dictionaries and Lists as the values are recursively serialized in the method.

Additionally, as you can see in the above example, to ensure simplicity of serializing **"Map"** and **"Dictionary"** types across the frameworks, we only allow the datatypes to have keys that are explicitly **strings**. This is because, in the Python framework, we serialize the values into **dict** type objects containing the **"value_type"** and the **value** itself, and a Python **dict** cannot have a `Hashable` value as a `key` in a **dict** (because **dicts** are mutable in Python and a **key** in a **dict** must remain constant/immutable in order to ensure that the key maps to the same value consistently every time[41]). For example, we can have a **dict** that looks like:
'{"one" : 1, "dict_ex" : {"nested_dict_ex" : "seven"}}',
but we cannot have a **dict** that looks like: **'{"one" : 1, "dict_ex" : {7 : "seven"}}'**.

Furthermore, the `deserialize()` method takes as parameter a serialized object and returns just the regular object in the way that it must be inferred for that specific framework. For primitive types, this function just returns the value given in the **"value"** of the serialized object. For non-primitive/compound datatypes, we go through each individual element or key-value pairs in the structure and deserialize them into a List, Dictionary/Map, Numpy array, or tuple accordingly depending on the framework and its supported types.

The `getTypeMarker()` method is used for returning the string identifier for a certain datatype of an Object. It takes a value of any Object type as parameter, and has a switch case considering the various datatypes supported in the Java and C# components of **Zend**. For example, if the object parameter is an instance of an Integer, then the method will return the string **"int"**, indicating the value is an integer.

This kind of labeling when serializing the object is important so that the supported datatypes can be appropriately recognized and interpreted cross-platform appropriately. For example: if a value is serialized and labeled as a **"dict"**, then the Python framework will interpret the object as a `dict` type, while Java will have to interpret it as a `List<Object>` (i.e. a list of Objects). The datatypes that are supported in each framework are discussed in detail below.

7.5.1 Zend-supported datatypes

In the Java framework, there are five main datatypes that are recognized: Integers, Floats (i.e. decimal point numbers), Strings, HashMaps (of any mapping), ArrayLists (of any datatype). In Java, we check the type of the value by using the `getClass()` method within the `getTypeMarker()` method. The string returned by the function is dependent on the type of object passed in:

- **Integer** - returned type-marker string **"int"**.
- **Float** - returned type-marker string **"float"**.
- **String** - returned type-marker string **"string"**.
- **Hashmap** - returned type-marker string **"dict"**.
- **ArrayList** - returned type-marker string **"list"**.

The C# framework has the same type-markers and supported Objects as the Java framework. This component uses the `is` keyword to determine what type the value is. C# supports some additional datatypes (or similar datatypes with different names) as listed below:

- **int, long** or **int64** - returned type-marker string **"int"**.
- **int32** - returned type-marker string **"int32"**.
- **float** or **double** - returned type-marker string **"float"**.
- **string** - returned type-marker string **"string"**.
- **IDictionary** - returned type-marker string **"dict"**.
- **IList** - returned type-marker string **"list"**.

In the Python framework, we use the `isinstance()` method to determine the datatype of the object. There are more Python datatypes that are supported than in Java , C# or Typescript. The different supported types and the strings returned by the `converters.py` file are listed below:

- NumPy integer (**np.int32**) - returned type-marker string **"int32"**.
- Numpy float (**np.float64**) - returned type-marker string **"float64"**.
- NumPy array (**np.ndarray**) - returned type-marker string **"array"**.
- Integer (**int**) - returned type-marker string **"int"**.
- Float (**float**) - returned type-marker string **"float"**.
- String (**str**) - returned type-marker string **"string"**.
- List (**list**) - returned type-marker string **"list"**.
- Tuple (**tuple**) - returned type-marker string **"tuple"**.
- Dictionary (**dict**) - returned type-marker string **"dict"**.

When deserializing objects that exist in the Python type-marker strings, but not in the above list of Java type-marker strings, we convert them to alternate Objects in Java that most closely depict the Python Object. For example, Java does not have **tuples**, so any object serialized and marked as a **"tuple"** type is converted to a heterogeneous list. For simplicity, Objects marked at **"array"** are also converted to Java lists so that it is easier to make them heterogeneous. Objects marked **"int32"** and **"float64"** are implicitly converted to Java `Integers` and `Floats`.

In the Typescript framework, we use the `is<>()` function imported from the `typescript-is` library. There are four types supported in this component:

- **number** - returned type-marker string **"float"**.
- **String** - returned type-marker string **"string"**.
- **Map** - returned type-marker string **"dict"**.
- List **(any[ ])** - returned type-marker string **"list"**.

We must note here that Typescript does not have separate types for natural numbers and numbers including a decimal point (i.e. Integers and Floats/Double); hence to avoid any confusion, we declare and interpret any `number` type in Typescript as a decimal point number (i.e. **"float"**) cross-platform. Typescript hence interprets any **int, int32, float, float64** as a `number` value and any **array, list, tuple** as a Typescript `list`.

Another significant design choice that was made in this library was maintaining the property of Python to be able to create and store non-homogenous lists, dictionaries, arrays, and tuples. This is why when we deserialize such an Object in Java and C# (which can only contain homogenous compound datatypes), we deserialize a "list" as a **List** of **Objects** (i.e. `List<Object>`), a "dict" as a **HashMap** mapping from **String** to **Object** to maintain the non-homogeneity. The `Object` class in Java is a superclass of all other classes, may it be predefined or user-defined[2]. Similarly, the Typescript component allows these lists and Maps to contains `any` values. The `any` keyword denotes that the object/value can be of any datatype (i.e. it has a similar role to "Object" in Java).

7.6 Connection class

This class is used as a type that stores relevant data regarding any Client connections made to the server of a framework's server. The constructor of this class takes a:

- `host`(: string) - the host name of the connecting client
- `pid`(: Integer) - the process ID pertaining to the client connection
- `localPort`/`local_port`(: Integer) - the port that the client connects on
- `dateTime`/`date_time`(: Date type) - the date and time that the connection was made
- `client`(: Zend.Client) - the actual client (i.e. an instance of the Client class) that is connecting to the server

This class does not have any particular methods or functions apart from **getter** methods for the parameters passed into the constructor, and one setter method for the **client** parameter. It acts more similarly to an `enum`.

7.7 Connections class

This class stores a Map **"connections"** from **strings** to `Connection`'s. The string is the unique Connection "handle", which identifies the Connection to the server.

The `Connections` class has two main functions: `add()` and `sendToAll()`.

- **add() method:** This function takes parameters `handle`**(: string)** - the UUID for recognizing the connection in the Map, `connection`**(: Connection)** - the actual connection being made to the server, and adds a string-Connection key-value pair to the Map.
- **sendToAll() method:** This function takes a **request** as a parameter and sends it to all the Clients connected to the framework's server. It does this by iterating (or mapping) over all the entries in the **"connections"** map and calling the `sendRequest()` method on the **client**

of each **connection** by passing in the request. If the **client** is `null`, then we create a new instance of the **Client** class using the information contained in the **Connection** instance, and the string **handle**.

7.8 Proxy

This is a file only present within the Typescript component (**proxy.ts)** and contains functions that allow external components (Java and Python programs) to connect to the proxy by opening a TCP server, and communicate with the Excel component (through the activated add-in). This file contains the following global variables:

- **storage(: Zend.Storage)** - This is the central global storage that remains up to date with all the components connected to the proxy.
- **connections(: Zend.Connections)** - This instance keeps track of all the **Connection**s connected to the proxy server.
- **server(: Zend.Server)** - This is the proxy server that runs on a user-given specified port and connects to external clients over TCP connections.
- **serverPort(: number)** - This is the port number on which the server runs.
- **wsServer(: WebSocket.Server)** - This is the websocket server that the excel add-in connects to.

The main function that handles the communication between components is the `startWsServer()` method. This `async` function takes `wsServerPort`**(: number)** - the port number on which the secure WebSocket server must run, and the `host`**(: string)** - the hostname on which the server is being hosted (it is automatically assigned the value **"localhost"** if no value is passed) as parameters. The method first creates an HTTPS server using the `createServer()` method, and in turn, creates a "WebSocket.Server" server using the HTTPS server. This is assigned to the global variable **wsServer**. When a **"connection"** event occurs on the websocket server (i.e. when it receives a connection from the add-in), the method defines a lambda function to handle messages being received on the socket. This lambda function receives the message as an **ArrayBuffer**, deserializes, and casts it into a **ZendExcelRequest** and checks that the **request_type** field of the request is `"EXCEL_OPEN_PORT_REQUEST"`. If this condition is satisfied then the method calls the `openPort()` method to open a TCP server port to listen for connections from external frameworks. Then, an **Excel Open Port Response** is sent back to the add-in component. If the **request_type** field of the request is not an `EXCEL_OPEN_PORT_REQUEST`|, then the `processRequestFromExcel()` method is called on the request, and the appropriate response is sent back.

The `startWsServer()` handles connection with the add-in, but the `openPort()` and `_connect()` methods are used to open the TCP server for incoming connections from external clients (Python and Java), and for connecting to those clients in order to be able to send data respectively.

The `openPort()` method acts in the same way as described in the **Zend API** chapter, but rather than returning a sure string, it returns a **Promise** of a string. The `_connect()` method similarly returns a **Promise** of a **ZendClient**.

These functions are not global to the users and are only called within the library to communicate between the WebSocket client (Excel) and TCP clients (external) via their servers described or created in the proxy.

Chapter 8

Serialization and Deserialization

We previously discussed how the data was serialized to be stored within the **storage**s of each program or framework. This chapter discusses how the data is serialized and deserialized when it is being sent over and received from the socket connections made in the library across frameworks.

BSON stands for **Binary Javascript Object Notation**. The `BSON` library was used for serialization and deserialization in Python. The implemented code uses `BSON.encode()` to serialize any given object to BSON (in this case the serialized values with their tags as discussed in the previous sections, or an integer depicting the length of the serialized message) into a BSON binary string before writing it to the socket connection. Meanwhile, the `BSON.decode()` method was used to deserialize a BSON binary string message received over the network.

It is worth noting that the BSON library is cross platform as it is implemented in multiple languages, although some of these implementations are embedded within **MongoDB** as "MongoDB was the first large project to make use of BSON".[6]

The Java component of the library uses the `org.mongodb.bson` which is a standalone BSON library[6], in order to serialize and deserialize its objects. While the Python component initially serialized its storage values into a dict type containing the type marker along with the value, the Java component does this by maintaining a `org.bson.Document` objects for each serialized value. This object is a MongoDB data structure that "contains key/value fields in binary JSON (BSON) format"[7]. The **Document** class implements the `Map<String, Object>` class in Java[7], which allows for flexible addition, retrieval, and removal of key-value pairs in Java as one would have with a Dictionary in Python or another language. This object is then converted to a BSON byte array to be sent over the network using a custom function `toBytes()`[32]. Similarly, the `toDocument()`[32] method converts a BSON byte array received over the network to a **Document**.

The `toBytes()` method utilizes a `DocumentCodec` object **DOCUMENT_CODEC** to `encode()` the given document onto a `BsonBinaryWriter` instance **writer**, which uses a `BasicOutputBuffer` instance **buffer**.[32]

On the other hand, the `toDocument()` method wraps the bytes into a `BsonBinaryReader` instance **reader** using the `ByteBuffer.wrap()` method, then uses the **DOCUMENT_CODEC** variable to `decode()` the **reader** contents into a document.[32]

Furthermore, the Typescript component uses the `bson` library for serializing and deserializing purposes. It simply uses the `BSON.serialize()` method to serialize any object (in this case a **ZendType**) to a BSON byte array, and `BSON.deserialize()` to deserialize any BSON byte array to an appropriate **ZendType**.

Chapter 9

Evaluation

9.1 Choosing Excel

Excel was chosen to be integrated with other computational frameworks as is one of the most widely used spreadsheet software for storing and sharing supplementary data, as well as analysing data.

However, we must also consider the limitations and disadvantages of using Excel for the center of integration. A news article written in 2020 reported that when scientists inputted gene names into Excel, they were auto-corrected to dates (for example, MARCH1 - short for "Membrane Associated Ring-CH-Type Finger 1" - was auto-corrected to the date 1-Mar) [54]. This corruption of data led to the scientists having to rename some of the genes (like MARCH1 to MARCHF1) in order to work around the issue [37]. A research study conducted in 2016 concluded that approximately 20% of gene auto-correct errors were contained in related papers [37]. An article written about this issue in 2004 stated in its results: "The date conversions affect at least 30 gene names; the floating-point conversions affect at least 2,000 if Riken identifiers are included. These conversions are irreversible; the original gene names cannot be recovered."[59]

Additionally, in the business and finance sector: "2012, JP Morgan declared a loss of more than US$6 billion thanks to a series of trading blunders made possible by formula errors in its modeling spreadsheets."[37] In 2020, "a spreadsheet error at Public Health England led to the loss of data corresponding to around 15,000 positive COVID-19 cases."[37]

There also exist various other spreadsheet software that provide similar and some specific better functionality compared to Excel. These software may also be adopted by various fields and industries to overcome Excel's limitations. However, this project aims to provide a wider contribution as Excel is still widely used and accepted. Some of the software that users may choose to use instead of Excel are discussed below.

1. **Google sheets:** This software has a better user-friendly interface,collaborative features and a free option for users who don't require intensive statistical or data analysis tools.[42]

2. **Numbers:** This software, despite having less functions than Excel, can provide better or more aesthetic graphic visuals.[33]

3. **LibreOffice:** This software is a free an open-source alternative to Excel that can handle many of the same spreadsheet tasks.[45]

4. **Zoho Sheet:** This software provides collaborative features, an analytic assistant powered by artificial intelligence, and is a good pocket-friendly alternative to Excel's provided functionality.[14]

9.2 Limitations of the Library

This section discusses minorities in the library that could be improved in the future, or alternatively cannot be easily maneuvered but must be addressed.

- The "dict" type values can only contain a string key for its values.
- This library does not support the 'complex' datatype in Python as it is complicated to recognize cross-platform.
- The Java component supports Hashmaps, Arraylists and other primitive types, and does not support user-defined classes or types.
- Servers cannot be opened on any and all machines, which would make the connection between components impossible using this add-in.
- A user must ensure they have the permission to enable add-ins in Excel, and must deal with trust-certificates when using the add-in.
- The library is currently tested on the same machine (rather than across a network).

Overall, the project has met its base objectives of integrating Excel with Python and Java and being able to pass marshaled objects between the components. Despite the limitations, it can be used vastly in the scope of expanding Excel's tools and functionality, and methods of utilizing its data.

Chapter 10

Conclusion

10.1 Project Plan

The project first began with research and familiarising with the sydx library, followed by structuring the API and deciding functionalities and the protocol for the library and communication between the components. Then, the project shifted aim to the actual implementation of the library, along with choices to make in the architecture or design of each component, the type-safety of the data and the library code itself, and debugging of the code.
An approximate timeline of the deliverables set during the project process is as follows:

1. **October - December:** Familiarise with the working of the sydx library; run tests to observe Python-Python program communication. Research other similar libraries and their functionalities; write the interim report.
2. **January - February:** Decide the API and functionality for the library (i.e. the user experience). Decide protocol for sending and receiving data. Start adding snippets of design choices to the final report. Research MongoDB and understand how to use it for serialization and deserialization using BSON.
3. **March - April:** Implement the library based on previous design choices. Try different architectures if required to test the best implementations. Implement the Java component and test its communication with Python.
4. **May:** Implement the Excel component. Refactor the library and implement any leftover functionality or fixes to bugs. Progress the final report. Decide the working of the demo for the final presentation. Explore an alternative way of coding Excel add-in using typescript and switch to typescript and using Visual Studio Code IDE.
5. **June:** Complete the WebSocket implementation in Typescript along with proxy. Debug the code and modify the report in accordance with the changes in the code. Prepare for the presentation and demo.

Although the original project deliverable timetable was designed to have the library and its complete implementations completed by the end of May, the unexpected challenge of finding a way to communicate with Excel using regular sockets (through a Proxy and WebSockets) extended this soft deadline.

10.2 Key Insights

- Although VSTO add-ins are comparatively customizable and can be user-specific, it is more practical to build an Office Web Add-in for convenience to a larger audience and for a better user interface (due to the task-pane and its customizability).

- Web Add-ins have comparatively user-friendly implementation and support for debugging (namely the Safari Web Inspector) compared to VSTO add-ins.
- Maintaining type-safety in Python is imperative when integrating applications cross-platform.
- Utilizing the HTTP or HTTPS protocol may make transference of data over the network easier and more reliable (and secure) compared to TCP connections, as the HTTP headers would make the requirement of sending the length of the request before payload redundant.

10.3 Future Work

- Make the library database-backed instead of using the "Storage" class. This can be accomplished by making more extensive use of the MongoDB library that has already been integrated into being used by the library for serialization.
- Integrate more languages and frameworks with Excel using similar architecture (for example: Rust, C++, etc.)
- Support more predefined datatypes to be sent across the platforms, as well as user-defined types
- Implement a model that updates values live in Excel and other platforms, if a storage value in Excel is changed
- Expand the library to work and send requests across a network as opposed to on a local machine or local network

Apart from the above modifications that could be made to the library, it could also be noted that if the project was re-built, it could use WebSockets for connecting the different frameworks throughout. This would allow the project to discard the proxy (thus reducing overhead) and increase the safety of the connections between Excel and the other components, as they would run over the HTTPS protocol which is more secure compared to the TCP protocol.

Alternatively, the project could be built to be more reliable in terms of sending and receiving requests across components, to ensure all components remain synced. This could be done by considering a different view on the project: building it as a distributed system where the different components are treated as nodes, rather than a centralized system where the proxy acts as the central server and main point of contact.[58]

10.4 Ethical and Legal Discussion

As this project simply builds a library that integrates multiple frameworks, there are no obvious ethical or legal issues. However, despite the legal neutrality of the library itself, one cannot immediately predict the actions of users with malicious intent. For example, such a project could be used in a malicious application that scrapes sensitive data from a source and is stored or transferred between programs for quantitative analysis to keep track of progress or data attained.

Furthermore, since the library sends BSON-serialized data through sockets, if the network is intercepted, it may compromise confidential data that belongs to and is being stored by the user in the program, or being sent over the network.

Additionally, another issue that must be considered is the possibility that a user may incorrectly input data into the system, which will be assumed correct by other users who have access to the data; this may lead to much larger errors depending upon the impact of the data that the users choose to store. On the other hand, if the error is done by Excel (for example: the gen-date errors discussed previously, or rounding errors), this could also adversely affect users of the library.

Apart from the considerations above, this library does not store sensitive user data that could be used to identify users (unless voluntarily stored by the user in the Storage - only during the execution of the program), does not involve elements that may cause harm to the environment, humans or animals and does not have a direct application or usage that could be exploited for malevolent, or criminal abuse. Additionally, the project does not use or produce software that may involve copyright licensing implications, or produce any information that could have negative impacts on human rights standards or have data protection and other legal implications.

Given below is a more detailed checklist of the common possible ethical considerations when creating and analyzing a project:

1. Does your project involve human participants? No
2. Does your project involve personal data collection and/or processing? No
3. Does it involve the collection and/or processing of sensitive personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)? No
4. Does it involve processing of genetic information? No
5. Does it involve tracking or observation of participants? It should be noted that this issue is not limited to surveillance or localization data. It also applies to Wan data such as IP address, MACs, cookies etc. No
6. Does your project involve further processing of previously collected personal data (secondary use)? For example Does your project involve merging existing data sets? No
7. Does your project involve animals? No
8. Does your project involve developing countries? No
9. If your project involves low and/or lower-middle income countries, are any benefit-sharing actions planned? No
10. Could the situation in the country put the individuals taking part in the project at risk? No
11. Does your project involve the use of elements that may cause harm to the environment, animals or plants? No
12. Does your project involve the use of elements that may cause harm to humans, including project staff? No
13. Does your project have the potential for military applications? No

14. Does your project have an exclusive civilian application focus? No

15. Will your project use or produce goods or information that will require export licenses in accordance with legislation on dual use items? No

16. Does your project affect current standards in military ethics – e.g., global ban on weapons of mass destruction, issues of proportionality, discrimination of combatants and accountability in drone and autonomous robotics developments, incendiary or laser weapons? No

17. Does your project have the potential for malevolent/criminal/terrorist abuse? No

18. Does your project involve information on/or the use of biological, chemical, nuclear/radiological-security sensitive materials and explosives, and means of their delivery? No

19. Does your project involve the development of technologies or the creation of information that could have severe negative impacts on human rights standards (e.g. privacy, stigmatization, discrimination), if misapplied? No

20. Does your project have the potential for terrorist or criminal abuse e.g. infrastructural vulnerability studies, cybersecurity-related project? No

21. Will your project use or produce software for which there are copyright licensing implications? No

22. Will your project use or produce goods or information for which there are data protection or other legal implications? No

23. Are there any other ethics issues that should be taken into consideration? No

Bibliography

[1] [Multiple contributors - Microsoft Documentation]. *VSTO add-in developer's guide to Office Web Add-ins*. URL: https://learn.microsoft.com/en-us/office/dev/add-ins/overview/learning-path-transition. (accessed: 17.6.23).

[2] Unknown (Tutorialspoint). *Why Object class is the super class for all classes in Java?* URL: https://www.tutorialspoint.com/why-object-class-is-the-super-class-for-all-classes-in-java#:~:text=Object%5C%20class%5C%20is%5C%20the%5C%20root,the%5C%20subclasses%5C%20from%5C%20Object%5C%20class.. (accessed: 5.6.23).

[3] [Internet]. *Apache POI - the Java API for Microsoft Documents*. URL: https://poi.apache.org/. (accessed: 13.01.2023)(Last Published: 09/17/2022).

[4] [Internet]. *Debug Office Add-ins on a Mac*. URL: https://learn.microsoft.com/en-us/office/dev/add-ins/testing/debug-office-add-ins-on-ipad-and-mac. (accessed: 20.6.23).

[5] [Internet]. *IANA Service Name and Transport Protocol Port Number Registry*. URL: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml. (accessed: 22.05.2023).

[6] [Internet]. *Implementations of BSON*. URL: https://bsonspec.org/implementations.html. (accessed: 17.6.23).

[7] [Internet]. *MongoDB: Documents*. URL: https://www.mongodb.com/docs/drivers/java/sync/v4.3/fundamentals/data-formats/documents/. (accessed: 17.6.23).

[8] [Internet]. *Node.js v20.3.0 Documentation*. URL: https://nodejs.org/api/net.html#event-connection. (accessed: 15.6.23).

[9] [Internet]. *Runtimes in Office Add-ins*. URL: https://learn.microsoft.com/en-us/office/dev/add-ins/testing/runtimes. (accessed: 20.6.23).

[10] [Internet]. *The WebSocket API (WebSockets)*. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. (accessed: 6.6.23).

[11] [Internet]. *What Is a Socket?* URL: https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html. (accessed: 14.12.2022).

[12] [Internet]. *Why use Python when we already have Excel?* URL: https://www.icaew.com/learning-and-development/academy/specialist-qualifications/data-analytics-certificate/why-use-python-when-we-already-have-excel. (accessed: 22.01.2023).

[13] [Internet]. *xlwings documentation*. URL: https://docs.xlwings.org/en/stable/quickstart.html. (accessed: 9.12.2022).

[14] [Internet]. *ZohoSheet: The Best Excel Alternative on the Cloud*. URL: https://www.zoho.com/sheet/excel-alternative.html. (accessed: 19.6.23).

[15] Computer Hope [unknown]. *Server*. URL: https://www.computerhope.com/jargon/s/server.htm. (accessed: 12.12.2022).

[16] Cole Arendt Adrian Dragulescu. *Read, Write, Format Excel 2007 and Excel 97/2000/XP/2003 Files*. URL: https://github.com/colearendt/xlsx. (accessed: 14.10.2022, https://cran.r-project.org/web/packages/xlsx/xlsx.pdf).

[17] Clarence J M Tauro; N Ganesan; Saumya Mishra; Anupama Bhagwat. "Object Serialization: A Study of Techniques of Implementing Binary Serialization in C++, Java and .NET". In: *International Journal of Computer Applications (0975 – 8887)* 45.6 (2012), pp. 25–29.

[18] Prof. Paul Bilokon. *sydx library*. URL: https://github.com/thalesians/sydx/. (accessed: 7.11.2022).

[19] Prof. Paul Bilokon. *thalesians/sydx*. URL: https://github.com/thalesians/sydx. (accessed: 7.11.2022).

[20] [Internet - Multiple contributors]. *Debug Office Add-ins on a Mac*. URL: https://learn.microsoft.com/en-us/office/dev/add-ins/testing/debug-office-add-ins-on-ipad-and-mac. (accessed: 10.6.23).

[21] [Internet - IBM Documentation]. *Advantages of Java*. URL: https://www.ibm.com/docs/en/ssw_aix_71/performance/advantages_java.html. (accessed: 19.6.23).

[22] Charlie Clark Eric Gazoni. *openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files*. URL: https://openpyxl.readthedocs.io/en/stable/. (accessed: 24.11.2022).

[23] Behrouz A. Forouzan. *TCP/IP Protocol Suite Fourth Edition*. McGraw-Hill Higher Education, 2010. ISBN: 978-0-07-337604-2.

[24] Alexander S. Gillis. *UUID (Universal Unique Identifier)*. URL: https://www.techtarget.com/searchapparchitecture/definition/UUID-Universal-Unique-Identifier#:~:text=A%5C%20UUID%5C%20(Universal%5C%20Unique%5C%20Identifier,UUID%5C%20generated%5C%20until%5C%20A.D.%5C%203400.. (accessed: 5.6.23).

[25] Alberto Gimeno. *Node.js multithreading: Worker threads and why they matter*. URL: https://blog.logrocket.com/node-js-multithreading-worker-threads-why-they-matter/. (accessed: 15.6.23).

[26] [Internet - multiple GitHub contributors]. *Receive and handle data with custom functions*. URL: https://learn.microsoft.com/en-us/office/dev/add-ins/excel/custom-functions-web-reqs. (accessed: 6.6.23).

[27] Parikshit Hooda. *Comparison – Centralized, Decentralized and Distributed Systems*. URL: https://www.geeksforgeeks.org/comparison-centralized-decentralized-and-distributed-systems/. (accessed: 15.6.23).

[28] Chet Hosmer. *Python Passive Network Mapping: P2NMAP*. Syngress, 2015. ISBN: 978-0-12-802721-9.

[29] J. Hunt. *Advanced Guide to Python 3 Programming. Undergraduate Topics in Computer Science*. Springer, Cham, 2019.

[30] [Internet - JAVASCRIPT.INFO]. *WebSocket*. URL: https://javascript.info/websocket. (accessed: 14.6.23).

[31] [geeksforgeeks user jt9999709701]. *Why Node.js is a single threaded language?* URL: https://www.imperialcollegeunion.org/shop/csp/computing/docsoc-summer-party. (accessed: 15.6.23).

[32] [GitHub user KooBoo]. *Just a static utility class to convert BSON Documents to byte-array and vice versa*. URL: https://gist.github.com/Koboo/ebd7c6802101e1a941ef31baca04113d. (accessed: 13.4.23).

[33] Kasper Langmann. *Numbers vs Excel: When to Use Excel When Numbers Is Better*. URL: https://spreadsheeto.com/numbers-vs-excel/. (accessed: 19.6.23).

[34] Alexandre Bergel; Damien Cassou; Stéphane Ducasse; Jannik Laval. *TCP Server*. URL: https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_Languages/Book%5C%3A_Deep_into_Pharo_(Bergel_Cassou_Ducasse_and_Laval)/03%5C%3A_Sockets/3.03%5C%3A_TCP_Server. (accessed: 14.12.2022)(Last updated: Jul 26, 2021).

[35] Andrew Lombardi. *WebSocket: Lightweight Client-Server Communications*. 1st ed. O' Reilly, 2015.

[36] Michelle Lundberg. *MICROSOFT OFFICE 365 INTEGRATIONS: THE COMPETITIVE DIFFERENCE BETWEEN COM/VSTO ADD-INS AND OFFICE ADD-INS*. URL: https://www.netdocuments.com/blog/microsoft-office-365-integrations-the-competitive-difference-between-com-vsto-add-ins-and-office-add-ins. (accessed: 17.6.23).

[37] Mandhri Abeysooriya Mark Ziemann. *Excel autocorrect errors still plague genetic research, raising concerns over scientific rigour*. URL: https://theconversation.com/excel-autocorrect-errors-still-plague-genetic-research-raising-concerns-over-scientific-rigour-166554. (accessed: 30.5.23).

[38] Henri Parviainen. *Javascript Event Loop Explained*. URL: https://www.webdevolution.com/blog/Javascript-Event-Loop-Explained. (accessed: 15.6.23).

[39] Tharinda Dilshan Piyadasa. *Centralized vs Distributed Systems in a nutshell*. URL: https://dev.to/tharindadilshan/centralized-vs-distributed-systems-in-a-nutshell-48c7. (accessed: 17.6.23).

[40] Eric Rosenburg. "The Importance of Excel in Business". In: *Investopedia* (last updated - 27.11.2022), p. 1.

[41] Aly Sivji. *Python Dictionaries Require Hashable Keys*. URL: https://alysivji.github.io/quick-hit-hashable-dict-keys.html#:~:text=When%5C%20we%5C%20use%5C%20a%5C%20key,dictionaries%5C%20require%5C%20hashable%5C%20dict%5C%20keys.. (accessed: 4.6.23).

[42] Luke Strauss. *Google Sheets vs. Excel: Which is right for you? [2023]*. URL: https://zapier.com/blog/google-sheets-vs-excel/. (accessed: 19.6.23).

[43] Panya MD†; Singhatanadgige; Weerasak MD Tanavalee Chotetawan MD*; Luksanapruksa. "Limitations of Using Microsoft Excel Version 2016 (MS Excel 2016) for Statistical Analysis for Medical Research." In: *Clinical Spine Surgery* 29.5 (June 2016. | DOI: 10.1097/BSD.0000000000000382), pp. 203–204.

[44] [iFour Team]. *Office add-in development: VSTO Add-ins vs JavaScript API*. URL: https://www.ifourtechnolab.com/blog/office-add-in-development-vsto-add-ins-vs-javascript-api. (accessed: 13.6.23).

[45] David Trounce. *LibreOffice vs Microsoft Office – Which Is The Best For You?* URL: https://www.online-tech-tips.com/software-reviews/libreoffice-vs-microsoft-office-which-is-best-for-you/. (accessed: 19.6.23).

[46] Unknown. *Distributed VS centralized networks*. URL: https://icommunity.io/en/centralized-vs-distributed-networks/. (accessed: 17.6.23).

[47] Unknown. *Proxy servers and tunneling*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Proxy_servers_and_tunneling. (accessed: 12.6.23).

[48] Unknown. *TypeScript 1.7*. URL: https://www.typescriptlang.org/docs/handbook/release-notes/typescript-1-7.html#:~:text=Asynchronous%5C%20functions%5C%20are%5C%20prefixed%5C%20with,value%5C%20from%5C%20the%5C%20Promise%5C%20returned.. (accessed: 13.6.23).

[49] unknown. *Introduction to async/await in TypeScript*. URL: https://www.atatus.com/blog/introduction-to-async-await-in-typescript/#:~:text=The%5C%20async%5C%20keyword%5C%20allows%5C%20functions,more%5C%20like%5C%20writing%5C%20synchronous%5C%20code.. (accessed: 1.6.23).

[50] unkown. *Client-Server*. URL: https://www.heavy.ai/technical-glossary/client-server. (accessed: 12.12.2022).

[51] unkown. *JSON and BSON*. URL: https://www.mongodb.com/json-and-bson#:~:text=BSON%5C%27s%5C%20binary%5C%20structure%5C%20encodes%5C%20type,been%5C%20missing%5C%20some%5C%20valuable%5C%20support.f. (accessed: 6.01.2023).

[52] unkown. *Working with Microsoft Excel in Java*. URL: https://www.baeldung.com/java-microsoft-excel. (accessed: 8.01.2023)(Last modified: December 2, 2022).

[53] John V. *Forward Proxy vs. Reverse Proxy Servers*. URL: https://www.jscape.com/blog/forward-proxy-vs-reverse-proxy. (accessed: 12.6.23).

[54] James Vincent. *Scientists rename human genes to stop Microsoft Excel from misreading them as dates*. URL: https://theverge.com/2020/8/6/21355674/human-genes-rename-microsoft-excel-misreading-dates. (accessed: 30.5.23).

[55] Roberto Moro Visconti. "How to Prepare a Business Plan with Excel". In: *SSRN* (last revised - 2019), p. 1.

[56] Iveta Vistorskyte. *Reverse Proxy vs. Forward Proxy: The Differences*. URL: https://oxylabs.io/blog/reverse-proxy-vs-forward-proxy. (accessed: 12.6.23).

[57] Geoffrey Litt; Seth Thompson; John Whittaker. *Improving performance of schemaless document storage in PostgreSQL using BSON*. URL: https://www.geoffreylitt.com/resources/Postgres-BSON.pdf. (accessed: 6.01.2023).

[58] Maleeha Yasvi. *Synchronization in Distributed Systems*. URL: https://www.geeksforgeeks.org/synchronization-in-distributed-systems/. (accessed: 15.6.23).

[59] Riss J. Zeeberg B.R., Kane, and D.W. "Mistaken Identifiers: Gene name errors can be introduced inadvertently when using Excel in bioinformatics". In: *BMC Bioinformatics* 5.80 (2004), pp. 330–335.

Appendix A

sydx/Excel/Sydx.cs

```
1  using System;
2  using System.Collections.Generic;
3  using System.Net;
4  using System.Net.Sockets;
5  using System.Text;
6  using System.Threading;
7  using ExcelDna.Integration;
8
9  namespace Sydx
10 {
11     public class Server
12     {
13         public class StateObject
14         {
15             // Client socket
16             public Socket workSocket = null;
17             // Size of receive buffer
18             public const int BufferSize = 1024;
19             // Receive buffer
20             public byte[] buffer = new byte[BufferSize];
21             // Received data string
22             public StringBuilder sb = new StringBuilder();
23         }
24
25         private int port;
26
27         private Storage storage;
28
29         private ManualResetEvent allDone = new ManualResetEvent(
       false);
30
31         public Server(int port, Storage storage)
32         {
33             this.port = port;
34             this.storage = storage;
35         }
36
37         public void Run()
38         {
39             Socket socket = new Socket(AddressFamily.InterNetwork,
       SocketType.Stream, ProtocolType.Tcp);
40             EndPoint localEndPoint = new IPEndPoint(IPAddress.Any,
```

```
this.port);
41              socket.Bind(localEndPoint);
42              socket.Listen(100);
43              while (true)
44              {
45                  // Set the event to nonsignaled state
46                  allDone.Reset();
47
48                  socket.BeginAccept(new AsyncCallback(AcceptCallback
    ), socket);
49
50                  // Wait until a connection is made before
    continuing
51                  allDone.WaitOne();
52              }
53          }
54
55          public void AcceptCallback(IAsyncResult ar)
56          {
57              // Signal the thread to continue
58              allDone.Set();
59
60              Socket socket = (Socket)ar.AsyncState;
61              Socket handler = socket.EndAccept(ar);
62
63              this.storage.Put("connected", 357.0);
64
65              StateObject state = new StateObject();
66              state.workSocket = handler;
67              handler.BeginReceive(state.buffer, 0, StateObject.
    BufferSize, 0,
68                  new AsyncCallback(ReadCallback), state);
69          }
70
71          public void ReadCallback(IAsyncResult ar)
72          {
73              String content = String.Empty;
74
75              StateObject state = (StateObject)ar.AsyncState;
76              Socket handler = state.workSocket;
77
78              int bytesRead = handler.EndReceive(ar);
79
80              if (bytesRead > 0)
81              {
82                  // There might be more data, so store the data
    received so far
83                  state.sb.Append(Encoding.ASCII.GetString(state.
    buffer, 0, bytesRead));
84
85                  // Check for end-of-file tag. If it is not there,
    read more data
86                  content = state.sb.ToString();
87
88                  if (content.IndexOf("<EOF>") > -1)
89                  {
90                      // All the data has been read from the client
91                      this.storage.Put("message", content);
```

```
92
93                      Send(handler, "Thank you.");
94                  }
95                  else
96                  {
97                      // Not all data received. Get more
98                      handler.BeginReceive(state.buffer, 0,
    StateObject.BufferSize, 0,
99                          new AsyncCallback(ReadCallback), state);
100                 }
101             }
102         }
103
104         private void Send(Socket handler, String data)
105         {
106             byte[] byteData = Encoding.ASCII.GetBytes(data);
107
108             handler.BeginSend(byteData, 0, byteData.Length, 0, new
    AsyncCallback(SendCallback), handler);
109         }
110
111         private void SendCallback(IAsyncResult ar)
112         {
113             try
114             {
115                 // Retrieve the socket from the state object
116                 Socket handler = (Socket)ar.AsyncState;
117
118                 // Complete sending the data to the remote device
119                 int bytesSent = handler.EndSend(ar);
120
121                 handler.Shutdown(SocketShutdown.Both);
122                 handler.Close();
123             }
124             catch (Exception e)
125             {
126                 // TODO Log exception
127             }
128         }
129     }
130
131     public class Storage
132     {
133         private static readonly Dictionary<string, object> dict =
    new Dictionary<string, object>();
134
135         public void Put(string name, object value)
136         {
137             Monitor.Enter(dict);
138             if (!dict.ContainsKey(name))
139             {
140                 dict.Add(name, value);
141             }
142             else
143             {
144                 dict[name] = value;
145             }
146             Monitor.Exit(dict);
```

```
147         }
148
149         public object Get(string name)
150         {
151             object value = null;
152             Monitor.Enter(dict);
153             if (dict.ContainsKey(name))
154             {
155                 value = dict[name];
156             }
157             Monitor.Exit(dict);
158             return value;
159         }
160
161     }
162
163     public static class ExcelFunctions
164     {
165         private static readonly Storage storage = new Storage();
166
167         private static Boolean portOpen = false;
168
169         [ExcelFunction(Description = "Opens a port for interaction
    with other Sydx systems")]
170         public static string S_Port(
171             [ExcelArgument(Name = "port", Description = "port
    number, e.g. 4782")] int port)
172         {
173             if (!portOpen)
174             {
175                 Server server = new Server(port, storage);
176                 Thread serverThread = new Thread(server.Run);
177                 serverThread.Start();
178                 portOpen = true;
179                 return "`success";
180             }
181             else
182             {
183                 return "`failure(reason='port already open')";
184             }
185         }
186
187         [ExcelFunction(Description = "Returns the version of the
    Integral AddIn as a string, e.g. 1.0.0")]
188         public static string S_Version()
189         {
190             return "1.0.0";
191         }
192
193         [ExcelFunction(Description = "Puts an object into the
    Integral dictionary")]
194         public static string S_Put(string name, object value)
195         {
196             storage.Put(name, value);
197             return "`" + name + "(type=" + value.GetType().ToString
    () + ")";
198         }
199
```

```
200         [ExcelFunction(Description = "Gets an object from the
    Integral dictionary")]
201         public static object S_Get(string name, object value)
202         {
203             return storage.Get(name);
204         }
205 
206         [ExcelFunction(Description = "Greets the user by name")]
207         public static string IntegralHello(string name)
208         {
209             return "Hello " + name;
210         }
211 
212         [ExcelFunction(Description = "An identity function: returns
     its argument")]
213         public static object S_Identity(object arg)
214         {
215             return arg;
216         }
217 
218         [ExcelFunction(Description = "Describes the value passed to
     the function")]
219         public static string S_Describe(object arg)
220         {
221             if (arg is double)
222                 return "Double: " + (double)arg;
223             else if (arg is string)
224                 return "String: " + (string)arg;
225             else if (arg is bool)
226                 return "Boolean: " + (bool)arg;
227             else if (arg is ExcelError)
228                 return "ExcelError: " + arg.ToString();
229             else if (arg is object[,])
230                 // The object array returned here may contain a
    mixture of different types,
231                 // reflecting the different cell contents
232                 return string.Format("Array[{0},{1}]", ((object[,])
    arg).GetLength(0), ((object[,])arg).GetLength(1));
233             else if (arg is ExcelMissing)
234                 return "<<Missing>>"; // Would have been System.
    Reflection.Missing in previous versions of ExcelDna
235             else if (arg is ExcelEmpty)
236                 return "<<Empty>>"; // Would have been null
237             else
238                 return "!? Unheard Of ?!";
239         }
240     }
241 }
```

Appendix B

sydx/Python/src/main/sydx.py

```
import copy
import datetime as dt
import errno
import importlib
import inspect
import json
import logging
import logging.config
import os
import socket
import string
import threading
import uuid

class SydxException(Exception):
    pass

class Connection(object):
    def __init__(self, host, pid, local_port, connection_datetime,
    client):
        self.__host = host
        self.__pid = pid
        self.__local_port = local_port
        self.__connection_datetime = connection_datetime
        self.__client = client

    @property
    def host(self):
        return self.__host

    @property
    def pid(self):
        return self.__pid

    @property
    def local_port(self):
        return self.__local_port

    @property
    def connection_datetime(self):
        return self.__connection_datetime

```

```
42      @property
43      def client(self):
44          return self.__client
45
46      @client.setter
47      def client(self, client):
48          self.__client = client
49
50  class Connections(object):
51      def __init__(self):
52          self.__lock = threading.Lock()
53          self.__connections = {}
54
55      def add(self, handle, connection):
56          with self.__lock:
57              self.__connections[handle] = connection
58
59      def send_to_all(self, request):
60          with self.__lock:
61              for handle, connection in self.__connections.items():
62                  if connection.client is None:
63                      connection.client = _connect(connection.host,
    connection.local_port)
64                  connection.client.send_request(request)
65
66  class Storage(object):
67      def __init__(self):
68          self.__lock = threading.Lock()
69          self.__dict = {}
70
71      def __contains__(self, name):
72          logger = logging.getLogger()
73          logger.debug('Acquiring storage lock')
74          with self.__lock:
75              result = name in self.__dict
76              logger.debug('Releasing storage lock')
77          return result
78
79      def put(self, name, value):
80          logger = logging.getLogger()
81          logger.debug('Acquiring storage lock')
82          with self.__lock:
83              self.__dict[name] = copy.copy(value)
84              logger.debug('Releasing storage lock')
85
86      def get(self, name):
87          value = None
88          logger = logging.getLogger()
89          logger.debug('Acquiring storage lock')
90          self.__lock.acquire()
91          if name in self.__dict:
92              value = copy.copy(self.__dict[name])
93          logger.debug('Releasing storage lock')
94          self.__lock.release()
95          return value
96
97      def get_all(self):
98          values = {}
```

```
99          logger = logging.getLogger()
100         logger.debug('Acquiring storage lock')
101         self.__lock.acquire()
102         for name, value in self.__dict.items():
103             values[name] = copy.copy(value)
104         logger.debug('Releasing storage lock')
105         self.__lock.release()
106         return values
107
108     def put_all(self, values):
109         logger = logging.getLogger()
110         logger.debug('Acquiring storage lock')
111         self.__lock.acquire()
112         for name, value in values.items():
113             self.__dict[name] = copy.copy(value)
114         logger.debug('Releasing storage lock')
115         self.__lock.release()
116
117 class Interpreter(object):
118     def __init__(self, storage, connections):
119         self.__storage = storage
120         self.__connections = connections
121
122     def interpret(self, request_json):
123         request_obj = json.loads(request_json)
124         if request_obj['request_type'] == 'HANDSHAKE_REQUEST':
125             handle = str(uuid.uuid1())
126             host = request_obj['host']
127             pid = request_obj['pid']
128             local_port = request_obj['local_port']
129             self.__connections.add(handle, Connection(host, pid,
    local_port, dt.datetime.utcnow(), client=None))
130             response_obj = {
131                 'response_type': 'HANDSHAKE_RESPONSE',
132                 'connection_handle': handle
133             }
134         elif request_obj['request_type'] == 'SYNC_STORAGE_REQUEST':
135             their_storage_snapshot = request_obj['storage_snapshot'
    ]
136             self.__storage.put_all(their_storage_snapshot)
137             our_storage_snapshot = self.__storage.get_all()
138             response_obj = {
139                 'response_type': 'SYNC_STORAGE_RESPONSE',
140                 'storage_snapshot': our_storage_snapshot
141             }
142         elif request_obj['request_type'] == 'PUT_REQUEST':
143             name = request_obj['name']
144             value = request_obj['value']
145             self.__storage.put(name, value)
146             response_obj = {
147                 'response_type': 'PUT_RESPONSE',
148                 'result': 'SUCCESS'
149             }
150         else: raise SydxException('Unfamiliar request_type')
151         response_json = json.dumps(response_obj)
152         return response_json
153
154 class Server(object):
```

```
155     def __init__(self, host, port, interpreter):
156         self.__host = host
157         self.__port = port
158         self.__interpreter = interpreter
159         self.__socket = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
160         self.__socket.setsockopt(socket.SOL_SOCKET, socket.
    SO_REUSEADDR, 1)
161         self.__socket.bind((self.__host, self.__port))
162 
163     def listen(self):
164         logger = logging.getLogger()
165         self.__socket.listen(5)
166         while threading.main_thread().is_alive():
167             logger.info('Waiting for a client to connect to the
    socket')
168             client, address = self.__socket.accept()
169             client.settimeout(60)
170             logger.info('Client connected. Kicking off a new thread
     to service this client')
171             threading.Thread(target=self.listenToClient, args=(
    client, address)).start()
172         self.__socket.close()
173 
174     def receive_line(self, client):
175         size = 1024
176         message = ''
177         while True:
178             chunk = client.recv(size)
179             if not chunk: break
180             message += chunk.decode('utf-8')
181             if message.find('\n') != -1: break
182         return message
183 
184     def listenToClient(self, client, address):
185         request = self.receive_line(client)
186         request = request.rstrip()
187         response = self.__interpreter.interpret(request)
188         response = response.rstrip() + '\n'
189         client.send(response.encode())
190 
191 class Client(object):
192     def __init__(self, host, port, local_port, storage):
193         self.__host = host
194         self.__port = port
195         self.__local_port = local_port
196         self.__storage = storage
197         self.__handle = None
198 
199     @property
200     def handle(self):
201         return self.__handle
202 
203     def receive_line(self, client):
204         size = 1024
205         message = ''
206         while True:
207             chunk = client.recv(size)
```

```
208             if not chunk: break
209             message += chunk.decode('utf-8')
210             if message.find('\n') != -1: break
211         return message
212 
213     def send_request(self, request):
214         request_json = json.dumps(request)
215         logger = logging.getLogger()
216         self.__socket = socket.socket(socket.AF_INET, socket.
    SOCK_STREAM)
217         logger.debug('Connecting on socket %s' % str(self.__socket)
    )
218         self.__socket.connect((self.__host, self.__port))
219         logger.debug('Sending request: %s' % request_json)
220         self.__socket.send((request_json + '\n').encode())
221         logger.debug('Receiving response')
222         response_json = self.receive_line(self.__socket)
223         logger.debug('Received response: %s' % response_json)
224         response = json.loads(response_json)
225         self.__socket.close()
226         return response
227 
228     def connect(self):
229         request = {
230             'request_type': 'HANDSHAKE_REQUEST',
231             'host': socket.gethostname(),
232             'pid': os.getpid(),
233             'local_port': self.__local_port
234         }
235         response = self.send_request(request)
236         self.__handle = response['connection_handle']
237         values = self.__storage.get_all()
238         request = {
239             'request_type': 'SYNC_STORAGE_REQUEST',
240             'storage_snapshot': values
241         }
242         response = self.send_request(request)
243         their_storage_snapshot = response['storage_snapshot']
244         self.__storage.put_all(their_storage_snapshot)
245 
246 __version__ = '1.0.0'
247 
248 def __init_logging():
249     module_dir = os.path.dirname(os.path.abspath(__file__))
250     logging_config_file_name = 'sydx-logging.cfg'
251     default_config_file_path = os.path.join(module_dir,
    logging_config_file_name)
252     config_file_path = os.getenv('SYDX_PYTHON_LOGGING_CONFIG',
    default_config_file_path)
253     if not os.path.exists(config_file_path):
254         config_file_path = os.path.join(module_dir, '..', '..', '
    config', logging_config_file_name)
255     if os.path.exists(config_file_path):
256         logging.config.fileConfig(config_file_path)
257     else:
258         logging.basicConfig()
259 
260 def __load_converters():
```

```
261     module_names = os.getenv('SYDX_PYTHON_CONVERTER_MODULES', '
    converters').split(',')
262     all_from_json_object_converters, all_to_json_object_converters
    = [], []
263     for module_name in module_names:
264         module = importlib.import_module(module_name)
265         from_json_object_converters = [o[1] for o in inspect.
    getmembers(module) if inspect.isfunction(o[1]) and o[0].endswith
    ('_from_json_object')]
266         to_json_object_converters = [o[1] for o in inspect.
    getmembers(module) if inspect.isfunction(o[1]) and o[0].endswith
    ('_to_json_object')]
267         all_from_json_object_converters.extend(
    from_json_object_converters)
268         all_to_json_object_converters.extend(
    to_json_object_converters)
269     return all_from_json_object_converters,
    all_to_json_object_converters
270
271 __init_logging()
272
273 logging.getLogger().info('Initialising Sydx version %s' %
    __version__)
274
275 __from_json_object_converters, __to_json_object_converters =
    __load_converters()
276 logging.getLogger().info('Loaded %d deserialisers and %d
    serialisers' % (len(__from_json_object_converters), len(
    __to_json_object_converters)))
277
278 __global_lock = threading.Lock()
279
280 __server_port = None
281
282 __storage = Storage()
283
284 __connections = Connections()
285
286 def version():
287     return __version__
288
289 def port(port):
290     global __server_port
291     if __server_port is not None:
292         raise SydxException('Port already open')
293     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
294     interpreter = Interpreter(__storage, __connections)
295     server = Server('', port, interpreter)
296     threading.Thread(target=server.listen, args=()).start()
297     __server_port = port
298
299 def _connect(host, port):
300     client = Client(host, port, __server_port, __storage)
301     client.connect()
302     return client
303
304 def connect(host, port):
305     if __server_port is None:
```

```
306             raise SydxException('Must open port before connecting')
307         return _connect(host, port).handle
308 
309 def server_port():
310         return __server_port
311 
312 def connections():
313         pass
314 
315 def __serialise(value):
316         global __to_json_object_converters
317         logger = logging.getLogger()
318         json_object = None
319         success = False
320         for converter in __to_json_object_converters:
321             logger.debug('Trying converter %s' % converter)
322             try:
323                 json_object = converter(value)
324             except Exception as e:
325                 logger.debug('The converter has failed')
326                 logger.debug(e)
327                 continue
328             success = True
329             break
330         if not success:
331             raise SydxException('There is no converter that can
    serialise the object')
332         return json_object
333 
334 def put(name, value):
335         logger = logging.getLogger()
336         logger.debug("Attempting to serialise the object named '%s'" %
    name)
337         json_object = __serialise(value)
338         logger.debug('Serialisation result: %s' % str(json_object))
339         request = {
340                 'request_type': 'PUT_REQUEST',
341                 'name': name,
342                 'value': json_object
343             }
344         __connections.send_to_all(request)
345         __storage.put(name, json_object)
346 
347 def __deserialise(json_object):
348         global __from_json_object_converters
349         logger = logging.getLogger()
350         obj = None
351         success = False
352         for converter in __from_json_object_converters:
353             logger.debug('Trying converter %s' % converter)
354             try:
355                 obj = converter(json_object)
356             except Exception as e:
357                 logger.debug('The converter has failed')
358                 logger.debug(e)
359                 continue
360             success = True
361             break
```

```
362     if not success:
363         raise SydxException('There is no converter that can
    deserialise the object')
364     return obj
365
366 def get(name):
367     logger = logging.getLogger()
368     json_object = __storage.get(name)
369     logger.debug("Attempting to deserialise the object named '%s'"
    % name)
370     obj = __deserialise(json_object)
371     logger.debug('Deserialisation result: %s' % str(obj))
372     return obj
373
374 def describe(obj):
375     pass
376
377 def show(obj):
378     pass
379
380 def represent(obj):
381     pass
382
383 def evaluate(handle, code):
384     pass
385
386 def save(file_path):
387     pass
388
389 def load(file_path):
390     pass
```

Appendix C

sydx/Python/src/main/converters.py

```
import logging

import numpy as np

import sydx

def numpy_numeral_to_json_object(obj):
    if isinstance(obj, np.int32):
        return {
            'type': 'numpy.int32',
            'value': int(obj)
        }
    elif isinstance(obj, np.float64):
        return {
            'type': 'numpy.float64',
            'value': float(obj)
        }
    else:
        raise ValueError('Unable to serialise an object')

def numpy_numeral_from_json_object(json_object):
    if isinstance(json_object, dict):
        if 'type' in json_object and json_object['type'] == 'numpy.
    int32':
            return np.int32(json_object['value'])
        elif 'type' in json_object and json_object['type'] == '
    numpy.float64':
            return np.float64(json_object['value'])
        else:
            raise ValueError('Unable to deserialise an object')
    else:
        raise ValueError('Unable to deserialise an object')

def __numpy_array_to_list(arr):
    result = []
    for x in arr:
        if isinstance(x, np.ndarray):
            result.append(__numpy_array_to_list(x))
        else:
            result.append(sydx.__serialise(x))
    return result

```

```
def __list_to_numpy_array(lst):
    result = []
    for x in lst:
        if isinstance(x, list):
            result.append(__list_to_numpy_array(x))
        else:
            result.append(sydx.__deserialise(x))
    return np.array(result)

def numpy_array_to_json_object(obj):
    if not isinstance(obj, np.ndarray):
        raise ValueError('Unable to serialise an object')
    return {
        'type': 'numpy.ndarray',
        'values': __numpy_array_to_list(obj)
    }

def numpy_array_from_json_object(json_object):
    if isinstance(json_object, dict):
        if 'type' in json_object and json_object['type'] == 'numpy.ndarray':
            return __list_to_numpy_array(json_object['values'])
        else:
            raise ValueError('Unable to deserialise an object')
    raise ValueError('Unable to deserialise an object')

def plain_old_data_to_json_object(obj):
    logger = logging.getLogger()
    logger.debug('Attempting to serialise a plain old data object')
    if isinstance(obj, (int, float, complex)):
        return obj
    elif isinstance(obj, str):
        return obj
    elif isinstance(obj, list):
        return [sydx.__serialise(x) for x in obj]
    elif isinstance(obj, tuple):
        return {
            'type': 'tuple',
            'values': list(obj)
        }
    elif isinstance(obj, dict):
        return {
            'type': 'dict',
            'values': {sydx.__serialise(x): sydx.__serialise(y) for x, y in obj.items()}
        }
    elif obj is None:
        return None
    else:
        logger.debug('This object does not seem to be a plain old data object')
        raise ValueError('Unable to serialise an object')

def plain_old_data_from_json_object(json_object):
    logger = logging.getLogger()
    logger.debug('Attempting to deserialise a plain old data object')
    if isinstance(json_object, (int, float, complex)):
```

```
95          return json_object
96      elif isinstance(json_object, str):
97          return json_object
98      elif isinstance(json_object, list):
99          return [sydx.__deserialise(x) for x in json_object]
100     elif isinstance(json_object, tuple):
101         return tuple([sydx.__deserialise(x) for x in json_object])
102     elif isinstance(json_object, dict):
103         if 'type' in json_object:
104             if json_object['type'] == 'dict':
105                 return {sydx.__deserialise(x): sydx.__deserialise(y
    ) for x, y in json_object['values'].items()}
106             elif json_object['type'] == 'tuple':
107                 return tuple([sydx.__deserialise(x) for x in
    json_object['values']])
108         else:
109             logger.debug('This JSON object does not seem to
    represent a plain old data object')
110             raise ValueError('Unable to deserialise an object')
111     elif json_object is None:
112         return None
113     else:
114         logger.debug('This JSON object does not seem to represent a
     plain old data object')
115         raise ValueError('Unable to deserialise an object')
```