Using Web Services for Customised Data Entry

A thesis submitted in partial fulfilment of the requirements for the Degree of Master of Applied Computing

at Lincoln University

by Yanbo Deng

Lincoln University

2007
Abstract of a thesis submitted in partial fulfilment of the requirements for the Degree of Master of Applied Computing

Using Web Services for Customised Data Entry

by Yanbo Deng

Scientific databases often need to be accessed from a variety of different applications. There are usually many ways to retrieve and analyse data already in a database. However, it can be more difficult to enter data which has originally been stored in different sources and formats (e.g. spreadsheets, other databases, statistical packages).

This project focuses on investigating a generic, platform independent way to simplify the loading of databases. The proposed solution uses Web services as middleware to supply essential data management functionality such as inserting, updating, deleting and retrieval of data. These functions allow application developers to easily customise their own data entry applications according to local data sources, formats and user requirements.

We implemented a Web service to support loading data to the Germinate database at the New Zealand Institute of Crop & Food Research (CFR). We also provided language specific client toolkits to help developers invoke the Web service. The toolkits allow applications to be easily customised for different platforms. In addition, we developed sample applications to help end users load data from their project data sources via the Web service.

The Web service approach was evaluated through user and developer trials. The feedback from the developer trial showed that using Web services as middleware is a useful approach to allow developers and competent end users to customise data entry with minimal effort. More importantly, the customised client applications enabled end users to load data directly from their project spreadsheets and databases. It significantly reduced the effort required for exporting or transforming the source data.

**Keywords:** Data integration; Data management; Data loading; Web services
Acknowledgements

Firstly, I would like to acknowledge my supervisors, Clare Churcher and Walt Abell, for their support and guidance. They made significant efforts to provide detailed comments and suggestions for this research. Without them, I could not have completed this thesis.

Secondly, I would like to express my thanks to my external advisor, John McCallum (from Crop and Food Research), for his contributions to the research. Also, I would like to acknowledge the support from a FRST sub-contract.

Thirdly, I would like to acknowledge all those who helped me for this research. I wish to thank Keith Armstrong, Gabriel Lincourt and Gail Timmerman-Vaughan for their assistance with the user trials, and Flo Weingartner for his technical assistance. Special thanks to Caitriona Cameron and Karen Johnson for improving my thesis writing and research presentation skills.

Finally, I would like to thank my mother for her encouragement and support.
Table of Contents

Abstract...........................................................................................................................................i
Acknowledgements ........................................................................................................................ii
Table of Contents........................................................................................................................iii
List of Figures.....................................................................................................................................vi
List of Tables.....................................................................................................................................viii

Chapter 1  Introduction ............................................................................................................... 1
  1.1 Case study.......................................................................................................................... 1
  1.2 Aim and scope.................................................................................................................... 2
  1.3 Structure of the thesis......................................................................................................... 2

Chapter 2  Background................................................................................................................. 4
  2.1 Data integration ................................................................................................................ 4
  2.2 Data retrieval .................................................................................................................... 5
  2.3 Data loading issues.......................................................................................................... 5
    2.3.1 Different data structures and schemas................................................................. 6
    2.3.2 Different subsets....................................................................................................... 6
    2.3.3 Different data representation styles......................................................................... 7
    2.3.4 Different data management systems....................................................................... 7
    2.3.5 Different distributed data source locations............................................................ 7
  2.4 Applications for loading data........................................................................................... 8
    2.4.1 Understanding data................................................................................................. 8
    2.4.2 Gathering data......................................................................................................... 8
    2.4.3 Manipulating data................................................................................................... 8
    2.4.4 Variety of user requirements................................................................................... 9
    2.4.5 Changes to the database implementations............................................................ 9
    2.4.6 Data maintenance................................................................................................. 9
  2.5 Current data loading techniques and approaches......................................................... 10
    2.5.1 An example of data loading.................................................................................... 10
    2.5.2 Staging tables approach......................................................................................... 13
    2.5.3 Data warehousing ETL tools................................................................................ 14
    2.5.4 Customized data loading applications................................................................... 17
  2.6 Summary and guidelines................................................................................................... 18
    2.6.1 End users............................................................................................................... 19
    2.6.2 Developers............................................................................................................. 19

Chapter 3  Case Study - Germinate database............................................................................. 20
  3.1 Germinate overview......................................................................................................... 20
  3.2 Data browsing................................................................................................................ 23
  3.3 Data loading and maintenance....................................................................................... 25
5.3.3 Encapsulating database implementation ......................................... 70
5.3.4 Error handing .................................................................................. 71
5.3.5 Performance ...................................................................................... 72
5.3.6 Reliability ......................................................................................... 73
5.3.7 Security ............................................................................................. 74
5.4 Summary .............................................................................................. 75

Chapter 6 Conclusion .................................................................................. 76
6.1 Guidelines ............................................................................................ 76
6.1.1 End users ......................................................................................... 77
6.1.2 Developers ...................................................................................... 78
6.2 Additional guidelines ............................................................................ 78
6.3 Final remarks ....................................................................................... 79

References .................................................................................................. 81

Appendices ................................................................................................. 84
Appendix A Instructions for using the Germinate VBA toolkit (updated version).... 84
Appendix B Evaluation form for users .............................................................. 97
Appendix C Evaluation form for developers ................................................... 100
List of Figures

Figure 2-1 Separate and merge different data subsets ........................................................... 6
Figure 2-2 Loading data from a spreadsheet to several tables .............................................. 11
Figure 2-3 ETL data flow ..................................................................................................... 16
Figure 2-4 Repetitive development ..................................................................................... 18
Figure 3-1 Simplified Germinate data module diagram ....................................................... 21
Figure 3-2 Relationship between the Accession table and lookup table ................................ 22
Figure 3-3 GDPC Web service and GDPC clients ................................................................. 23
Figure 3-4 GDPC Browser .................................................................................................. 24
Figure 3-5 Genotype visualization tool (GVT) ..................................................................... 24
Figure 3-6 Data loading flow .............................................................................................. 25
Figure 3-7 Exporting and converting data from multiple sources ...................................... 27
Figure 3-8 Rearranging and reformatting of data .............................................................. 28
Figure 3-9 Error messages from PL/SQL scripts ................................................................ 30
Figure 3-10 Typical error message from PhpPgAdmin ...................................................... 31
Figure 4-1 Middleware solution used by multiple applications ........................................... 34
Figure 4-2 Data access middleware ................................................................................... 36
Figure 4-3 Data access using a web service ........................................................................ 38
Figure 4-4 Java RMI versus SOAP .................................................................................... 39
Figure 4-5 Proposed architecture ....................................................................................... 41
Figure 4-6 Web service client toolkits ............................................................................... 42
Figure 4-7 Data loading workflow ..................................................................................... 43
Figure 4-8 Web service implementation architecture ......................................................... 47
Figure 4-9 Data flow through the system .......................................................................... 48
Figure 4-10 Data loading workflow ................................................................................... 50
Figure 4-11 Example of using Accession object .............................................................. 53
Figure 4-12 Example of using the toolkit initialisation procedure ...................................... 54
Figure 4-13 Example of error handling ............................................................................. 55
Figure 5-1 Typical Germinate data in Excel ..................................................................... 58
Figure 5-2 Excel application showing input data and response ........................................ 59
Figure 5-3 Data errors and response ................................................................................. 60
Figure 5-4 Result of resubmitting data ............................................................................. 62
Figure 5-5 Excel data for the developer trials ................................................................. 64
Figure 5-6 Typical code to call toolkit functions .............................................................. 66
List of Tables

Table 2-1 Store source data in a staging table ................................................................. 13
Table 3-1 Data stored in Germinate .................................................................................. 20
Table 4-1 International coding schema transformations .................................................. 44
Table 4-2 Date transformations ......................................................................................... 45
Table 4-3 Local data transformation ................................................................................. 46
Table 4-4 Result codes and messages .............................................................................. 52
Chapter 1 Introduction

Scientific databases often need to be accessed from a variety of different applications. Usually there are a number of ways to retrieve and analyse data provided by the databases, however, it is often more difficult to load original data from different sources. In many cases, the original data may be stored in different types of software and with a variety of formats. Therefore, users often have difficulty in loading data from different data sources into target databases.

The target database structure and data formats might be quite different to that of the source data, making it difficult to import data directly into the target database. To develop software to help load data into a large database, developers have to understand the often intricate database structure which might include triggers, sequences, database functions and referential integrity constraints. It can take a lot of time to become familiar with an intricate database structure. Also, the original data may need to be transformed into different formats (e.g. different formats for dates) in order to be loaded. Assisting users to reformat and rearrange their data to fit the target database can be a time-consuming and complicated process. For example, it might include several operations, such as reformatting data, eliminating duplicate records and validating input data.

The original data may be stored in a variety of sources, applications, and platforms, such as in text documents in a Pocket PC, Excel spreadsheets on Windows, on a database server on Linux or in statistical software packages on any platform. As a result, there is no easy way to provide a single data loading application which can satisfy different users’ needs. Data loading software that is suitable for one group of users, may be inappropriate for other users at different sites.

1.1 Case study

Germinate is a generic plant database developed by the Genetic Resources Community (Lee, 2005). Community users can download and install Germinate in different organisations to store and analysis plant data.

Crop and Food Research (CFR) has installed a copy of Germinate to manage onion and oat data. Currently Germinate provides data templates (in Excel and Access) and SQL scripts for loading data. CFR users find using these templates and scripts difficult. The main
problem is that each different source of data has to be exported and converted into the appropriate format to fit into the template. Users can not directly load data from their own data sources. Sometimes the data must traverse several systems and formats before being ready to import into Germinate.

1.2 Aim and scope
We investigate how to simplify loading data according to users’ local data sources, formats and requirements. The approach to customisation must be easy and not require developers to work directly with complex database management systems. This project aims to provide generic database functionality, which allows application developers to customise data loading applications based on local data sources with minimal effort. We propose a middleware solution for the general problem and implement it for the specific case of Germinate. In particular, we have designed and implemented Web service middleware for Germinate which can be easily accessed by any programming language and operating system over the network.

1.3 Structure of the thesis
Chapter 2 reviews the literature on data integration and summarises the difficulties of loading data from multiple data sources to a central database. It also examines common data loading approaches and points out the usefulness and limitations of each of these approaches. Chapter 2 also suggests guidelines for data loading applications and for tools to help develop such applications.

Chapter 3 describes the case study example: the Germinate database as used at CFR. It describes the problems of loading data from different sources into Germinate at CFR using the provided templates and SQL scripts.

Chapter 4 presents a proposed general solution to match the guidelines. It describes using Web services as middleware to supply database access functions for loading data from different data sources and applications. It introduces the concept of Web service client toolkits, which help application developers easily include the database functionality in different customised applications. It also describes the implementation of these ideas at CFR.

Chapter 5 evaluates the Web service approach. It describes trials for both application developers and end users. Developers assessed using a VBA toolkit to customise a
complete data loading application, and end users evaluated using the customised client applications to load data. Finally, conclusions are drawn in Chapter 6.
Chapter 2  Background

In this chapter, we will first introduce using large databases for data integration and data retrieval applications. Then in section 2.3 and 2.4 we will describe the difficulties of loading data into a large database from multiple heterogeneous data sources. After that, the limitations of current data loading techniques will be discussed in section 2.5. Finally, we will outline some guidelines for making data loading applications which are useful and efficient for both end users and developers.

2.1 Data integration

People often need to integrate data from multiple data sources, such as different databases, software applications, spreadsheets and text documents. For example, sales records might need to be collected from different stores for commercial analysis, or scientists might need to combine experimental results from different projects to support further study. To simplify data retrieval efforts, it may be necessary to build a central (large) database system or data warehouse to combine data from the different sources. This method can provide an easy way to manage and access information collected from multiple data sources (Zisman, 1995).

For scientific projects, some organisations develop large databases to integrate research results. These databases store a wide range of data that are assembled from various information sources or individual research projects. With a common database schema to link data from diverse sources and projects, it is possible to carry out queries across the different sets of data.

The benefit is that the large database provides an easier way for researchers to investigate and manage different types of data via a unified interface. Many organisations publish their databases on the internet to allow a wider community to access different type of data or different research results.

Some of these databases are implemented in open source database management systems, such as MySQL and PostgreSQL. As well as accessing common data users can also download and set up their own copy of the database to manage their local data.
2.2 Data retrieval
Large databases that integrate data from different sources can be accessed from a variety of applications depending on the users’ requirements. Scientific research communities in particular usually concentrate on different methods of data retrieval and analysis. For example, a plant genetic database can be connected to a variety of data browsing applications such as: applications for searching for particular plant sample details, visualisation applications to represent genetic experiment data, Geographical Information Systems to locate the regions of plant samples. These tools maximize the accessibility and usability of the data stored in the databases (Lee, 2005).

In order to satisfy specific data retrieval requirements, some data providers also supply Web services to help with data extraction. Web services allow data consumers to create their own client applications. That is to say, developers can build a data analysis application that can easily obtain source data from a public database via a Web service interface. This data retrieval approach is widely utilized on scientific research projects to address different issues, such as, exporting genotype experiment data (Casstevens, 2003), publishing Weather Information (Steiner, 2005) and providing a DNA sequence data service (NCBI, 2005). A data extracting Web service can also allow advanced users to retrieve data into their own customised applications.

2.3 Data loading issues
While there are a number of ways to retrieve and analyse data already in a database it is more difficult to upload data which has been stored in different applications and formats. Gathering existing data and transforming them into a central database can be more complex than extracting data from a database. Normally, research communities and organisations focus on how to use data, but only a few projects provide easy to use data loading and maintenance applications. The following sections will discuss some of the challenges and difficulties associated with data loading and maintenance.

Where we have a complex database which is designed for a large community of users there are often problems with data entry. The data model often reflects a particular community standard which makes the data contents understandable to everyone. However, different institutions and groups might use different terms and definitions for their local data. Also the original data may be stored by different types of local users in a number of ways. We
will consider the issues of loading such data into a target database from multiple data sources.

2.3.1 Different data structures and schemas
The target database’s structure or schema might be different from the original data structure/schema. For example, a user’s original data may be stored in single table, whereas the target database might have several tables that contain complex internal relationships such as lookup tables, database triggers and referential integrity constraints. As a result, the data is difficult to import directly into the target database.

2.3.2 Different subsets
Each existing data source may have been designed to satisfy a particular group of users, and may contain a number of types of information that might or might not be relevant for the integrated database. In these cases, only part of original data needs to be loaded into a table in the target database (such as data source 1 of Figure 2-1). Also, some datasets (from a single source) may need to be separated into two or more subsets to populate different tables in the target database (such as data source 2). In addition, some data sources may have overlapping data that needs to be rearranged and merged into one set of data in the database such as data source 3 and 4 in Figure 2-1. (Davidson, 1995)

![Figure 2-1 Separate and merge different data subsets](image_url)
2.3.3 Different data representation styles

There may be many different ways to represent similar data items. Therefore, after the necessary datasets have been extracted from each source, the data representation styles might need to be transformed to conform to the requirements of target databases. For example, currency values might be presented in N.Z. dollars, U.S. dollars or U.K. pounds on different systems, and it may be necessary to express them in a common currency in the integrated database. This can be done with a very simple calculated field based on one or more data sources. However, it can be difficult to transform data to consistent representation styles and conventions in some complex scientific databases. Even something as simple as a date can cause problems. Date data might be stored in different ways in each data source. For instance, MM/DD/YYYY date style may be used in data source 1, and DD/MM/YYYY style in data source 2, while the target database uses YYYY-MM-DDD. Some transformation of the data to preferred types and styles is usually required before loading data into a central database (Rosenthal, 1994).

2.3.4 Different data management systems

The original data may come from a variety of sources, applications, and platforms such as text documents in a pocket PC, Excel spreadsheets on a Windows OS, database tables on a Linux OS or statistical applications developed in different programming languages. This makes integrating the data into a central database difficult. A single form of data loading application or ad hoc SQL script can not satisfy all the different data loading requirements. The complexities of each user’s IT environment needs to be take into account by the data loading applications (Banerjee, 2002).

2.3.5 Different distributed data source locations

Existing data sources used to support particular projects and research may be in different sites. If designers wish to combine data from different geographic locations, then each data source needs to be accessible via a public network. The key issues here are data security and accessibility. For example, some data sources can not be shared over the Internet for security reasons. Therefore developers can not extract data from all distributed data sources over the network, and some original data must be shipped from data sources (Rosenthal, 1994).
2.4 Applications for loading data

The issues described in the previous section can make it very difficult for an end user to upload and maintain data in a centralised database. Often they need to pass the tasks on to a more knowledgeable database administrator or have a front end specially developed for them. In this section we will look at some of the issues in providing specialised loading and maintenance applications.

2.4.1 Understanding data

To create applications for loading data from different sources, developers have to understand both the target database structure and the data structure of each source. Developers often spend considerable time and effort in order to understand the data relationships between a target database and data sources (Rosenthal, 1994). The target database might include lots of tables, triggers, sequences, database functions and referential integrity constraints. Although, database providers supply entity relationship diagrams and data dictionaries to help users comprehend the data schema, it is still difficult to master a large schema in a short time. Developers may need to work though much documentation to understand the data and talk with domain experts to fully understand terms and attribute names.

2.4.2 Gathering data

We have mentioned that data may need to be extracted from different geographical locations. The difficulty is that the data sources might not be accessible over a network. Some scientists might just manage their data within spreadsheets on their PCs, and for security reasons they may not want to share the data on a network. That means an extract strategy can not be applied in such cases. Therefore individual data has to be uploaded manually (populated from sources) from the different distributed sources (Armstrong, 2006).

2.4.3 Manipulating data

The original data may need to be manipulated into appropriate formats and types in order to be loaded into the target database. Assisting users to validate and transform data is a critical and complicated process. A developer of a data loading application should provide some validation functions to check for invalid data, such as checking for duplicates or checking the original data for what might become potential referential integrity problems.
in the target database. The data validation functions will need to prevent invalid data being introduced into the target database. It may also be necessary to provide some transformation functions, such as converting original data types and representation styles to conform to the target database. These types of checks and transformations can be quite difficult. (Shaw, 2007)

2.4.4 Variety of user requirements

It is difficult to provide a single application that can satisfy different users’ data loading requirements. The original data sources and formats can be very different in each organisation and even for different research projects within an organisation. Data loading software which is suitable for one group of users may be inappropriate or cumbersome for other clients at different sites. Some research databases offer a single data loading method such as a spreadsheet data template with scripts and SQL commands. Users will have to export their original data into a format compatible with the template provided and then run the scripts to load the data into a target database. This approach is a “single satisfactory strategy” which ignores the requirements of loading data from other data storage forms such as database tables or applications (Davidson, 1995). This approach means each group of users will have to develop ways to get their data into the required format. As a result, repetitive development efforts are needed.

2.4.5 Changes to the database implementations

If the target database has major changes to structure or access, each data loading applications (at individual organisations) may need to be redeveloped to conform to the new version of the target database. This may occur if the database schema is redesigned to meet new demands. Moreover, if the database management system software is changed to a new product, such as from PostgreSQL to Oracle, the database provider may need to rewrite the stored database procedures and SQL data loading scripts.

2.4.6 Data maintenance

Simple data maintenance tools such as table editors are inadequate for managing a complex database, because these tools are suitable for inserting and maintaining data in individual tables. When users attempt to update what they consider to be a single record (such as depicted in Figure 2-2) they discover that it is actually distributed over several tables. A simple table editor is not a convenient way to update, because several tables must
be accessed to modify all relevant data. Some end users with advanced database skills might be able to write SQL scripts to ease the data modification operations, but many end users are not familiar with SQL techniques. Therefore, even simple correction of data can be quite difficult.

2.5 Current data loading techniques and approaches
There are number of techniques and approaches to assist users with loading data into a centralised database. Today, these mature techniques are widely applied in data loading projects, but they still have some limitations for communities of researchers wanting to load data from multiple data sources. In this section, we will present a simple data loading scenario, and then we will explain how three different data loading approaches could be applied to the scenario. After that, the limitations will be discussed for each approach.

2.5.1 An example of data loading
The following Figure 2-2 shows a very simple data loading scenario for data about plant samples. The source data is in the spreadsheet at the top of the Figure 2-2, and the target database includes 3 tables. The spreadsheet stores original data in a flat format which is typical of many research data sources. The spreadsheet stores 4 columns of data: sample code (string), species (string), collecting site (string) and collecting date information (date format with dd/mm/yyyy style). The target database implements a relational design that requires reference/lookup data (Species and Collecting Site information) to be stored in separate reference/lookup tables (Species and Site tables shown in Figure 2-2). Then, in the main Sample table there are foreign keys referencing the lookup tables. There are referential integrity constraints to ensure the Sample table can not have Species ID and Site ID values that are not present in the lookup tables. In addition, the Collecting date type in the target database is a date data type, but the recommended input style is (yyyy-mm-dd) as a string.
The process depicted in Figure 2-2 involves 5 steps to load the source data into the target database. Developers of applications to help with this process would need to address each step.

1. **Mapping data:**

Developers need to map data between the source data columns and the various target table columns (shown in Figure 2-2).

2. **Transforming data representation styles:**

Developers must create data transformation procedures to convert the original data into the required formats and styles. In this example, the date format in the spreadsheet needs to be transformed into a yyyy-mm-dd string format, because the original date format is entered as dd/mm/yyyy in the spreadsheet. There may also be other complex conversions required before loading to destination tables (e.g. different plant descriptor coding schemes might be used in the source and target database).
3. **Transforming lookup data into IDs:**

In many cases, the lookup data are already preloaded into the lookup tables in the target database. For example, the Species names and Site names are already in the Species table and the Site table in the target database. In this example, the Species and Collecting Site information are represented by foreign keys in the Sample table. The developers need to write a procedure to find the SpeciesID and SiteID corresponding to the original data, and load these ID values into the SpeciesID and SiteID columns in the Sample table. For instance, we should put 1 in the SpeciesID column of the sample table to reflect “oat” in the species column in the spreadsheet.

4. **Checking for invalid data:**

Before loading the data into target database table, the loading procedures should verify the data values. If any invalid data is detected, the procedures should provide understandable messages to inform the users so they can fix the errors. The following are some data checks which should be included for this example:

First, with data coming from several data sources there may be some duplicate records. Developers should supply checking methods to detect and handle any duplicate records.

Second, developers must check that the source data will not cause referential integrity problems. For example, if the Species name and Site name values (in data source columns) can not be found in the lookup tables of the target database, the procedure should provide messages to inform the user of the referential integrity errors.

Third, developers must check the format and values of the source data. E.g. invalid date data can not be converted into correct yyyy-mm-dd style. Therefore, the procedures should provide error messages to inform the user of any data formatting problems.

5. **Loading data:**

After the previous four steps, the data records need to be loaded into the plant sample table.
2.5.2 Staging tables approach

Developers can apply a staging tables approach. This allows data to be loaded from staging tables to target database tables using SQL scripts. This is a common approach which is used in different data integration systems (Seiner, 2000; Lee, 2005). The staging table is a (or several) temporary table which can store the entire original data. The data can then be migrated into the different destination tables. To improve SQL script capabilities, embedded SQL scripts and procedural languages are often combined together to process and load data from staging tables to proper tables. For example, a database procedural language, called PL/SQL, has been introduced into Oracle database system, as well as other DBMS such as PL/pgSQL for PostgreSQL and SQL PL for IBM DB2. (IBM, 2006; Oracle, 2006; PostgreSQL, 2006).

- **Applying the staging table approach**

  Developers can import all the data from different sources, and store them in a staging table (in the target database) where data is in a flat format. It is very easy to import different source data into a staging table, because there are not any data checks and validation rules for the staging table. Therefore, any data can be simply imported from different sources. The source data (in Figure 2-2) can be extracted from the spreadsheet, and stored in a staging table shown Table 2-1.

<table>
<thead>
<tr>
<th>plantSampleCode</th>
<th>Species</th>
<th>collectingSite</th>
<th>collectingDate</th>
<th>Other columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>plant 001</td>
<td>Oat</td>
<td>site 1</td>
<td>8/11/2001</td>
<td>...</td>
</tr>
<tr>
<td>plant 002</td>
<td>Oat</td>
<td>site 2</td>
<td>8/07/1999</td>
<td>...</td>
</tr>
<tr>
<td>plant 003</td>
<td>Onion</td>
<td>site 1</td>
<td>18/06/2006</td>
<td>...</td>
</tr>
<tr>
<td>plant 004</td>
<td>Onion</td>
<td>site 1</td>
<td>4/03/1982</td>
<td>...</td>
</tr>
<tr>
<td>plant 005</td>
<td>Onion</td>
<td>site 1</td>
<td>18/06/2006</td>
<td>...</td>
</tr>
<tr>
<td>plant 006</td>
<td>Onion</td>
<td>site 2</td>
<td>4/03/1982</td>
<td>...</td>
</tr>
<tr>
<td>plant 007</td>
<td>Oat</td>
<td>site 3</td>
<td>5/03/1982</td>
<td>...</td>
</tr>
</tbody>
</table>

After importing the data into the staging table from different sources, developers can program a database loading procedure with embedded SQL scripts that include several functions such as checking data, transforming data formats and types, transferring lookup data, and providing error messages. Eventually, the loading procedures can migrate the staging data into appropriate database tables. If any invalid data is detected, users have to correct the data values in the staging table.
**Limitations**

It is easier and quicker to program the procedures and SQL scripts for a staging table than to develop a complete data loading application. Therefore, some scientific databases offer procedures with embedded SQL scripts in order to help community users to load data. For example, community users extract original data and store them in a staging table. Then users can utilize the procedures to migrate records from the staging table into proper tables. Although the procedures can be developed quickly to provide data loading features, this approach has 2 major limitations for end users.

1. These scripts are only suitable for advanced users (e.g. database administrators) who have programming and SQL scripts experience. Novice end users may find it is difficult to run SQL scripts and understand and handle error messages. For example, some scientific databases are installed on open sources operating systems such as Linux or UNIX, and the scripts are commonly executed on the shell command line environment which can be daunting for end users. Also, if some invalid data is loaded, the SQL error messages may not be easily understood by end users.

2. Another problem is that designers expect users to extract data and import them into the staging tables. This can still be a difficult process as there is not a simple way to streamline data export from a number of very different sources to the staging tables. It results in repetitive and often manual data import processing.

**2.5.3 Data warehousing ETL tools**

Using ETL software (Extract Transform and Load) is another approach for data loading. ETL software can support database (data warehouse) architects to extract data from different sources, transform source data based on target database structure and rules, and load them into the target databases (Kimball, 2004). If data warehouse architects use ETL software to load data, they need to build an ETL workflow which consists of data source locations, target tables locations, and data transformation rules. Once the ETL workflow is set up, the source data can be automatically loaded into the target databases or data warehouses. In addition, there is a similar approach called scientific workflow systems (SWFs); SWFs allows scientists to use different data processing components/models to compute results based on raw input data, so it provides a flexible way to retrieve heterogeneous data in order to support the workflow (Altintas, 2006).
Applying ETL software for data loading

The following example illustrates how to load data from the Excel spreadsheet source (Figure 2-2) into the target database using an ETL tool called Microsoft SQL Server 2005 Integration Service (SSIS). SSIS supports a wide range of ETL functionality with a series of data processing components. Each component plays an important role in the overall ETL operation. For example, a data connection component assists designers to define a data source for data extraction, and a data destination component helps designers to route source data into target data tables. In SSIS development environments, designers can include different components in a data workflow panel to execute different processes, and then link the components together to achieve overall ETL goals (Figure 2-3).

First, ETL designers need to specify data file locations or database connections. The next step is to transform the source data into the appropriate representation or format to conform to the target database. SSIS provides the transformation components to help designers to convert data. For example, the date format will be converted into the right format, and the lookup data (Collecting Site and Species names) will be transformed into the appropriate foreign key values to be put in the plant sample table. Eventually, designers can set up data mappings between the source data columns and destination data columns using a data flow destination component. For example, the SampleCode column (source data) needs to be mapped into the SampleCode column in the Sample table. In addition, SSIS allows designers to handle error messages when errors occur during the processing. For example, messages could be stored in a text file as shown in Figure 2-3.
Figure 2-3 ETL data flow

- Limitations

There are some limitations for research community users using ETL tools to load data. The ETL operation is the most complex and time-consuming task in data warehousing projects (Kimball, 2004). Developing, implementing and managing ETL applications can consume a substantial amount of staff and computing resources and so it is not ideal for a small community of research users. The ETL tools are more often employed to automate loading data from commercial transaction systems which process large amounts of data, such as banks and supermarkets.

Although using ETL tools (such as SSIS) means designers do not need to write too many SQL scripts for transforming and loading data, designers still have to spend considerable time to understand the relationship between the source data and the destination database in order to implement the data transformation process.
Some ETL tools (e.g. SSIS) apply a data extraction strategy to get data from different sources. To do this, each data source must be shared over the network in order to be extracted with ETL tools. However, some data sources might not be able to be published via the network for security reasons. It is quite difficult to implement the data extraction principle to automatically import data from each source.

### 2.5.4 Customized data loading applications

A data loading application is often customized to carry out data loading according to specific users’ requirements. It could provide a user friendly data entry interface and useful error handling features, which allow users to enter data and upload it to the database. This is the best solution for the end user but it can be time consuming for the developer.

- **Example of a VBA data loading tool**

As part of our initial investigation of data loading applications, we developed an Excel VBA application prototype to help users load data from an Excel spreadsheet into a database. The application we developed included several functions, such as data checking, data transforming, and data loading functions. This approach was successful for bulk loading a large number of records into the database.

The user interface was based on the Excel spreadsheet interface, and it used the Microsoft ActiveX Data Object technique with the PostgreSQL ODBC driver to communicate with the database. This application allowed users to fill data into the excel spreadsheet and submit them to the database simply by pressing a button. In addition, if the application detected errors in the input data, it provided useful errors messages to help end users to locate and correct the invalid data.

- **Limitations**

Although a complete data loading application provides a friendly user interfaces to transform and load data, it was developed to load data from a single source (e.g. a Web form, a Windows form or a spreadsheet). However in an organisation, the data is likely to be stored in different places such as text files, spreadsheets, or databases. Users might require several different data loading applications to be developed for them.

The data loading functions we developed were all built using VBA. This means they cannot be reused for other applications or languages. For example, users might need to communicate with the database from a Java or PHP application running on different platforms (shown in Figure 2-4). The VBA loading data functions can not be utilized in the...
Java or PHP application. Therefore, developers have to rebuild all the fundamental database access code such as the inserting data function for each language and platform. Developers need a solution that will minimise these repetitive development efforts.

Figure 2-4 Repetitive development

In some organisations, there may not be sufficient in-house programming skills to undertake such development. It is difficult to provide complete data loading applications suitable for the different end users, because the applications need to provide a user interface, data base access functions, data transformation functions and error handling functions. Data loading to central databases is often not well supported in many research organisations.

2.6 Summary and guidelines

Loading data from heterogeneous data sources to a centralised database is the most complex and time consuming part of data integration projects. This is because the source data structures, types, formats and attribute representation styles may be quite different from the destination database. This means that end users can find it difficult to load their data to a central database without significant input from technical experts. Developers of data loading applications have to spend considerable time to understand the relationships between data sources and targets, and program data transformation functions, data loading functions, and user interfaces to facilitate data integration. Although there are some approaches such as staging tables with SQL scripts, ETL tools and customised
applications, they all have some limitations which reduce their usefulness for developers and end users especially in research organisations.

In order to facilitate the loading of data from several different sources into a central database it would be useful to develop some tools to help overcome the problems outlined in this chapter. Here we present some guidelines for tools to help develop data loading applications. We look at what the applications need to provide for the end user and what functionality could be useful for the developers of such applications.

### 2.6.1 End users

Data loading applications should:

- be able to help users load data directly from their local data sources.
- convert source data types and representation styles to match target database requirements.
- provide a user friendly interface.
- provide input data checking and pass useful errors messages back to the user as appropriate.
- be able to allow omission of non essential data and provide defaults if necessary for any missing data.

### 2.6.2 Developers

Developers need tools which:

- allow them to customise data loading applications for different data sources with minimal effort.
- insulate them from the details of the database as much as possible.
- have as small a learning curve as possible.

In the following chapter, we will describe Germinate which is a large database for storing plant data. We will describe the current data loading approach for Germinate and compare it with our guidelines.
Chapter 3  Case Study - Germinate database

3.1 Germinate overview

Germinate is an open source data integration system to store and maintain crop plant genetic resources and relevant information. The main goal of Germinate is to associate Genotype and Phenotype data for research. The Genetic resources community (Lee, 2005) has developed Germinate using the PostgreSQL database management system. Community users can download and install Germinate at different organisations to carry out similar research activities. The system stores different types of crop plant data such as Passport, Phenotype and Genotype data. A description of the data types are shown in Table 3-1.

Table 3-1 Data stored in Germinate

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passport data</td>
<td>Descriptive information of plant Accessions (plant samples), e.g. plant sample identification number, collecting location, collecting date. (Williams, 1984)</td>
</tr>
<tr>
<td>Phenotype data</td>
<td>Physical appearance and constitution, e.g. plant weights and length (Brenner, 2002)</td>
</tr>
<tr>
<td>Genotype data</td>
<td>Specific genetic makeup which is measured by genetic markers e.g. Markers data. (Lewontin, 2004)</td>
</tr>
</tbody>
</table>

Germinate is a very large genetic resources database which has 72 tables with intricate data relationships. The tables are grouped into four modules to store the plant data (the simplified data model is shown in Figure 3-1).

- The Passport module (left top part of Figure 3-1) is designed for storing crop plant accession descriptive information. Germinate requires Passport data to conform to the Multi-Crop Passport Descriptors Standard (FAO & IPGRI, 2001), this ensures that the plant data is represented in a widely recognised form.
- Genotype data and Phenotype data for plant Accessions are stored in the datasets module (right bottom part of Figure 3-1). This includes Markers data, Trait data and Micro-satellite data.
The integration module (right top part of Figure 3-1) associates Genotype data, Phenotype data and Passport data via Accessions. Figure 3-1 shows how the Datasets module and the Passport module are connected to the Accession table by the Data Integration module.

The last module is the information module (left bottom part of Figure 3-1), which stores general information about research institutions, countries and database users.

Figure 3-1 Simplified Germinate data module diagram

The Germinate data model links different types of data via plant Accessions (plant samples), so biological scientists can conveniently query the crop plant data with an integrated database system. The Passport data, Genotype data and Phenotype data are usually originally recorded in multiple databases or applications, and (if not using Germinate) users might need to access several data sources or applications to find information. If users employ Germinate to manage plant data, the different types of data are associated through plant Accessions. Then, the Germinate system can answer very complex questions. For example, biological scientists can find “all Accessions that both share a particular allele for a given marker and that were collected from a specific geographical region” (Lee, 2005). As a result, the Germinate system can help users to query their desired information efficiently. These questions might not be answered simply before using Germinate.
Many triggers and referential integrity constraints are applied in Germinate, and these rules prevent data entry errors. There are 17 triggers to check invalid input data. For example, the Germinate_Identity table is a linking table that connects the Accession table with other tables. There is a trigger (for checking the Germinate_Identity table) which is intended to make sure that the accession number must be unique in a particular domain, e.g. within an institution (the combination of Accession Number and Institution ID must be unique), in order to eliminate duplicate Accession records in Germinate.

Most tables have dependencies to other tables, so referential integrity constraints are used to make sure data records are valid between related tables. For instance, Institution ID is a primary key of the Institution table, and it is also a foreign key of the accession table (shown in Figure 3-2). Therefore, when users add a new Accession record to the Accession table, they must give an Institution code that is already present in the Institution table (all institution information has been preloaded before users load their data). If the database cannot find the Institution code value (submitted within the Accession record) in the Institution table, the Accession record cannot be inserted in Germinate. These database triggers and constraints keep the plant data accurate. However, they can make entering the data more difficult.

**Figure 3-2 Relationship between the Accession table and lookup table**

<table>
<thead>
<tr>
<th>Institutions</th>
<th>Accessions</th>
</tr>
</thead>
<tbody>
<tr>
<td>PK InstitutionID</td>
<td>PK AccessionID</td>
</tr>
<tr>
<td>U1 Institution Code</td>
<td>U1 AccessionNumber</td>
</tr>
<tr>
<td></td>
<td>AccessionName</td>
</tr>
<tr>
<td></td>
<td>InstitutionID</td>
</tr>
<tr>
<td>FK1.U1</td>
<td>GerminateID</td>
</tr>
<tr>
<td>FK2.U2</td>
<td>TaxonomyID</td>
</tr>
<tr>
<td>FK3</td>
<td>SubtaxalID</td>
</tr>
</tbody>
</table>
3.2 Data browsing

The Germinate community focuses on developing different types of tools to browse data stored in Germinate (Lee, 2005). There are several well developed applications for data retrieval. These applications facilitate end users to search, analyse and visualise data stored in Germinate. Some of these are described below:

- Perl-CGI Web based application: This application constructs SQL queries to interact with the Germinate database. It supplies simple data query functionality such as searching Accession descriptive information. This application allows users to search crop plant data by keywords. After searching, the application lists query results in Web pages. (SCRI, 2005)

- GDPC Web service: The Genomic Diversity and Phenotype Connection (GDPC) Web service provides a standard data access interface between data browsing tools and genetic database. (shown in Figure 3-3) (Casstevens, 2003). A GDPC Web service is set up to extract data from Germinate (and other community databases) through JDBC (Java Database Connectivity) connections. Any client application can call a “Get Data” operation of the GDPC Web service in order to consume data stored in Germinate. Currently, there are two main data analysis tools developed to access data by the Web service.

![Figure 3-3 GDPC Web service and GDPC clients](image)

The GDPC data browser: This is a JAVA based client application, which can extract data from Germinate via the GDPC Web service for end users (SCRI, 2005). It is
mainly used to browse Genotype experiment data. For example, retrieving Genotype
data for Accessions by selecting one or more experiments (shown in Figure 3-4).

Figure 3-4 GDPC Browser

Genotype data visualization tool: This tool is able to connect to a GDPC Web service to
obtain data from Germinate, and represent Genotype markers’ data by different colours
and shapes (SCRI, 2005). It assists scientists to compare Genotype data associated with
Accessions (Shown in Figure 3-5).

Figure 3-5 Genotype visualization tool (GVT)
Applications such as these allow Germinate data to be browsed in a number of different ways according to user’s requirements. These applications also have friendly interfaces, which help users to retrieve and analyse data easily. Furthermore, the GDPC Web service is data retrieval middleware that improves the accessibility of data stored in Germinate, so that other client software can be programmed more easily to retrieve data from Germinate.

### 3.3 Data loading and maintenance

Currently, there is only one method to load data into Germinate. This is through the use of Perl and PL/SQL scripts, which are provided to import data from Microsoft Excel or Access templates distributed by the Germinate website (http://germinate.scri.sari.ac.uk/cgi-bin/germinate/germinate.cgi). The Perl and PL/SQL scripts can load different types of data such as Passport data, Genetic data, Trait data and Micro-satellite data from templates into the database. For example, after collecting Passport data from different data sources, it can be mapped into an Access template with a specific format and arrangement, and then Perl scripts are executed to migrate the data from the template to a temporary table in the Germinate database. Finally, PL/SQL scripts distribute data from the temporary table into the main Germinate tables as shown in Figure 3-6. Germinate does not provide any special user facilities to update data stored in Germinate; users have to modify data by using other PostgreSQL management tools (Lee, 2005).

![Figure 3-6 Data loading flow](image1)

### 3.4 Germinate implementation

The New Zealand Institute for Crop & Food Research (CFR) is a Biological Sciences Crown Research Institute. CFR has implemented a version of Germinate to manage plant
data, such as onion and oat data. This section will describe how people at CFR currently collect input data and load data into the Germinate database.

### 3.4.1 Loading data from multiple data sources

At CFR, Germinate has been implemented on a SUSE Linux server. The SUSE Linux server operating system (version 10) supplies a broad range of open source software that includes Perl Library, PostgreSQL Server 8 and Tomcat 5 for Java Web applications. Once the SUSE Linux server is setup properly, the Germinate database can be installed on a PostgreSQL server by SQL scripts (provided on the SCRI Web site), and lookup data (e.g. institution information) can also be loaded into the database. A directory shared between Linux and Windows has been created to allow transfer of data from the loading templates in Excel and Access. Windows users have to fill in the templates on their PCs, log in to the SUSE server via Linux terminal software (e.g. Putty or VNC) and use the Perl and SQL data loading scripts to migrate the data. The following section will describe some typical data loading processes that occur at CFR.

### 3.4.2 Collecting data from different sources

Original plant data is stored in a variety of different databases and applications which are maintained by different people. The Germinate database administrator needs to extract the original data from these different sources and import them into the Germinate Excel and Access templates (McCallum, 2006). We now describe an example for loading oat data. The oat data is collected from three different sources: a statistical software package, a plant breeding system and the Gene Bank database (shown in Figure 3-7).

- **DNA markers data files**
  DNA markers (Genotype data) data is stored in text files (Vaughan, 2006). These text files are used for markers data analysis by a statistical software package. The data analysts can export markers data into an Excel spreadsheet using the statistical software, and send it to the Germinate administrator.

- **Plant breeding system**
  Some plant description information (Passport data) is stored in an Access database, which is maintained by a plant breeder (Armstrong, 2006). The plant breeder records plant information, such as plant accession name, plant appearance and collection date in the Access database. At the present time, the plant breeder takes a snapshot of part of the
Passport data, and pastes it into an Excel spreadsheet which is then passed to the Germinate administrator.

- **CFR Genebank database**
  The CFR Genebank database stores some Passport data in an SQL Server database. The information based on IPGRI oat Accession descriptors includes plant source, height, and node colour (IPGRI, 1984). Access is used to export data required at CFR from the SQL Server, and it is then given to the Germinate administrator.

![Diagram of data flow](image)

**Figure 3-7 Exporting and converting data from multiple sources**

### 3.4.3 Loading data into the database

Once the Germinate administrator gets the data from the multiple data sources described above, the data needs to be fitted into the Germinate templates. Then the data is loaded into the database using the Perl and PL/SQL scripts. There are three steps:

**Step1: Fit data into template**

Organising the supplied data into the data loading templates is a time-consuming task. Each individual data source has its own data formats, which can be quite different from the Germinate data loading templates. For this reason, the supplied data need to be rearranged
and reformatted in the templates before loading (middle part of Figure 3-7). There are some typical operations for the native data (shown in Figure 3-8).

1. Extract data: The supplied data files may contain data which does not need to be loaded into Germinate. For example, the original data might be used for a breeding system database. It not only stores a part of Passport data for Germinate, but also contains other information. Therefore, the Germinate administrator needs to extract the data needed to be put into the Germinate templates (Circle 1 of Figure 3-8).

2. Convert representation styles (Circle 2 of Figure 3-8): Some data (in the original sources) may have different representation codes to the Germinate database. Although the values represent the same contents, they are not permitted values in Germinate. For instance, Collecting Sources data must use permitted codes such as 10 for “wild”, 20 for “farm” or 30 for “market” in Germinate (MCPDs standard). However, the IPGRI oat descriptors standard for oat data, defines code 1 for “wild”, code 3 for “farm” and code 6 for “market” (IPGRI, 1984). Each organisation may also have their own definitions and descriptions for Passport data, so the Germinate database administrator has to transform their original data into valid values before loading.

3. Convert data formats (Circle 3 of Figure 3-8): The administrator also needs to modify data types in the templates to meet specific formats and rules. For instance, users must change raw data types like dates to match the format in the Germinate database. The dates in original data sources (based on the IPGRI oat descriptors standard) might be stored as a string DDMMYYYY, but Germinate requires YYYY-MM-DD. Consequently, the data values must be reformatted.

Figure 3-8 Rearranging and reformatting of data
**Step2: Transfer to Germinate**

Next, the Germinate administrator transfers the filled data templates (with the converted data) into the shared directory. The administrator logs on to SUSE Linux via the terminal software, and the Perl scripts are run (from a Linux command console) to migrate the data from the templates (in the shared directory) into a temporary table in the Germinate Database.

**Step3: Load data into proper tables**

Eventually, the administrator connects to the PostgreSQL server (through the Linux Shell console) to execute the PL/SQL data loading scripts (bottom part of Figure 3-7). The PL/SQL scripts first checks the data in order to reject invalid entries. If the data formats and values are valid, the scripts will distribute the data to the appropriate tables. Otherwise, if any invalid data input is detected in the template, the SQL scripts display error messages on the Shell console. Then, the administrator has to go back Excel/Access templates to correct the invalid values. Then they must return the Perl and PL/SQL scripts under Linux. This is repeated until no errors are found in the template data.

### 3.5 Problems with current data loading method

The current data loading method has a number of problems that mean users are reluctant to use it, so the data loading task is currently carried out by the Germinate administrator. We will now describe some of these problems.

#### 3.5.1 Collecting data from multiple sources

There are no facilities to help users to collect and load data easily from multiple data sources. The original data can be loaded only from an Excel or Access template. Any data in another form or system, must first be exported into the Excel and Access templates. In our example (top part of Figure 3-7), the oat data originates from different systems such as text files, Excel, Access and SQL server. It is not too difficult to convert data between these applications because Microsoft provides data import and export tools. However, in other institutions, the crop data might be stored in a variety of DBMS or other applications that might cause even more complications.
### 3.5.2 Rearranging and reformatting data

Each data file (exported from different sources) has to be rearranged and reformatted by the Germinate administrator to match the Germinate validation rules (middle part of Figure 3-7). Database referential integrity constraints and triggers do not allow invalid data to be imported into Germinate, so each piece of information must be processed before loading. This task takes considerable time and effort to perform. Although, Figure 3-7 just illustrates three sources for oat data at CFR, there are lots of sources for other crops data which will be migrated into Germinate eventually. If the data can be converted (to match Germinate) in each data source, and loaded directly to Germinate from different sources, it would be able to maximise data loading efficiency.

### 3.5.3 Error handling

Users often struggle with the error messages provided if they try to load invalid data (e.g. duplicate data or invalid values) (bottom part of Figure 3-7). The PL/SQL scripts only return debugging error messages provided by the database. These can be quite difficult for users to understand. For example, one common error message about date formats is shown in Figure 3-9. Actually, the problem here is that the user submitted the date data in an invalid format, because Germinate can only accept dates entered as “YYYY-MM-DD” format. However, the error and hint messages have not communicated the “YYYY-MM-DD” issue to help end users to correct their date formats (McCallum, 2006).

```
psql:load_Passport_data_g1.8.4.sql:211:  ERROR: syntax error at or near "month" at character 1
QUERY : month := substring($1 from 5 for 2)
CONTEXT: SQL statement in PL/PgSQL function "acqdate_to_datee" near line 35
psql:load_Passport_data_g1.8.4.sql:211:LINE 1: month ($1 from 5 for 2)
psql:load_Passport_data_g1.8.4.sql:213: function acqdate_to_date() does not exist
HINT: No function matches the given name and argument types. You may need to add explicit type casts
psql:load_Passport_data_g1.8.4.sql:275:
```

*Figure 3-9 Error messages from PL/SQL scripts*

### 3.5.4 Data modification

Germinate does not provide any user interface for data modification, so users have to utilise other PostgreSQL data management tools, such as PhpPgAdmin or EMS SQL
Manager. These table editors are suitable for modifying records in a single table, but they can be difficult to apply to a large database that has complex relationships among tables. For example, if users want to edit a plant accession record, they may have to access several tables to modify all the relevant information. It is also difficult to delete a record from the Accession table, because users have to first delete all the referencing data (in related tables) in order to avoid violating the referential integrity constraints. Figure 3-10 shows the error message when a user tries to delete an accession record. Most end users find this very difficult to deal with. Therefore, this approach has several limitations that make it difficult to use.

![Figure 3-10 Typical error message from PhpPgAdmin](image)

### 3.5.5 User interface

Users who are not experienced with SQL and programming do not find it easy to employ the Perl and PL/SQL scripts to populate data. At CFR this means the data needs to be loaded by a power user who understands Perl and SQL scripts well. The scientists mostly work with Excel spreadsheets and Access tables and there is no convenient way to run Perl and PL/SQL scripts from within these products. Users have to run the scripts from a different interface such as a Linux shell. This is another barrier to end users who have to switch between two environments to enter data and run scripts. This drawback is exposed when invalid data (in templates) is detected by SQL scripts on Linux Shell console, users have to switch frequently between Excel/Access templates (for editing data) and Linux Shell console (for loading/checking data). Consequently, they need a more user friendly application to submit data easily.
3.6 Summary
Germinate integrates Passport, Genotype and Phenotype data from multiple data sources in order to facilitate data queries in single database system. The plant data can be browsed and analysed from different well developed applications. However, current data loading facilities do not support loading data from different sources easily, and there are also some usability problems such as user interface and error handling. In addition, there is not a convenient end user tool for maintaining existing data in Germinate. These problems can cause difficulties when a diverse community of users needs to populate and maintain data in Germinate (Shaw, 2007).
Chapter 4 Research Design

Chapter 2 described some difficulties with building data loading applications, and Chapter 3 illustrated an example case about the problems of using a single data loading facility to load data from multiple data sources. In this chapter, we investigate a data loading solution for the example case to meet the guidelines we proposed in Chapter 2.

4.1 Review guidelines and problems

The key point of the guidelines (described in chapter 2) is that data applications should be able to help end users load data from their local data sources without having to shift the data to a template or separate loading application. We suggested that the data loading application should be tailored to conform to local data sources and users’ requirements. In order to help application developers to easily customise data loading applications for different users (in different organisations), we investigate an efficient way to support developers building data loading and updating applications.

One of the biggest difficulties in developing a data loading application is that application developers have to understand complex target database structures and implement intricate data loading and updating code. A logical record, which consists of a number of attributes from an original data source, may be stored in different physical tables in target databases. For example, in Figure 2-2 of Chapter 2, the source data (a logical record) is stored in a single spreadsheet, but the values need to be distributed among three tables in the target database. Therefore, developers have to be familiar with what data are in the target database, what the interrelationships are between different data, and how to load and update the logical records into different columns of the different tables in the target database.

The developers have to write this data access code for every different application where the original data is being stored, such as spreadsheets and Access tables. These data access procedures reflect common data loading and updating processes, which could be extracted into a single separated software component. After that, application developers can use these common data access procedures to interact with databases from client applications. The benefit is that application developers do not need to spend a considerable time in mastering target database structures and writing complex SQL commands in the client applications. In addition, if the developers need to write different data loading applications
to address multiple data sources, they do not have to repeat the data access coding for each client.

### 4.2 Middleware approach

Middleware is software, which encapsulates part of the programming logic (such as database access code) in a high-level programming interface. It facilitates application developers to easily build different distributed client applications that need similar middleware functionality (Bray, 2007; Nock, 2004). The middleware approach is suitable to address complex database access requirements for individual client application.

Although data management application types (such as Web clients, Windows clients and Mobile clients) may be different in different organisations or projects, the data access code for loading and updating data should be similar (Figure 4-1). Therefore, it is proposed to assemble these common data access operations into middleware in order to make each client application smaller and simpler.

![Figure 4-1 Middleware solution used by multiple applications](image)

We propose to separate the data loading and updating code from individual client applications. We are going to design middleware to provide a unified programming interface. It will include the common data access operations that can be used by all applications involved in data loading and updating. These operations can also validate the input data and pass appropriate feedback to the client applications. There are two main
purposes for providing the middleware solution. One is that, the middleware sits between applications and databases to insulate the details of the database from application developers. The other one is that, the middleware approach only supplies logical operations for loading and updating data requirements. We attempt to develop useful logical operations to make client application development as straightforward as possible in order to minimise the application development efforts (Nock, 2004).

**Encapsulate database structures**

It is important to encapsulate complex database structures by providing useful logical operations, because application developers can use these operations to directly insert or update logical records in the target databases. As a result, application developers do not need to understand complex database models nor write any data access code (e.g. SQL) in the client side. The data access operations significantly reduce the data loading and data maintenance development efforts and costs.

Most large databases include several logical groups of records (Figure 4-2), and an individual application might only need to manage one group of data. For instances, CFR users need to load plant Accession data from an Excel application, and use another application to load Genotype data to target databases. For this reason, we have decided to define logical records based on the different types of data and users’ requirements, e.g. descriptive data records for plant samples and laboratory records for plant samples. Each logical record normally contains many pieces of information that are separately stored in different columns of different tables in the actual database. We propose to develop a set of logical operations that use SQL scripts to check, save, read and modify these groups of logical records across different tables.
There are four typical data access operations in the middleware:

- **Insert data**: Inserting data is the most complex operation. It allows client applications to supply all relevant data attributes for entry as a unit, (e.g. 24 attributes of the Accession data), and the operation is able to map each attribute of the logical record into appropriate columns of different tables in the target database. With this operation supplied by the middleware application developers do not need to worry about which table each field of data should go into. For example, if application developers intend to build a data loading application to load the Accession data to the Germinate database in CFR, they simply invoke the “insert_logical_record” operation (Figure 4-2). Then, the operation transforms the data attributes of the logical Accession data record into appropriate columns of target tables.

  The Insert data operation also converts input data types in order to match target database requirements and carries out lookup data transformations. For example, some lookup values, such as plant species, and institution names in the source data need to be represented as foreign keys in many tables of the target database. Therefore, the Insert Data operation must find the ID values corresponding to the original data, and load these ID values into the appropriate columns (Other data converting functions can also be built in the middleware, and these are described in the section 4.5- Data transformation).
- **Select data:** The select data operation is designed to address data updating requirements for data management applications. It extracts existing logical records from different database tables, such as a dataset of a genotype record. After that, the Delete and Update methods can be used to manage the selected records. In addition, the select data operation can also be used for retrieving data for analytical software tools.

- **Update data and delete data:** The update and delete operations allow maintenance of existing data in the database. Once the selecting data operation extracts a logical data record from the database, the update method can modify the logical record across the different tables. Similarly, the delete operation can erase the logical record that may have values stored in many database tables.

**Encapsulate database rules and constraints**

It is necessary to encapsulate database rules and constraints by providing useful input data checking features in the middleware, such as checking for duplicate data and referential integrity validation. We provide data checking features within the logical operations such as Insert Data and Update Data. Any invalid input data is detected and the logical operations pass back useful error messages to client applications. After initial trials with developers (see section 5.4.1) we decided to include a result code along with a short text. The client application developers can then write specific error handling methods based on the result code or pass the text to the end user.

**Encapsulate internal database changes**

The middleware approach also prevents any purely internal database changes requiring updating each client application. For example, if the current database management system is migrated to a different product, or the database schema is redesigned, only the logical operation code in the middleware needs to be updated and the clients need not be amended. If new features or fields are added into the database, we will need to update the programming interface of the middleware. Changes to the client applications in order to take advantage of new features will be minimal.

### 4.3 Web services

In the previous section we proposed centralising the common data access operations as much as possible in middleware, so that the individual applications for each client can be smaller and quicker to develop. If the middleware software uses proprietary
implementations, such as Java and .Net, that might cause the client applications to be dependent on a single programming language and a specific platform (Bray, 2007). Therefore, we propose employing a Web service to implement the middleware to address the platform issues for developing multiple/heterogeneous applications. Web services expose the middleware programming interface as a universal interface available for different platforms and programming languages.

### 4.3.1 Background of Web services

In 2002, the World Wide Web Consortium (W3C) defined the Web service as an industry standard system to interoperate between different software applications across diverse platforms via the network. Web services use the XML format to transfer data between Web service clients and Web service servers through a standard TCP/IP protocol, such as HTTP (Figure 4-3). XML and HTTP are industry standards, which allow information to be transferred between multiple platforms. In addition, a Web service is “firewall friendly”, because most firewalls open port 80 for HTTP communications. As a result, the Web service can be easily invoked across firewalls.

![Figure 4-3 Data access using a web service](image)

WSDL (Web Service Description Language) files are XML based definitions that are designed to describe details of a Web service’s operations/methods. The definitions include the operation name, input/out parameter names and types. The client side applications need to read the WSDL files in order to find out which operations are provided by the Web service and how to invoke these service operations (W3C, 2001).

A Web service has advantages over more traditional approaches. For example, Java Remote Method Invocation (RMI) enables remote Java applications (e.g. server applications) to be invoked from other Java applications (client applications) (Sun, 1995). However, RMI is a Java language specific protocol, so it can be difficult to use with other programming languages, such as Microsoft .NET. In contrast, Web services normally apply a generic XML based communication protocol, such as SOAP, to support exchange
client requests and server responses in XML (Figure 4-4). Therefore, the Web services can be easily accessed from different client applications that are developed on different platforms and languages.

![Diagram of Java RMI versus SOAP](image)

**Figure 4-4 Java RMI versus SOAP**

Furthermore, it is easy to use Web services from different programming languages and platforms. There are many development tools, such as Microsoft .Net and Java NetBeans that can read the WSDL files and generate Web service client (stub) code to map remote Web service methods as native functions (Microsoft, 2005; Sun, 2005). The client applications can directly call the Web service methods via the stub code.

### 4.3.2 Example of using a Web service

**Data querying**

A Web service provides a simple API to different applications for querying databases. For example, different types of information need to be shared among international communities via the Internet. Many data providers employ Web services to supply APIs with logical data query operations, such as retrieving weather temperature information (Steiner, 2005) and DNA sequence data (NCBI, 2005). Application developers can easily use these services to retrieve data for their local data analysis applications. In addition, Web services are also used for federated database queries. For example, the Australian
government department of agriculture allows users to query plant pest information from different data sources via a unified Web service interface (Naumann, 2006).

**Data integration and data warehousing**

Web services are being increasingly used in data integration systems. For instance, Hansen (2002) reported using a Web service to integrate data from heterogeneous systems and applications over the Internet. Web services are also used to support ETL applications to extract data from heterogeneous data sources. For example, Schlesinger (2005) described using Web services to collect data from each data source and store the collected data into a centralised Oracle database. The ETL Web service client application is attached to each distributed data source for delivering source data to the target database.

**Data management**

Using Web services to insert and update data is still comparatively rare. However, a few eCommerce projects are using Web services to support data management. The Amazon Web service provides several methods such as cartCreate, cartModify and cartClear to support developers in building local shopping client applications. These client applications allow customers to create and modify remote shopping cart contents hosted on Amazon servers (Amazon, 2006).

The Google AdWord Web service allows client applications to manage online advertising accounts with functionality such as Add Ads, Modify Ads and Get Ads. The Web service provides a generic programming interface for developers to easily build client applications for managing online ads (Google, 2006). These approaches demonstrate that a Web Service can be used as a unified programming interface that connects individual data sources or data management applications to target databases.

**4.4 Using a Web service as middleware**

We propose to use a Web service as our data access middleware. It sits between client applications and a database server and provide methods for managing logical data records. These methods can be accessed from any Web service enabled language on any platform, because they use the standard XML based communication protocol to exchange data between clients and Web services. With the Web service in place, individual applications can be simply tailored for local data sources and user requirements, e.g. bulk loading data from Excel spreadsheets.
In order to help developers to interact with the Web service we can provide a toolkit for each language or system where there is source data. Each toolkit will provide native data access functions to map to the remote Web service functions. Application developers can use the native toolkit functions to directly access the databases.

The proposed system (see Figure 4-5) includes four parts, of which we provide part 2 and part 3. We also create sample user applications (part 1) for testing purposes.

1) Client application           (e.g. An Excel application)
2) Client toolkit               (e.g. a VBA toolkit)
3) Data access Web service      (e.g. a PHP Web service)
4) Database

We will now look at the different parts of this architecture in more detail.

### 4.4.1 Data access Web service

The Web service exposes essential data access methods in a unified programming interface to enable developers to interact with the database. Part 3 of Figure 4-5 shows the Web service providing a set of middleware functions to manipulate data in the target database. The functionality includes mapping and transforming data for loading. This approach centralises common data access code in the Web service in order to reduce the effort required to write this intricate data access code in each of the client applications. Therefore, the Web service makes client applications smaller and easier to develop and maintain.

Application developers can call the Web service methods from different applications, such as a spreadsheet or a Web form. Because the Web service API is accessible from different
applications developed in different languages and platforms, it provides a platform independent way for customising client applications to suit diverse local data sources.

The Web service provides an alternative way to enable developers to satisfy different users’ needs. The biggest benefit for users is that the customised client applications help users load data directly from local data sources. Therefore, users do not need to shift different original data into a specific application for loading (e.g. a specific data loading template). It eliminates the need to export and transform data into a data loading template from the different data sources. Users can simply load and update data in the environment that they are familiar with (e.g. Excel or Web clients).

### 4.4.2 Client toolkit

We provide Web service client toolkits (Figure 4-6) to help developers invoke the Web service by calling a set of language specific functions (e.g. Visual Basic for Application, PHP, JAVA). Each toolkit can be easily incorporated into a variety of client applications in order to communicate with the Web service. For example, the VBA toolkit provides a function to call the Insert Data function of the Web service. Application developers just need to provide the appropriate data from Excel worksheets or Access tables as parameters to the VBA function. Application developers do not need to know anything about manipulating XML messages between applications and Web services, because they can use these toolkit functions to directly interact with databases. This approach considerably reduces the effort required to develop client applications.

![Figure 4-6 Web service client toolkits](image-url)
Another function provided by each toolkit is Web service initialisation which prepares the application to make use of the Web service. The function accepts the Web server name and authentication information as parameters. This function also checks that the database and Web services are available. For example, if the database management system is not operational, the initialisation function returns an appropriate error message to the application.

### 4.4.3 Web service client application

After providing the Web service and the client toolkit, client applications can be developed to load and maintain data. Figure 4-7 shows a typical data loading workflow from a client application to the target database via the Web service. For example, an Excel data entry application batch process to submit several records from an Excel worksheet.

- The data in the user’s worksheet (Step 1) are passed to the client toolkit (Step 2).
- Then, the toolkit transfers the data as parameters of an XML request (Step 3), which calls the appropriate data access function of the Web service, e.g. the Insert Data function.
- Next, the Web service executes SQL queries to interact with the database (Step 4 and 5).
- Whether the transaction succeeds or fails, the Web service function sends back XML response messages to the toolkit (Step 6 and 7).
- Eventually, the toolkit parses the XML responses to a native data structure and displays it on the Excel worksheet (Step 8).

![Figure 4-7 Data loading workflow](image-url)
This structure makes the client application development easier. Figure 4-7 shows the entire processes that involves nine steps. However, client application developers only need to deal with passing data (step 2) to the toolkit and returning responses (step 8) to the user application. Other complex steps are done by the Web service and the client toolkit. It means that application developers only need to add a few lines of code to pass the relevant parameters from the user interface or data source to the toolkit for data loading.

4.5 Data transformation

It is important to convert native data types and representation styles to match target database requirements. We supply automatic data converting functions that allow end users to supply data in native formats for loading. Data transformation functions could be developed on the Web service, the toolkit, or the application to address different data converting requirements. We propose to provide some transformation functions in the Web service and the client toolkit for common requirements. Application developers can write individual converting functions in the client applications to satisfy specific user requirements.

4.5.1 Web service: data transformation functions

In the Web service, it is necessary to meet common data conversions required by most community users. An example is the conversion of plant data descriptors between two international coding schemas (mentioned in section 3.4.3). Germinate requires data to conform to the MCPDs standard, an international coding schema. Some institutions might use a different data coding schema such as oat descriptor standard (IPGRI, 1984). E.g. for “Collecting Source”, IPGRI uses number 1 for “Wild” whereas the MCPD standard uses 10 (see Table 4-1). We provide transformation functions for these kinds of data converting operations in the Web service where they will be available to the whole community.

Table 4-1 International coding schema transformations

<table>
<thead>
<tr>
<th>Source data (IPGRI)</th>
<th>Germinate (MCPDs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collecting Source</td>
<td>Comments</td>
</tr>
<tr>
<td>1</td>
<td>Wild</td>
</tr>
<tr>
<td>3</td>
<td>Farm store</td>
</tr>
<tr>
<td>6</td>
<td>Commercial market</td>
</tr>
</tbody>
</table>
4.5.2 Client toolkit: data transformation functions

In the toolkit, we also plan to develop functions for converting native data types (based on specific languages) to match database required types. One example is adapting Excel or VBA dates into date formats acceptable in Germinate. These functions are a part of the insert data toolkit function so application developers do not need to explicitly call the converting functions. For example, a date converting function is provided to convert dates entered in different ways in Excel, e.g. standard date formats or dates entered as strings. These need to be converted to string format “yyyy-mm-dd” required by Germinate (example data are shown in Table 4-2). Users do not need worry whether the original dates are stored as text, number, or date formats, because the toolkit can detect the different formats and deal with them appropriately.

Table 4-2 Date transformations

<table>
<thead>
<tr>
<th>Source data</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/03/2001</td>
<td>entered as date</td>
</tr>
<tr>
<td>1 March 2001</td>
<td>entered as date</td>
</tr>
<tr>
<td>01-03-2001</td>
<td>entered as string</td>
</tr>
<tr>
<td>20010301</td>
<td>entered as integer or string</td>
</tr>
</tbody>
</table>

4.5.3 Client applications: data transformation functions

In addition, application developers can create specific conversions in individual client applications to cope with local data conversations. These conversions are not be provided by the Web service nor the client toolkit, because these functions only need to be supplied to address particular local needs. There are two examples: First, at CFR some local dates may be missing or only consist of a year that is entered as an integer or string (year plant introduced), e.g. “1995”. Different organisations may have different ways of handling these situations so the application developers can write functions to convert the data into standard date formats, or add default values for the missing data. The conventions adopted at CFR are shown in Table 4-3. Moreover, each institution may have their own definitions and descriptions for local data, such as the institution name. Descriptors for institution (in the source data) need to be converted to match the values in Germinate. For example, “CFR” in the source data needs to be converted to the international institution descriptor:
NZL027 (Table 4-3). Once the conversion functions are built in the client applications, the local descriptors are automatically converted into valid presentations for loading.

Table 4-3 Local data transformation

<table>
<thead>
<tr>
<th>Source data</th>
<th>Germinate required data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995 (only given as year )</td>
<td>1995-01-01</td>
</tr>
<tr>
<td>(missing)</td>
<td>2007-06-15 (current date as default)</td>
</tr>
<tr>
<td>CFR</td>
<td>NZL027</td>
</tr>
<tr>
<td>CFR</td>
<td>NZL027</td>
</tr>
</tbody>
</table>

4.6 Implementation of the Germinate Web service

The Germinate database can be thought of as three groups of interrelated logical data record: description data of plant samples (Accession data), Genotype data and Phenotype data. We initially developed the Web service for loading Accession data. The Web service methods hide the Germinate structure and expose logical operations as a simple interface for client application developers. Application developers can load the Accession data records across different tables by using the appropriate toolkit in their applications. Web service functions for the other two groups of data can be developed in a similar way in the future.

4.6.1 Germinate web service architecture

The Germinate database at CFR runs on a PostgreSQL 8.2 database management system on a SUSE Linux server 10. The Germinate Web service was developed in PHP 5 and under Apache 2 Web server (running on the same Linux server).

The Germinate Web service consists of two parts: The Web service API and the database access functions (shown in Figure 4-8). The NuSOAP library (NuSphere, 2005) has been used to provide the Web service API and the WSDL for the Web service. The PEAR DB_DataObject (PEAR, 2006) has been used to provide database access code to implement the Web service API abstractions.
The Web service interface is developed using NuSOAP. NuSOAP is an open source Web service Library for PHP. It includes a set of PHP classes to facilitate developers to build Web services based on the SOAP 1.1 standard. It parses client XML request messages into native PHP data structures and constructs XML responses to be returned to the client application. Figure 4-9 shows the generic flow of data through the system using the example of loading an Accession record into Germinate from an Excel worksheet.

The client toolkit sends an Insert Data Web service request as a XML (SOAP) message. The Web service parses the request and passes the values into the internal Insert Data function which executes the data loading code. After the data is inserted, the Web service returns the Insert Data result as an XML response (e.g. “01: The accession has been inserted”) to the Client toolkit. The application can then display the text of the response message in another cell (e.g. D2).
Figure 4-9 Data flow through the system
4.6.2 Insert data function

At this stage, we have only developed the Insert Data function for the Accession data, but we plan to provide other functions, such as Select and Update functions in the future. In this section, we describe details of the insert data method.

Inserting Accession data is the most complex operation of the Germinate Web service. Accession data is the base data which must be present before other genotype and phenotype data can be loaded. A logical Accession data record contains 24 fields of data that are stored in 20 different tables in Germinate. Our data insert method can receive the 24 fields of Accession data from a client application and direct them into the correct tables in Germinate.

The insert method has 24 input parameters for a logical Accession data record. We have discussed with the CFR Germinate administrator the minimal set of plant data that must be loaded into Germinate for data analysis. At CFR, there are 11 pieces of information which are always required. These include Accession number, Species and Collecting date. Other pieces of data, e.g. Accession name, are not necessary for all research projects, so we have set them as optional input parameters.

The insert data method includes six procedures: data mapping, input data checking, default values setting, data transformation, inserting data, and returning response messages.

Mapping data

The insert data method allows application developers to supply all relevant data, e.g. for an Accession record, as a unit. First, we must map the logical record attributes into the proper columns of the different Germinate tables as shown in Figure 4-10. The benefit of doing this in the Web service is that application developers do not need to worry about which table each field of data should go into. Eventually, we use the DB_DataObject to generate and execute SQL commands to insert the data into Germinate.
Check invalid data

Before loading data into the tables, the insert function checks the data in order to detect invalid values. It also returns useful error messages and codes to client toolkits.

- Firstly, the function checks for duplicate records based on the Germinate identity name. The Germinate identity name is the combination of Accession number and Institution code and must be unique in Germinate.

- Secondly, the function checks all required input data values. If the required data values are not supplied, the function returns an error message.

- Thirdly, the function checks for Collecting date and Acquisition date representations. If the format is different from the required format, our Insert Data function returns an error message.

- Fourthly, the function validates lookup data values to ensure referential integrity. That is to say, the function checks that the lookup values are correct. For example, if
users submit a wrong institution code (i.e. one that is not present in the Institution table) the function returns an error message to the client toolkit to handle this problem.

**Transform Lookup data**
The Web service insert method converts the lookup data values into primary keys, and stores the keys in target tables as foreign keys. There are 10 pieces of input data that are lookup data, which are stored as foreign keys to related tables. These include the Institution code ID in the Accession table. The value of the Institution code is located in the Institution table, and the corresponding primary key value is recorded in the corresponding records in the Accession table as a foreign key as shown in Figure 4-10.

**Setting default values**
We also setup default values for some data attributes based on CFR users’ requirements. For example, the CFR is a research institution, so the Biological status default value is assigned to "Breeding/research material”. We also set default values for Collecting source and Storage attributes. If users do not provide input data for Biological status, Collecting source, and Storage, the default values are applied to take the place of the empty values. In the future, we propose to provide a configuration file for users to define their required inputs and default values (refer to Section 5.3.2), because users might use Germinate in different ways.

**Load data into proper tables**
After the above operations have taken place, the input data can be loaded into the appropriate tables in Germinate. We use DB_DataObject to generate SQL scripts to insert the 24 fields of Accession data into 20 tables.

**Outputs**
The data insert function returns response messages to the toolkit. A message consists of a code and a short text (Table 4-4). For example, if the insertion was successful, the function returns “0: The accession has been inserted”. However, if there are problems during the loading, an error is returned that includes a result code and a short text. For example “1: No Accession number supplied”. The 1 is the result code, the text is the explanation of the error. The following table shows all messages that are returned from the insert data function. These error messages are designed to help users to locate and correct any invalid input data which they have tried to load into Germinate. Application developers can
provide some error handling procedures in the client applications based on the returned result code or simply present the error message text to the user.

Table 4-4 Result codes and messages

<table>
<thead>
<tr>
<th>Result Code</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The accession has been inserted</td>
</tr>
<tr>
<td>1</td>
<td>No accession number supplied</td>
</tr>
<tr>
<td>2</td>
<td>The accession already exists</td>
</tr>
<tr>
<td>3</td>
<td>Invalid institution code</td>
</tr>
<tr>
<td>4</td>
<td>Invalid donor code</td>
</tr>
<tr>
<td>5</td>
<td>Invalid collecting code</td>
</tr>
<tr>
<td>6</td>
<td>Invalid breeding code</td>
</tr>
<tr>
<td>7</td>
<td>Invalid duplicate site code</td>
</tr>
<tr>
<td>8</td>
<td>Invalid taxonomy</td>
</tr>
<tr>
<td>9</td>
<td>Invalid subtaxa</td>
</tr>
<tr>
<td>10</td>
<td>Invalid biological status</td>
</tr>
<tr>
<td>11</td>
<td>Invalid collecting source</td>
</tr>
<tr>
<td>12</td>
<td>Invalid storage</td>
</tr>
<tr>
<td>13</td>
<td>Invalid date format (collecting date), please follow YYYY-MM-DD</td>
</tr>
<tr>
<td>14</td>
<td>Invalid date format (AcqDate), please follow YYYY-MM-DD</td>
</tr>
</tbody>
</table>

4.6.3 Web service authentication

We have enabled the Apache server HTTP authentication to control access to the Germinate Web service. An access control list has been created on the Apache server. This file contains a group of Web service user names and passwords. Only valid users can access the Web service to load data into Germinate. The credentials can be passed into the Web service from a client SOAP message header. This authentication process is handled by the toolkit initialisation function. In addition, if the user name and password are invalid, the Web service returns an error message to inform the user that access has been denied.

4.7 Client toolkit implementation

At CFR, most data are managed by Microsoft Excel and Access applications, so we have developed a VBA toolkit for loading data from these data sources. The Microsoft Office 2003 Web service Toolkit can generate VBA proxy functions code (local functions) from a
Web service WSDL file (Microsoft, 2005). This feature was used to assist in creating the client toolkit. The toolkit source code is distributed in two VBA module files, “accession.cls” and “toolkit.bas” respectively. Application developers can easily import these two files into a VBA project.

4.7.1 Web service toolkit functions

The toolkit sits between a client application and the Web service. It receives the logical Accession data records from Excel or Access for inserting via the Web service. It involves the following functionality.

Accession object

The toolkit provides a VBA object (Accession) to load Accession data records (24 pieces of data). The Accession object includes 24 properties for the corresponding fields of Accession data. The Accession object can be passed to the toolkit Insert Data function for loading data to Germinate via the Web service. Figure 4-11 shows how to create an Accession object and to sets its properties’ values to local data in an Excel spreadsheet. For example, if the institution code value is stored in cell C2 of the Excel spreadsheet (top part of Figure 4-11), we can set the InstCode property value of the Accession object to the value in cell C2 (line 3 of Figure 4-11).

```
1   Dim Acce As New Accession
2   With Acce
3          .InstCode = Range("C2")
4          .AccNum = Range("D2")
5         .AccName = Range("M2")
6         . . .
7   End With
```

Figure 4-11 Example of using Accession object

Web service initialisation procedure

The TK_initialisation procedure uses the Microsoft SOAP client object to initialise the Web service. Application developers only need provide the Web server name as a parameter to the initialisation procedure. A user name and password are optional input
parameters for this procedure. If application developers setup HTTP authentication for controlling access to the Web service, the initialisation procedure can pass credentials to the Web service based on the supplied user name and password as shown in Figure 4-12.

| TK_initialisation | "localhost", "Scott", "abc123" |

**Figure 4-12 Example of using the toolkit initialisation procedure**

The procedure handles initialisation problems and alerts the user via popup error messages. The toolkit catches any VBA errors from the Microsoft SOAP object during the Web service initialisation, and passes back understandable text error messages to the user. For example, if the Web service or the Germinate database are not operational, the initialisation procedure displays a message to check the Web and database servers. If the Web service authentication fails, it informs users to check the user name and password.

**TK_insertAcc function**

The TK_insertAcc function passes the Accession object data (24 data attributes) to the Web service for loading. After the Web service processes the request and sends back a response, the TK_insertAcc function returns a result code (shown in Table 4-4). The code provides a convenient way to programmatically handle errors. Application developers can use the result code if they wish to take a particular action. For example, application developers could highlight error messages in different colours. For example, a duplicate record could be considered a “minor mistake” as the data has previously been loaded. In this example the developer has chosen to show it as lighter colour (line 7 of Figure 4-13). In contrast, the invalid Institution code error is a fatal error as it means the record can not be loaded into Germinate. Therefore, the developer has chosen to use red to highlight this problem (line 9 of Figure 4-13). The application developer could take other actions or customise the error message displayed to the user if they wish.
Sample code

1  . . .’code for collecting data from the cells into the Accession object

2  ResponseCode = Tk_InsertAcc (Acce)  ‘Insert a Accession record

3  Range("AJ" & rowNumber) = TK_LastErrorText  ‘Print response texts in column AJ

4  If ResponseCode = 0 Then

5       Range("AJ" & rowNumber).Font.Color = RGB(0, 0, 255)  ‘Blue to display inserting successfully

6  ElseIf ResponseCode = 2  Then

7       Range("AJ" & rowNumber).Font.Color = RGB(0, 255, 0)  ‘Green to display duplicate records errors

8  Else

9       Range("AJ" & rowNumber).Font.Color = RGB(255, 0, 0)  ‘Red to highlight fatal input errors

10 End If

. . .

Figure 4-13 Example of error handling

TK_LastErrorText function

The TK_LastErrorText returns the text of the most recent error message returned from the Web service. This is in a suitable format for displaying to end users. The example shows how to use the Tk_LastErrorText function (Line 3 of Figure 4-13) to display the text of an error message in a column of the Excel spreadsheet.

TK_ConvertDate function

In the toolkit, we have built a date conversion function to convert Excel or Access specific date representation styles and formats into the Germinate required date format. The TK_ConvertDate function is a sub function of the TK_InsertAcc function, so application developers do not need to call it directly. TK_InsertAcc function automatically converts native date values to a correct format.
4.7.2 Instruction documentation

We have developed a set of instructions to assist VBA developers use the toolkits to customise their data loading applications. The toolkit instruction documentation includes descriptions of the toolkit function, a step by step development tutorial, and sample application code for bulk data loading (see Appendix A).

4.8 Summary

In order to meet the guidelines for the data loading applications which we developed in Chapter 2, we have developed a Germinate Web service to provide a data loading function. We have also proposed the idea of a Client toolkit to allow application developers to customise data loading applications for different data sources with minimal effort. We have developed a VBA toolkit for Microsoft Office applications which allows application developers to load Accession data stored in Excel into Germinate by calling only two toolkit functions.

The Web service hides the details of the database so that application developers do not need to spend considerable time and effort in understanding the complex Germinate schema which has 72 interrelated tables. Moreover, the Web service insulates local applications from database changes. If new functionality is added to Germinate, we only need update the Web service and toolkit. The changes to the client applications themselves will be minimal.

The Web service and toolkit include functions for automatic conversion of the stored data types and representations styles to match the target database requirements. These functions can address the most common data transformation needs for community users. In addition, application developers can also write specific functions to satisfy local users’ conversion requirements.

More importantly, the customised applications allow users to load data directly from their project data sources, such as spreadsheets, small databases, or text files. End users do not need to fit original data into a specific template or application for loading and the effort required to transform data between different formats and systems is minimised. Moreover, the users are provided with useful error messages which enable them to correct any invalid data they attempt to upload.
Chapter 5 Evaluation

The Germinate Accession records involve 24 fields of plant sample data, which, at CFR, are stored in different applications and databases, such as Access or Excel. To load into Germinate, data must be exported from each different source and fitted into the data loading template. Using this Germinate data loading approach makes it difficult for novice users to transform and reformat the different Accession data into the required formats for loading.

In order to support users loading data directly from different data sources, we suggested using Web services to provide data access middleware between client applications and databases. This solution assists application developers to easily and quickly build client applications based on local data sources and user requirements. We have implemented a Web service at CFR to allow users to load data into the Germinate database.

The Web service approach was evaluated through user and developer trials. The trials involved several tasks for users and developers to complete. We also prepared questionnaires about the trial tasks to get detailed feedback from both end users and application developers. The aim of the user trials was to evaluate how well the customised applications assist users to load data from different data sources via the Web service. The goal of the developer trials was to assess how the Web service and the client toolkit assist developers to create customised data loading applications. In addition, we also looked at some Web service issues, e.g. performance, reliability, and security. In this chapter, we summarise and discuss the trial results and these Web service issues.

5.1 End user trial

The end user trial looked at how well users could use customised applications to directly load data from data sources via the Web service.

5.1.1 Evaluation methodology (for end users)

In order to determine if the Web service approach would help users to easily load data from different data sources, we developed Excel and Access client applications with essential data loading functionality (e.g. validating, converting, and loading data). These enable end users to directly load data from their Excel and Access data sources via the Web service. The user trials were designed to assess the usefulness of these customised
applications. We prepared questionnaires based on the user trial tasks (shown in Appendix B).

Data loading applications for user trials
A VB for Excel application allows Excel users to load data from a worksheet directly to Germinate via the Web service. We took a typical worksheet from a CFR scientist and added functionality to load the data to Germinate. Figure 5-1 shows a part of the data in a typical worksheet. There were 24 columns of Accession data, which needs to be loaded into Germinate.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ID</td>
<td>NICODE</td>
<td>INSTCODE</td>
<td>ACCENUMB</td>
<td>COLLNUMB</td>
<td>COLLCODE</td>
<td>GENUS</td>
</tr>
<tr>
<td>2</td>
<td>26 NZL</td>
<td>NZL027</td>
<td>ACC100360</td>
<td>1 NZL027</td>
<td>Avena</td>
<td>sativa</td>
<td>. . .</td>
</tr>
<tr>
<td>3</td>
<td>27 NZL</td>
<td>ACE123</td>
<td>ACC100361</td>
<td>1 NZL027</td>
<td>Avena</td>
<td>sativa</td>
<td>. . .</td>
</tr>
<tr>
<td>4</td>
<td>28 NZL</td>
<td>NZL027</td>
<td>ACC100362</td>
<td>1 NZL027</td>
<td>Avena</td>
<td>sativa</td>
<td>. . .</td>
</tr>
<tr>
<td>5</td>
<td>29 NZL</td>
<td>NZL027</td>
<td>ACC100363</td>
<td>1 NZL027</td>
<td>Avena</td>
<td>sativa</td>
<td>. . .</td>
</tr>
<tr>
<td>6</td>
<td>30 NZL</td>
<td>NZL027</td>
<td>ACC100364</td>
<td>1 NZL027</td>
<td>Avena</td>
<td>sativa</td>
<td>. . .</td>
</tr>
</tbody>
</table>

Figure 5-1 Typical Germinate data in Excel
We wrote program to collect input data from the worksheet cells, and pass them to the VBA toolkit for loading via the Web service. We also created an Insert Data button on the Worksheet to help users run the program. Eventually, we displayed Web service response messages in the last column of the worksheet.

The user trial application is shown in Figure 5-2. It allows users to load the Accession data directly into Germinate from the worksheet. The figure shows 5 typical Accession data records on the worksheet. After users click the Insert Data button (top right corner of Figure 5-2), the data in the cells is sent to the Web service for loading. Response messages from the Web service are presented in the last column (AJ) to indicate loading status. If a record is loaded successfully, it displays “the accession has been inserted” in a blue colour. If invalid input data is detected, the application uses a red colour to highlight the error message in the response column. For example, if an invalid institution code is detected, the error message shows “Error: Invalid institution code”.

58
In addition, users are allowed to specify the input data (column N) in a variety of formats: for example, a String date ("1990-11-11") or the year as an Integer ("1999"). The application is able to transform the dates into the required format for loading into Germinate. However, if an input date (e.g. ‘1991-16’) can not be recognised by the application, an error message is displayed so users can correct the input.

Similarly to the Excel application, an Access application was developed to help users to load original data from a typical Access data source. Users can load Accession data directly from the Access table by an “Insert Data” button. Then the Web service response messages are displayed in the last column.

**User trial tasks**

Users performed three trial tasks based on actual data loading scenarios: loading a single record, loading invalid data, bulk data loading.

**User task 1: Load single record**

This task involved inserting a single record into Germinate. The main purpose was to evaluate whether the client applications are easy and understandable to users. Users had to manually enter a single Accession data record into the Excel or Access application, and submit this record for loading. In addition, we also examined whether the response message clearly showed the data loading outcome for users.

**User task 2: Load multiple records with invalid data**

This involved the users trying to load several records with invalid data (every record contained an error). The main purpose was to evaluate whether the error messages helped users to identify and correct invalid data. End users were asked to copy and paste 13 records with different types of invalid data into the Excel or Access data application. These data records are shown in top part of Figure 5-3 with the errors highlighted (but they were not highlighted during the trial). After users clicked Insert Data, the error messages were displayed in the response column shown in bottom part of Figure 5-3. Then the users were
required to try to correct the invalid data based on the error messages. There were three possibilities of user action:

- Users fix an error and resubmit data (e.g. fix a spelling mistake)
- Users discard a data entry that is already in Germinate (e.g. a duplicate record)
- Users note a data entry which needs further correction (e.g. need to lookup a field value)

---

### Errors highlighted in the provided data

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>J</th>
<th>N</th>
<th>T</th>
<th>U</th>
<th>V</th>
<th>X</th>
<th>Y</th>
<th>AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INSTCODE</td>
<td>ACCNUMB</td>
<td>COLLGOC</td>
<td>GENUS</td>
<td>SUBACODE</td>
<td>COLLAB</td>
<td>PREDCO</td>
<td>SAMPS</td>
<td>COLLSRC</td>
<td>DONOR</td>
<td>STORAGE</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>NZL027</td>
<td>NZA10</td>
<td>Auckland</td>
<td>ex</td>
<td>NZL027</td>
<td>Arena</td>
<td>oat</td>
<td>5120-0211</td>
<td>05120002</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
</tr>
<tr>
<td>3</td>
<td>NZL027</td>
<td>NZA10</td>
<td>Auckland</td>
<td>ex</td>
<td>NZL027</td>
<td>Arena</td>
<td>oat</td>
<td>5120-0212</td>
<td>05120001</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
</tr>
<tr>
<td>4</td>
<td>NZL027</td>
<td>NZA409, 763</td>
<td>NZL027</td>
<td>Auckland</td>
<td>oat</td>
<td>5120-0213</td>
<td>05120001</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>NZL027</td>
<td>NZA408, 764</td>
<td>NZL027</td>
<td>Auckland</td>
<td>oat</td>
<td>5120-0214</td>
<td>05120001</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>NZL027</td>
<td>NZA408, 765</td>
<td>NZL027</td>
<td>Auckland</td>
<td>oat</td>
<td>5120-0215</td>
<td>05120001</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>NZL027</td>
<td>NZA405, 700</td>
<td>NZL027</td>
<td>Auckland</td>
<td>oat</td>
<td>5120-0216</td>
<td>05120001</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>NZL027</td>
<td>NZA408, 766</td>
<td>Auckland</td>
<td>oat</td>
<td>5120-0217</td>
<td>05120001</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>NZL027</td>
<td>NZA408, 766</td>
<td>NZL027</td>
<td>Auckland</td>
<td>oat</td>
<td>5120-0218</td>
<td>05120001</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>NZL027</td>
<td>NZA408, 768</td>
<td>NZL027</td>
<td>Auckland</td>
<td>oat</td>
<td>5120-0219</td>
<td>05120001</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>NZL027</td>
<td>NZA409, 770</td>
<td>NZL027</td>
<td>Auckland</td>
<td>oat</td>
<td>5120-0220</td>
<td>05120001</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>NZL027</td>
<td>NZA419, 771</td>
<td>NZL027</td>
<td>Auckland</td>
<td>oat</td>
<td>5120-0221</td>
<td>05120001</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>NZL027</td>
<td>NZA419, 931</td>
<td>NZL027</td>
<td>Auckland</td>
<td>oat</td>
<td>1999-11-9</td>
<td>111-11-9</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>NZL027</td>
<td>NZA419, 932</td>
<td>NZL027</td>
<td>Auckland</td>
<td>oat</td>
<td>1999-13-10</td>
<td>1111-11-11</td>
<td>NZL027</td>
<td>Natural Farm storeNZL027</td>
<td>Seed collection</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Corresponding Errors messages

<table>
<thead>
<tr>
<th>INSTCODE</th>
<th>ACCNUMB</th>
<th>Response messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>NZL027</td>
<td>NZA10</td>
<td>Error: invalid institution code</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA408, 766</td>
<td>Error: no accession number supplied</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA408, 768</td>
<td>Error: invalid subsite</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA409, 764</td>
<td>Error: invalid taxonomy</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA409, 770</td>
<td>Error: invalid donor code</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA403, 700</td>
<td>The accession already exists</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA403, 767</td>
<td>Error: invalid collecting code</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA403, 768</td>
<td>Error: invalid breeding code</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA403, 769</td>
<td>Error: invalid biological status</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA403, 770</td>
<td>Error: invalid collecting source</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA419, 771</td>
<td>Error: invalid storage</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA419, 931</td>
<td>Error: invalid data format (collecting date), please follow YYYY-MM-DD</td>
</tr>
<tr>
<td>NZL027</td>
<td>NZA419, 932</td>
<td>Error: invalid data format (AcqDate), please follow YYYY-MM-DD</td>
</tr>
</tbody>
</table>

---

**Figure 5-3 Data errors and response**

**User task 3: Bulk loading 500 records**

This was to test acceptability of the time to the load a large record set. Users were asked to load a set of 500 records into Germinate. The 500 records were provided in another worksheet/Access table. Users needed to copy and paste these records into their Excel and Access application for loading.
5.1.2 Results

There were two CFR users who tested the applications. One CFR user normally stores plant data in Excel spreadsheets, and the other user normally stores plant data in an Access database. They were asked to carry out the user trial using the applications with which they were most familiar. Before the user trials, both were introduced to the basic usage of the data loading applications, such as how to enter data into the spreadsheet or database table and how to use the Insert Data button.

User task 1 results:
Both users were able to load the single record into Germinate successfully. They said that the data loading response message clearly informed them that they had successfully inserted the record in Germinate.

User task 2 results:
After each user tried to load the 13 invalid data records, the error messages were displayed in the last column. They could understand the error messages and correct invalid inputs. For example, they first had a look at the response column to find which rows had error messages, then located the invalid data in the specific fields based on the error messages. They both thought that the error messages were easy to understand, so they were able to correct most of the invalid data, and resubmit the records for loading. For instance, they corrected the invalid date values and some spelling mistakes for the plant sample species names.

In some cases, it was difficult for the users to correct the invalid data, e.g. they needed to look up the correct code or name in a related database table. For example, if the system shows “Error: Invalid institution code”, users would have to find the correct code from the Institution table in Germinate before they could fix the invalid code. An improvement to the client applications would be to extract the lookup data from Germinate into dropdown list boxes from which users can select. This approach would also reduce input errors in the lookup data.

There was another issue with reloading the corrected data during the trial. After users corrected the errors and tried to reload, the application resubmitted all the data records including these that had been successfully inserted before. Response messages for these records then showed they had already been inserted (Figure 5-4).
During the trial, this did not affect the users’ ability to resubmit data, but it would be better to improve application in order to avoid this duplication. The application could allow users to only reload the “wrong rows”, or the application could allow users to choose the rows that they want to resubmit. Alternately, it could move all inserted data records (“correct rows”) into another worksheet/table, so the data loading worksheet would only contain the rows with errors for users to modify.

**User task 3 results:**
The 500 data records were successfully loaded into Germinate by both users. The average response time was less than 2 minutes. Both the Excel and Access users were satisfied with the system processing time. (A discussion about the Web service processing time is included in Section 5.3.5)

The users made one useful suggestion. When loading a large number of records, the application did not provide any information about the loading progress. Users suggested providing a “progress bar” that indicates the current loading operation status.

**5.1.3 Summary of user comments**
The users filled out questionnaires on their use of the applications (see Appendix B). Below is a summary of their comments.

**Loading data directly from sources**
CFR users thought that the Excel and Access applications were very useful tools for loading data directly from their project data sources. They agreed that being able to directly load data into Germinate from their customised applications was an improvement and meant that there would be no need to export and send data to the Germinate administrator for loading.

In addition, the Web service approach can also be used to load data from any other data source to any other database. The Access user said that “the Web service approach can
also be applied to my project for collecting data from mobile devices into an Access database”. That is to say, the Web service approach is not only applicable for the Germinate database, but also it can be used on other projects for loading data from different sources into any central database. This particular user needs to work at different sites, and normally collects plant data in the plant fields in a PDA, he then uploads the collected data into the main Access database after he comes back his office. He recognised that the Web service approach could help him to immediately upload the collected data to the main database from his mobile device.

Useful error messages
Both users thought the application provided useful error messages. They were able to correct error inputs based on the messages. One user said “The error messages are more friendly than VBA error messages”.

End user instructions
The Access user suggested that it might be useful to provide simple instructions or a training program for novice end users. We will provide simple user instructions to show how to use the application in the future. Most Germinate users will probably use one of the trial applications by following the user instructions.
5.2 Developer trial

A key data loading guideline for developers (see section 2.6.2) is to allow them to customise data loading applications for different data sources with minimal effort. To help achieve this goal, we developed client toolkits to interface to the Web service from different programming languages and platforms e.g. a VBA toolkit for Microsoft Office applications. Development instructions (see Appendix A) were written to assist application developers use the toolkits.

In this trial, we assess the usefulness and ease of use of the VBA client toolkit. The developer trial evaluated the usefulness of the VBA client toolkit and development instructions to customise an Excel application.

5.2.1 Evaluation methodology (for developers)

We provided developers with the toolkit in a standalone VBA module, and a typical Excel worksheet. The worksheet (shown in Figure 5-5) included Accession data in 24 columns, and was similar to the worksheet used in the end user trial. The developers were asked to create a complete data loading application following the instructions. Sample VBA code was provided in the instructions to show developers how to use the toolkit functions. The trial consisted of four tasks: importing the toolkit, initialising the Web service, setting the attributes of an Accession object, and calling the Insert data function. In addition, we prepared questionnaires based on the tasks to get detailed feedback from developers (see Appendix C).

![Figure 5-5 Excel data for the developer trials](image)

**Developer task 1: Importing the toolkit**

The developers had to import the toolkit module into the Excel worksheet. Both the toolkit module and the Excel worksheet were available on a server. The developers needed to download both to a local computer and follow the instructions to import the toolkit.
Developer task 2: Initialising the Web service

The toolkit provides a TK_initialisation procedure that prepares the application to use the Web service. The instructions describe the usage of the TK_initialisation procedure. Developers had to specify the Web server name to initialise the Germinate Web service. The developers were also asked to pass an incorrect Server name in order to test the error handling features of the TK_initialisation procedure.

Developer task 3: Setting Accession attributes

This required developers to add VBA code to create an Accession object, and set values for its 24 plant data attributes. They needed to collect values from the appropriate cells of the Excel worksheet, and assign them to the correct Accession object attributes. In addition, in order to load multiple records from the worksheet, developers had to write a loop to collect the source data from the each of a set of rows.

Developer task 4: Calling the Insert Data function

The TK_InsertAcc calls the Web service to load data from the Accession object (created in task 3) into Germinate. Developers had to pass the Accession object to the Tk_InsertAcc function. Developers also needed to display the response messages returned by the Web service (via the TK_InsertAcc) in another column of the worksheet.

5.2.2 Results

There were two developers (occasional VBA users) who completed this trial with minimal assistance. One user is a Web administrator who is familiar with PHP, JSP, ASP and HTML. He read the instructions and successfully developed a data loading application in about 30 minutes. Another developer is a geneticist who has basic programming experience with VBA, PHP and Perl. He did not use the provided worksheet, but instead one of his own project worksheets. His worksheet included 33 columns of data with some non Accession data, and the layout was different to the provided worksheet. He spent around 2 hours to successfully add data loading functions to his worksheet.

Both developers started by importing the toolkit into a new VBA project, and both of them successfully added code necessary for tasks 2 - 4. The type of code they needed to write is shown in Figure 5-6
**Figure 5-6 Typical code to call toolkit functions**

*Developer task 1 results:*
Both developers were able to successfully import the toolkit into a new VBA project following the instructions.

*Developer task 2 results:*
Both developers successfully called the TK_initialisation procedure following the instructions. The developers were also asked to try the error handling feature of the TK_initialisation. For example, one developer used the wrong web server name, and the error message “The server has not been found, please enter a correct server name” was displayed. The developers thought the error message was useful to help them dealing with the Web service initialisation problems.
Developer task 3 results:
Developers were able to successfully create the Accession object and set values for the object following the instructions. One developer (the geneticist) said that it would be very easy for anyone with basic VBA knowledge to do this.

One suggestion made by a developer (geneticist) was to provide a template with VBA code for mapping cells to the Accession object attributes (show in figure 5-7). Application developers could then quickly create an application by filling in the relevant cells.

```
With Accession
    .Nicode = Range(""")  ' insert cell values between ""
    .InstCode = Range(""")
    .AccNum = Range(""")
    .Collnmb = Range(""")
    . . .
End With
```

Figure 5-7 VBA template code for mapping cells

Developer task 4 results:
Developers were able to successfully load the Accession data into Germinate by using the Insert Data function. This function also returned Web service response messages, and then developers were able to display the messages in another column of the worksheet.

5.2.3 Summary of developer comments
The developers filled out the questionnaires (see Appendix C). Below is a summary of their comments.

The Web service data access operations makes client side development very simple. The developers thought that the Web service and the toolkit provided data access operations which were much easier than writing complex data access code on the client side. One developer said “The format, delivery and simplicity of this approach are such that an application would be straightforward”. That is to say, the Web service function hides the complexities of database access details, and the client toolkit minimises the effort required to use the Web service. This means the client application can be smaller and simpler. The developer also commented he was “very impressed with this approach as a way of
breaking down the complexity of providing database access.” The developers also noted that it would be very easy to maintain client applications, because there are only a few lines of code in the client applications.

The developers were also asked how difficult it would be to add data format converting functions (e.g. the various date and coding schemes transformations) to the applications. The developers said that they were confident they would be able to build functions to transform local data formats. For instance, they thought that it could be done by adding simple VBA functions to their applications. In addition, for the converting different coding schemes, they could apply Excel built-in functions such as the Vlookup function.
5.3 Discussion and future work

5.3.1 Loading data from different sources

The Web service allows developers to easily build applications for a variety of local data sources. The toolkits provide language specific functions to allow developers to easily customise different applications, and these customised applications help end users load data directly from data sources such as Access and Excel.

It is easy to provide the functionality to other platforms for loading data via the Web service. For example, some CFR users collect data in the field, and would like to load the data into Germinate from mobile devices such as cell phones. To meet this need, we have built a J2ME (Java 2 Micro Edition) prototype (shown in Figure 5-8) to simulate loading data from Java enabled mobile phones (Sun, 2004). This would be very useful as plant data is often collected in the field. However, security issues (see section 5.3.7) will need to be dealt with before we can carry out a trial of this application.

![Figure 5-8 Mobile data entry application prototype](image)
5.3.2 Simplifying data entry for users

The customised applications allow users to enter data in different formats, so they do not need to manually transform data to match the database requirements. In the trial, the users tried to load dates with different formats, e.g. year only (integer), string dates or Excel/Access dates. The date conversion method (in the toolkit) automatically converts these dates into the required format.

The Web service also supports optional fields to simplify data entry. That is to say, end users do not need to enter optional fields (e.g. geographical information of a plant sample) that are not necessary for plant research at CFR. For some data, we also set default values (specific for CFR) such as Biological status, Collecting source, and Storage. This flexibility can make the data entry much easier and quicker.

Although specifying required fields and default values in the Web service can simplify data entry for CFR users, this configuration may be not suitable for all institutions that use Germinate in different ways. For example, at CFR, 11 pieces of Accession data are required fields for inserting into Germinate. Other inputs, such as geographical information, are optional. However, each organisation might want to define different attributes as required fields and the requirement for default values may vary from institution to institution. In addition, at CFR, the default input value for Biological Status was specified as Breeding/Research material. However, other organisations might want a different default value. Currently, this would require modification of the Web service code. This is not a flexible way to configure the Web service.

We plan to provide a Web service configuration file (e.g INI or XML file), that will allow an organisation to define the required data elements and default input values. Moreover, an organisation might create multiple configuration files for different user groups to address diverse project needs. Developers could simply specify an appropriate configuration file from the client application. We could also provide a Web form for novice users to configure the Web service, i.e. to specify required fields and default values.

5.3.3 Encapsulating database implementation

It is easy to handle internal changes to the database, because the Web service encapsulates the database details. If there are database changes (such as changes to table names and so on), we will only need to update Web service code. The toolkits and client applications will not need to be changed. If new functionality is added to Germinate, (e.g. additional
fields), then both the Web service and client toolkits will need to be updated, but the changes to the client applications themselves should be minimal.

5.3.4 Error handling

During the trial, the toolkit Insert Data function returned response messages from the Web service as text only. The text nature of the messages could make it difficult for developers to customise the way they handle different errors. For example, if developers want to display different messages in different colours, they had to apply colours based on the text (highlighted line shown in Figure 5-9).

```
Range("AJ" & rowNumber) = Response
If Response = 'the Accession has been inserted' Then
    Range("AJ" & rowNumber).Font.Color = RGB(0, 0, 255)
    ...
End If
```

Figure 5-9 Error handing by check message string

To enable developers to more easily handle errors, we improved the Insert data function after the trial. Now, the Web service sends back a numeric code as well as a text response. The toolkit Insert Data Function only returns the code, but a new toolkit function was added to provide the text. The code provides a convenient way to handle different errors programmatically. Application developers can use the error code to determine different processing (shown in Figure 5-10). In addition, the TK_LastErrorText function returns the detailed error text that is suitable for displaying to end users if required.

```
ResponseCode = Tk_InsertAcc (AcceObj)
Range("AJ" & rowNumber) = TK_LastErrorText
If ResponseCode = 0 Then
    Range("AJ" & rowNumber).Font.Color = RGB(0, 0, 255)
    ...
End If
```

Figure 5-10 Error handing by catching error number
5.3.5 Performance
An important issue is the performance of the Web service. As middleware for data access, Web services impose additional overheads over direct database manipulation. Web servers and client applications must encode and decode native data into XML formats, which can add considerable time to the processing (IBM, 2005). The cross platform capability of Web services is paid for by increased processing time. Although Web services are slower than directly accessing the database, it is a reasonable trade-off for the increased flexibility of data loading from different data sources in this project.

In the trial, although end users were satisfied with the elapsed time, using a Web service to load data is much slower than direct database access. We used the PEAR Benchmark Module to monitor elapsed time of the Web service while inserting a sequence of Accession records from a PHP Web client. The client application and Web service ran on the same PC (Intel Pentium4 2.4GHz, 512M memory). The PEAR Benchmark recorded elapsed time to insert each record. The average Web service elapsed time for each record is about 0.3 seconds, compared with about 0.09 seconds for direct database access (see Figure 5-11). It is clear that inserting a record with our Web service is approximately 3 times slower than direct database access.

![Comparison of Elapsed Time](image)

**Figure 5-11 Comparison of elapsed time**
In the future, we will look at ways to improve the Web service data loading process. We will investigate compression of the XML messages for more efficient transmission. XML compression effectively reduces the size of the data, so applications need less time to exchange data between Web service clients and servers (Meier, 2004). This could also benefit VPN and mobile device users, because they normally have slower Internet connections.

We will also investigate bulk data loading to reduce the overheads of the system. Currently, every record loaded is treated as an independent transaction, which requires a Web service request and response. We could allow applications to submit multiple records for loading in a single Web service request. Minimising the number of requests could significantly reduce overheads in the Web service, client application and network, thus improving the overall speed of data loading (IBM, 2005).

**5.3.6 Reliability**

It is important to provide a reliable Web service. In the trial, each user inserted 500 records into Germinate without any problems. In other tests we have inserted more than 20,000 records via the Web service (including bulk loading a set of 2000 records) and there have been no problems. So far the testing appears to show that the Germinate Web service is reliable. However, reliability problems could emerge as the system is put into actual production use with a scaled up number of records and also with simultaneous transactions.

In classic database systems, all operations within a transaction must be either entirely completed or aborted (Date, 1990). This means that each table update must succeed or the entire sequence must be rolled back. For our case, the logical Insert Data operation generates several SQL operations to insert data into different tables. To protect database integrity, we have to make sure that all the Insert operations are carried out or all are rolled back. For example, if users submit 24 data attributes to load an Accession logical record we have to guarantee that the operation inserts the entire 24 data attributes or leaves the tables unchanged.

Currently, no explicit protection is provided for each logical transaction. To protect database transactions, we will implement ROLLBACK in the Insert Data function. This can be done via a BEGIN and ROLLBACK command around the entire logical transaction
(i.e. multiple SQL Insert statements). If there is an exception (e.g. database server crashes) during the transaction, the databases will rollback the logical transaction. For instance, if the Insert data function only inserts the first 10 input attributes (partially completed), and then the database server crashes, ROLLBACK will “undo” the inserted 10 attributes. Therefore, the Insert Data function will only insert 24 attributes as a complete unit in order to ensure data integrity. In addition, this approach can also be extended to protect simultaneous updating to the same logical record, e.g. two persons updating the same Accession record at the same time.

The use of rollbacks raises the issue of how to handle failed logical transaction. There are three possibilities to handle the situation of a failed transaction.

1. The Web service returns an error message with an error code so that developers can display text error messages to end users. For example, developers can inform users that “The submitted record has not been inserted due to an unexpected error. Please try again or contact the system administrator”.

2. The application retries the transaction: Developers can catch the error code, and then make the application retry the Insert Data function. This could be done a number of times or over a specified time period (e.g. 5 times or 30 seconds). If the transaction still fails, developers need to display the error message to users.

3. The Web service retries the transaction: The Web service still has the data from the logical record when a transaction fails. The Web service can directly retry loading data. This would be more efficient than the applications needing to resend and reload data. Eventually, if the transaction still fails after a specific number of attempts or period, the Web service will return an error message to the client toolkits.

5.3.7 Security

We have provided HTTP authentication to control access to the Web service. Currently there are no restrictions on inserting or updating data. Once an application authenticates to the Web service, full access is provided to all data in Germinate. In a production environment, we would need methods to restrict and audit user actions. Germinate already contains a table to record user names, passwords and user types. We plan to extend this through to the Web service to provide a set of security roles and to record each action against an authenticated user. Example roles could include:
• Data retrieval only
• Entry of specific types of data (e.g. Accession or genotype data)
• Modification or removal of existing data

Currently, the Web service is deployed on an internal CFR Web server that cannot be accessed directly via the Internet. In the future, the Web service needs to be available on the Internet for off-site users. To ensure the data transaction integrity and security, we could look at providing Secure Sockets Layer (SSL) support to the Web service and toolkit for encrypting input and output data. Alternatively, off-site CFR users could access the Web service via a secure VPN tunnel.

5.4 Summary

To evaluate the usefulness of the Web service approach for end users, we provided two sample client applications to test loading data from Excel and Access data sources. Users were able to load data successfully from these applications. Users were also able to correct errors in the source data based on returned error messages.

To test the effectiveness of developing applications using the Web service, developers were asked to customise an Excel application. Developers were able to create the application successfully following the development instructions, and they found that the approach was a useful and efficient way to customise data loading applications.

The Web service approach meets our data loading application guidelines developed in Chapter 2, which are to help end users to easily load data from local data sources, and to assist developers to easily create the applications to support this. The current version of the system has some limitations, such as configuration, performance, transaction and security issues. We will work to improve the Web service to provide an efficient, stable, and robust system.
Chapter 6 Conclusion

Loading of data from heterogeneous data sources and applications to a central database can be the most complex and time-consuming part of data integration projects. This is because the source data structures, types, formats and data representations may be quite different from the destination database. As a result, end users can find it very difficult to load their data without significant input from technical experts.

To develop appropriate data loading applications for end users, application developers have to spend considerable time to understand the schema of the target database, comprehend relationships between data sources and targets, and write complex data access code to facilitate data integration.

Although the developers of the target database may provide a data loading interface, it will often load data from only a single source (e.g. a web form or a spreadsheet template). However in many organisations, data is likely to be stored in a variety of different formats such as text files, spreadsheets, or databases. Users might require several different data loading applications to be developed for loading data from each individual source. To do that, developers must repeat the data access coding for each application, this can be time-consuming and error-prone.

6.1 Guidelines

We developed guidelines for data loading applications in order to simplify loading of data for end users (section 2.6). We indicated that data loading applications need to be customised according to users’ local data sources, formats and requirements. To minimise the efforts of application development for developers, we pointed out that the solution to customisation of data loading applications must be simple and not require developers to work with complex database management systems (below we discuss the guidelines further).

In order to meet our guidelines, we investigated using Web services as middleware to support data loading from a variety of data sources and applications. As an example, a Web service was developed to provide functions for maintaining data records in the Germinate plant database. We also provided language specific client toolkits to help developers invoke the Web service. The toolkit functions can easily be incorporated into a
variety of client applications to enable them to use the Web service to manage the database. In addition, sample client applications were provided to assist users to load data from Excel and Access data sources via the Web service.

We evaluated how well the approach met the guidelines via user and developer trials. The overall feedback showed that the approach successfully satisfied user and developer needs. The outcomes are summarised below for each of the guidelines.

### 6.1.1 End users

*Applications should be able to help users load data directly from their local data sources.*

The sample applications we provided allowed users to load data directly from their project data sources (Access and Excel sources). Users did not need to import or transform original data into a specific template or application for loading. This greatly reduced the effort required for end users to load their data.

*Applications should convert source data types and representation styles to match target database requirements.*

The sample applications handled original data entered in different formats (e.g. string dates or integer dates). It automatically converted the different data types to match target database requirements. This reduced the time for loading data and the potential for errors from manual conversions of formats.

*Applications should provide a user friendly interface.*

The data loading functionality was included in familiar user applications (e.g. Excel spreadsheets) to provide a user friendly interface. The familiar interface made it easier for users to understand and quicker to use.

*Applications should provide input data checking and pass useful errors messages back to the user as appropriate.*

The sample applications passed back clear error messages from the Web service regarding input data problems. Users found the messages simple to understand and were able to use them to correct errors. In addition, the use of a familiar application allows users to more quickly make any corrections required.
Applications should be able to allow omission of non-essential data and provide defaults if necessary for any missing data.

The sample applications support optional fields and provide appropriate default values. Users do not need to enter fields that are not necessary for their projects or those for which defaults are specified in the Web service. This made data entry easier and quicker.

**6.1.2 Developers**

Developers need tools which allow them to customise data loading applications for different data sources with minimal effort.

The middleware approach is a useful and efficient way to customise applications. With the Web service in place, individual applications can be easily customised for local data sources and user requirements. In addition, the client toolkits make it easy for developers to add data loading functionality to their applications. Generally only a few lines of code are required to successfully upload data via the Web service.

Developers need tools which insulate them from the details of the database as much as possible.

The Web service approach hides most of the internal details (schema, constraints, etc) and provides a simple “logical record” interface. CFR developers commented that “the approach makes database access amazingly easily”. Moreover, the Web service approach prevents any purely internal database changes from requiring changes to client applications.

Developers need tools which have as small a learning curve as possible.

The toolkits and development instructions we developed appeared to be very easy to learn. Both developers spent only a few hours to use the toolkit to complete a customised Excel application following the instructions.

**6.2 Additional guidelines**

This study appears to provide a very useful solution for data integration projects. However there are some limitations in the current system. After carrying out the evaluation, we consider that there is a need for additional work to provide a flexible, reliable, efficient and secure system, below we present additional guidelines:
Additional guidelines:

*Developers and database administrators need tools which allow them to change the configuration of the system*

Organisations use databases in different ways, and may want to specify different configurations, e.g. required input data and default values. It will be necessary to provide a simple way to allow developers and database administrators to specify varying configurations.

*Developers need tools which provide adequate processing performance.*

Adding middleware to a system imposes additional overheads over direct database manipulation. Investigation into improving the Web service performance should be undertaken (e.g. implementing XML compressions and bulk loading).

*Developers need tools which ensure database integrity.*

A Web service logical operation may generate a number of SQL sub-operations across different tables. We need to implement protection so that logical transactions are atomic, i.e. either succeed entirely or fail. We must also provide a mechanism to retry failed transactions or notify applications of failure.

*Developers and database administrators need tools which provide appropriate user roles and auditing methods to ensure database security*

Currently, there are no restrictions on inserting, updating or deleting data once an application authenticates to the Web service. The system needs to include a set of security roles and to record each action against an authenticated user. This will help prevent accidental or intentional pollution of the data.

### 6.3 Final remarks

This study looked at the issues involved in providing simple customised data entry applications for communal database systems. The approach used Web services as middleware, and provided language dependent toolkits. The approach was demonstrated with a real world example: loading data from multiple data sources and applications into the Germinate database. In addition, the approach was evaluated by user and developer trials at CFR. The feedback shows that the Web service approach was a very useful and efficient way for developers to customise client applications, and that these applications made data entry much easier and quicker for end users.
The study has made an important contribution to the research in data integration and management. The core contributions can be summarised as the development of the data loading application guidelines, and the Web service and toolkit solution.

- **Guidelines**: These provide a number of recommendations for developing better data loading applications from both the end user and developer perspectives. These guidelines can be used to guide the development of customised data entry applications irrespective of the software used.

- **Web service and toolkit solution**: The solution satisfied the original guidelines, and made it easier to add database functionality to a variety of applications. More importantly, it allowed the development of applications that can significantly reduce the time and effort for users to get their data into communal databases. The ideas developed here can be applied to any database to provide convenient data maintenance functions.

This study investigated simplifying data integration from a variety of different data sources and applications. The results of the user and developer trial show it is possible to provide tools to simplify the creation of customised data loading applications, and that these applications can greatly assist end users with data maintenance tasks. The importance of this work is best captured by this quote from one of the CFR users “it is absolutely crucial to simplify data manipulation and data inserting into main databases. When it is done, more people can use the applications.”
References


IBM. (2005). Quality of service for Web services: Managing Information Access to an Enterprise Information System Using J2EE and Services Oriented Architecture. Durham, NC USA: IBM.


Appendix A

Instructions for using the Germinate VBA toolkit (updated version)

After the developer trial, we updated the original instructions (that were used for the trial). Some changes were relatively minor but larger changes include the TK_insertAcc function (Section 2.3.2), and we added a new function TK_LastErrorText to display error messages (Section 2.3.3)

1 Introduction

The VBA toolkit simplifies the customising of data loading client applications (e.g. creating an application to load data from Excel, see Figure 1-1). The toolkit provides a set of local functions to access Germinate via a Web service. The benefit is that developers do not need to know how to use the Web service but can call the local toolkit functions e.g. Insert data, to directly access the Germinate database.

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>L</th>
<th>Response messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INSTCODE</td>
<td>ACCNUMB</td>
<td>COLLCODE</td>
<td>GENUS</td>
<td>SPECIES</td>
<td>CROPNAME</td>
<td>Insert Data</td>
</tr>
<tr>
<td>2</td>
<td>NZL027</td>
<td>ACC00000044</td>
<td>NZL027</td>
<td>Arena</td>
<td>sativa</td>
<td>sativa</td>
<td>Error: Invalid taxonomy</td>
</tr>
<tr>
<td>3</td>
<td>NZL027</td>
<td>ACC00000046</td>
<td>NZL027</td>
<td>Arena</td>
<td>sativa</td>
<td>sativa</td>
<td>The accession has been inserted</td>
</tr>
<tr>
<td>4</td>
<td>NZL027</td>
<td>ACC00000046</td>
<td>NZL027</td>
<td>Arena</td>
<td>sativa</td>
<td>sativa</td>
<td>Error: Invalid institution code</td>
</tr>
<tr>
<td>5</td>
<td>NZL027</td>
<td>ACC00000047</td>
<td>NZL027</td>
<td>Arena</td>
<td>sativa</td>
<td>sativa</td>
<td>The accession has been inserted</td>
</tr>
<tr>
<td>6</td>
<td>NZL027</td>
<td>ACC00000047</td>
<td>NZL027</td>
<td>Arena</td>
<td>sativa</td>
<td>sativa</td>
<td>Error: No accession number supplied</td>
</tr>
</tbody>
</table>

Figure 1-1 Data loading application

1.1 The toolkit overview

1. **Accession object:** This allows developers to set plant Accession data attribute values. e.g. Accession number, Institution code, Accession name.

2. **Initialisation function:** This allows developers to initialise the Web service.

3. **Database functions:** These allow developers to interact with Germinate via the Web service.

1.2 The toolkit workflow

Figure 1-2 shows how to use the client toolkit to invoke the Web service to insert an Accession record into the Germinate database. Developers collect users’ input data (e.g. from cells in Excel) into an Accession object and initialise the Web service. Then
developers can call the toolkit’s “TK_insertAcc” function for loading data (step 1). After that, the toolkit calls the Web service (step 2) in order to load data into Germinate. Eventually, the Web service sends responses back to the application via the toolkit (step 3 and 4).

![Diagram of the Client toolkit workflow]

**Figure 1-1** The Client toolkit workflow

### 1.3 Supported platforms:
Microsoft Office 2003 and Microsoft Office 2007

### 1.4 Requirements:
The Web service server and the Germinate database server must be accessible for client computers via a network. It is necessary to check the operational status of the database and the Web service before using the toolkit.

### 2 Toolkit

#### 2.1 Toolkit source code

The toolkit source code is distributed in two VBA module files, “accession.cls” and “toolkit.bas” respectively. Developers have to import these two VBA module files into a VBA project. The details about how to import the toolkit are introduced step by step in Section 3.

#### 2.2 Accession object

An Accession object encapsulates specific values for all 24 Accession data attributes as shown in Table 2-1. Developers need to collect the values from the Excel spreadsheets or Access tables, and assign them to the Accession object. Once a client application sets the accession data attributes properly, then the Accession object can be passed into the Insert Data function that loads the Accession data values into Germinate. The following table shows all Accession object attributes.
### 2.2.1 Accession object attributes

#### Table 2-1 Accession object attributes

<table>
<thead>
<tr>
<th>Attribute name</th>
<th>Type</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccNum</td>
<td>String</td>
<td>Accession number</td>
<td>√</td>
</tr>
<tr>
<td>InstCode</td>
<td>String</td>
<td>Institution code</td>
<td>√</td>
</tr>
<tr>
<td>AccName</td>
<td>String</td>
<td>Accession Name</td>
<td></td>
</tr>
<tr>
<td>Genus</td>
<td>String</td>
<td>Genus</td>
<td>√</td>
</tr>
<tr>
<td>Species</td>
<td>String</td>
<td>Species</td>
<td>√</td>
</tr>
<tr>
<td>Spauthor</td>
<td>String</td>
<td>Species authority</td>
<td></td>
</tr>
<tr>
<td>CropName</td>
<td>String</td>
<td>Crop name</td>
<td>√</td>
</tr>
<tr>
<td>Subtaxa</td>
<td>String</td>
<td>Subtaxa</td>
<td></td>
</tr>
<tr>
<td>SubAuthor</td>
<td>String</td>
<td>Subtaxa authority</td>
<td></td>
</tr>
<tr>
<td>Taxonomic_ Identifier</td>
<td>String</td>
<td>Taxonomic identifier</td>
<td></td>
</tr>
<tr>
<td>DonorNumb</td>
<td>String</td>
<td>Donor number</td>
<td></td>
</tr>
<tr>
<td>DonorCode</td>
<td>String</td>
<td>Donor code</td>
<td></td>
</tr>
<tr>
<td>CollDate</td>
<td>Variant/Date</td>
<td>Collecting date</td>
<td>√</td>
</tr>
<tr>
<td>CollCode</td>
<td>String</td>
<td>Collecting code</td>
<td></td>
</tr>
<tr>
<td>Sampstat</td>
<td>String</td>
<td>Biological status</td>
<td>√</td>
</tr>
<tr>
<td>Collsrc</td>
<td>String</td>
<td>Collecting sources</td>
<td>√</td>
</tr>
<tr>
<td>Collnumb</td>
<td>String</td>
<td>Collecting number</td>
<td></td>
</tr>
<tr>
<td>Acqdate</td>
<td>Variant/Date</td>
<td>Acquisition date</td>
<td>√</td>
</tr>
<tr>
<td>Storage</td>
<td>String</td>
<td>Storage</td>
<td>√</td>
</tr>
<tr>
<td>BredCode</td>
<td>String</td>
<td>Breeder code</td>
<td></td>
</tr>
<tr>
<td>DuplicateSite</td>
<td>String</td>
<td>Duplicate site code</td>
<td></td>
</tr>
<tr>
<td>Ancest</td>
<td>String</td>
<td>Ancestral data</td>
<td>√</td>
</tr>
<tr>
<td>Nicode</td>
<td>String</td>
<td>National code</td>
<td></td>
</tr>
<tr>
<td>Othernumb</td>
<td>String</td>
<td>Other number</td>
<td></td>
</tr>
</tbody>
</table>
2.2.2  Example of using an Accession object:

Sub createAcc()
    Dim Acce As New Accession  ' create a new accession object
    Acce.InstCode = "NZL027"  ' set values to accession attributes
    Acce.AccNum = "Accession001"
    Acce.AccName = "AccName001"
    'set remainder of attributes here
End Sub

2.3  Toolkit functions:

Currently, three functions are provided, a “Tk_initialisation”, a “TK_insertAcc” and a “Tk_LastErrorText” function.

2.3.1  Tk_initialisation procedure

**Synopsis:**

Public Sub Tk_initialisation (hostName As String, Optional userName As String, Optional password As String)

**Description:**

The Tk_initialisation procedure prepares the application to use the Web service. The developer must supply the Web service server name. For example, at CFR the Web service is installed at “genie.crop.cri.nz” server, so the host name parameter value is “genie.crop.cri.nz”. The developers can also specify user name and password (if HTTP Authentication is applied).

**Input Parameters:**

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>type</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>hostName</td>
<td>String</td>
<td>✓</td>
</tr>
<tr>
<td>userName</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td></td>
</tr>
</tbody>
</table>
**Output value:**
Tk_initialisation is a procedure that does not return any output, but it prints error messages via the VB message box, if the Web service initialisation failed.

**Example:**

```vba
Sub initialisation()
    Tk_initialisation "genie.crop.cri.nz", "Tom", "abc123"
    ...
End Sub
```

**2.3.2 **TK_insertAcc function:

**Synopsis:**

```vba
Public Function TK_insertAcc(AccObj As Accession)
```

**Description:**
The TK_insertAcc function passes the Accession object data (24 data attributes) to the Web service for loading. After the Web service processes the request and sends back a response, the TK_insertAcc function returns a result code along with a short text (shown in Table 2-2). Application developers can use the result code if they wish to take a particular action.

**Input Parameters:**
The input is the Accession object with attribute values collected from the Excel worksheets or Access tables.

**Output value:**
The “TK_insertAcc” function returns the result code from the Web service. The following table (Next page) shows all result codes and corresponding text messages.
Table 2-2 Response messages list

<table>
<thead>
<tr>
<th>Result code</th>
<th>Text messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The Access has been inserted</td>
</tr>
<tr>
<td>1</td>
<td>No Accession number supplied</td>
</tr>
<tr>
<td>2</td>
<td>The accession already exists</td>
</tr>
<tr>
<td>3</td>
<td>Invalid institution code</td>
</tr>
<tr>
<td>4</td>
<td>Invalid donor code</td>
</tr>
<tr>
<td>5</td>
<td>Invalid collecting code</td>
</tr>
<tr>
<td>6</td>
<td>Invalid breeding code</td>
</tr>
<tr>
<td>7</td>
<td>Invalid duplicate site code</td>
</tr>
<tr>
<td>8</td>
<td>Invalid taxonomy</td>
</tr>
<tr>
<td>9</td>
<td>Invalid subtaxa</td>
</tr>
<tr>
<td>10</td>
<td>Invalid biological status</td>
</tr>
<tr>
<td>11</td>
<td>Invalid collecting source</td>
</tr>
<tr>
<td>12</td>
<td>Invalid storage</td>
</tr>
<tr>
<td>13</td>
<td>Invalid date format (collecting date),please follow YYYY-MM-DD</td>
</tr>
<tr>
<td>14</td>
<td>Invalid date format (AcqDate),please follow YYYY-MM-DD</td>
</tr>
</tbody>
</table>

* Invalid data means: the input value is Lookup data that must exist in the related lookup table. Some Input values must be consistent with the Lookup data in Germinate database. For example, when users enter an institution code, if the code is not present in the Institution table of Germinate, the error messages returns “Invalid institution code”.

2.3.3 TK_LastErrorText function:

**Synopsis:**

```plaintext
Public Function TK_LastErrorText()
```

**Description:**

The TK_LastErrorText returns the text of the most recent error message returned from the Web service. These are in a suitable format for displaying to end users. The example shows how to use the TK_LastErrorText function (Figure 2-1 in next page) to display the text of error messages in a Message Box of the Excel spreadsheet.

**Output value:**

The “TK_LastErrorText” function returns the error text messages. (Shown in table 2-2)
Example of using **TK_insertAcc** and **Tk_LastErrorText** function:

```
Sub insertdata()
    WS_initialisation "genie.crop.cri.nz" 'prepare to use the Web service
    Dim Acce As New Accession ' create a new accession object
    Acce.InstCode = "NZL027" ' assign values to accession attributes
    Acce.AccNum = "Accession001"
    Acce.AccName = "AccName001"
    ...  
    ResCode = TK_insertAcc(Acce) 'insert the Accession data into DB
    'it returns result code
    MsgBox (TK_LastErrorText) 'display response/error messages
End Sub
```

Figure 2-1 Sample code of using TK_insertAcc and Tk_LastErrorText function
3 Application examples
3.1 Example 1: Load a single record from an Excel worksheet

Step 1 Create a new VBA project based on source data:
A typical worksheet is provided (Figure 3-1) on:
L:\SulfurDB\Endnote\OnionDatabases\Germinate\Germinate_WS\VBA_Toolkit

Figure 3-1 a typical record in an Excel worksheet
Open the VB editor from within the Excel worksheet.

Step 2 Add Web service references: Select the References option (under the VBA Tool menu) to open a dialog box (Figure 3-2). Then, add “Microsoft Soap Type Library” into the Reference list. The “Microsoft Soap Type Library” supports VB applications to call Web services.

Figure 3-2 References dialog box

Step 3 Import the VBA module files (“accession.cls” and “toolkit.bas”): Select the Import File option under the VBA File menu. Figure 3-3 shows how to import VBA modules into a VBA Excel project. Then, a new dialog box will pop up for importing the toolkit module files.
At CFR, the toolkit module files are placed at:
“L:SulfurDB\Endnote\OnionDatabases\Germinate\Germinate_WS\VBA_Toolkit”.

**Figure 3-3 Import toolkit code into VBA project**

**Step 4: Coding:** (sample code is shown Figure 3-4)

a) Initialise the Web service

b) Create a new Accession object

c) Pass values from worksheet cells to the accession object

d) Call the “TK_insertAcc” function to insert data

e) Call TK_LastErrorText to display response messages in MsgBox

```
Sub loadaccdata()
    Tk_initialisation "genie.crop.cri.nz" ‘a) initialise a Web service call
    Dim Acce As New Accession
    Acce.AccNum = Range("A2") ‘b) create a new Accession
    Acce.InstCode = Range("B2") ‘c) get user inputs data from cells
    Acce.AccName = Range("C2")
    ‘. . . can add more attributes here
    rescode = TK_insertAcc (Acce) ‘d) call insert data function
    MsgBox (TK_LastErrorText) ‘e) display response messages
End Sub
```

**Figure 3-4 Sample code**

Now, we have completed a simple data loading application. Developers can run the code, a message box will then pop up to show the Web service response message.
3.2 Example 2: Load multiple records from a worksheet

Example 1 shows loading a single record into Germinate from a worksheet. Example 2 demonstrates how to load multiple records from a typical worksheet (Figure 3-5) into Germinate. The worksheet is provided on:

L:\SulfurDB\Endnote\Onion Databases\Germinate\Germinate_WS\ExcelTemplate.xls.

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>D</td>
<td>G</td>
<td>H</td>
<td>L</td>
<td>M</td>
</tr>
<tr>
<td>2</td>
<td>NZL027</td>
<td>ACC000035</td>
<td>Avena</td>
<td>sativa</td>
<td>oat</td>
<td>AS93183,28</td>
</tr>
<tr>
<td>3</td>
<td>NZL027</td>
<td>ACC000036</td>
<td>Avena</td>
<td>sativa</td>
<td>oat</td>
<td>AS93183,29</td>
</tr>
<tr>
<td>4</td>
<td>NZL027</td>
<td>ACC000037</td>
<td>Avena</td>
<td>sativa</td>
<td>oat</td>
<td>AS93183,30</td>
</tr>
<tr>
<td>5</td>
<td>NZL027</td>
<td>ACC000038</td>
<td>Avena</td>
<td>sativa</td>
<td>oat</td>
<td>AS93183,31</td>
</tr>
<tr>
<td>6</td>
<td>NZL027</td>
<td>ACC000039</td>
<td>Avena</td>
<td>sativa</td>
<td>oat</td>
<td>AS93183,32</td>
</tr>
</tbody>
</table>

Figure 3-5  A typical worksheet with multiple records

**Step 1 Create a new VBA project:** (details described in Example 1)

**Step 2 Add Web service references:** (details described in Example 1)

**Step 3 Import the VBA code modules** (details described in Example 1)

**Step 4 Coding:** (sample code is shown Figure 3-6 on next page)

a) Initialise the Web service

b) Create a new Accession object
c) Start a Loop
d) Map data values from worksheet cells to the Accession object
e) Call the TK_insertAcc function to insert data
f) Call TK_LastErrorText function to display responses on Column AJ
g) End the Loop
Sub loadaccdata()

Tk_initialisation "genie.crop.cri.nz"  ' a) initialise a Web service call

Dim Acce As New Accession  'b) create a new Accession object

rowNumber = 2  ' c) start a Loop
Do While Len(Range("A" & rowNumber).Formula) > 0
  With Acce
    .Nicode = Range("b" & rowNumber)
    .InstCode = Range("c" & rowNumber)
    .AccNum = Range("d" & rowNumber)
    .Collnumb = Range("e" & rowNumber)
    .CollCode = Range("f" & rowNumber)
    .Genus = Range("g" & rowNumber)
    .
    .
    End With
  rescode = TK_insertAcc(Acce)  ' d) Map values to accession object
  Range("AJ" & rowNumber).Value = TK_LastErrorText  ' e) call insert Accession function

  rowNumber = rowNumber + 1  ' f) display response in AJ
  Loop
  ' g) end the loop
End Sub

Figure 3-6 Sample code for bulking loading

<table>
<thead>
<tr>
<th>INSTCODE</th>
<th>ACCNUMB</th>
<th>ACQDATE</th>
<th>Response messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>NZLC02</td>
<td>ACC100365575</td>
<td>11/12/2000</td>
<td>The accession has been inserted</td>
</tr>
<tr>
<td>NZLC02</td>
<td>ACC100365576</td>
<td>212324</td>
<td>Error: Invalid date format (AcqDate), please follow YYYY-MM-DD</td>
</tr>
<tr>
<td>NZLC02</td>
<td>ACC100365577</td>
<td>1999</td>
<td>Error: Invalid institution code</td>
</tr>
<tr>
<td>NZLC02</td>
<td>ACC100365578</td>
<td>2000</td>
<td>The accession has been inserted</td>
</tr>
<tr>
<td>NZLC02</td>
<td></td>
<td>1991-12-1</td>
<td>Error: No accession number supplied</td>
</tr>
</tbody>
</table>

Figure 3-7 Bulk loading result
Now, we have completed a simple bulk loading application. After running the code, the Web service response messages are displayed in Column AJ (as shown in Figure 3-7). There are error messages for rows 3, 4 and 6, because the input data has some invalid values. The TK_LastErrorText function displays the error messages to help the user to correct the input data, e.g. “Invalid institution code” is displayed on row 4.

4 Trouble shooting

4.1 Web service initialisation fails

If Web initialisation fails, error messages are displayed in a Message box.

![Figure 4-1 An Error message caused by a Web service initialisation problem](image)

Solutions:

1. Developers should check whether the Web service server is available on the network. If it is not available, they should contact the Web master to make the server accessible via the network.

2. If the Web service server requires authentication, you should set the correct the username and password in the TK_initialisation function.

```
Tk_initialisation “genie.cri.co.nz”, “username”, “password”
```
4.2 Germinate database server does not respond

The TK_initialisation function can also detect the Germinate database operational state. If Germinate does not respond to Web service requests, an error message will pop up to inform the user of this problem (Figure 11).

![Error message](image)

**Figure 4-2 Error message about Web service initialisation problem**

**Solution:** Developers should contact Germinate database administrators to solve this problem.
Appendix B

Evaluation form for users

Name:                             Job Title:

Experience of using Excel/Access:

New
Occasional
Often

● Step 1: Insert a record :
(The sample data are provided in another worksheet, the data layout is the same as the data loading sheet, and users can copy the sample data, and simply paste them in the data loading sheet)

1. Did you insert this record successfully?

2. Could the feedback messages inform you of the inserting status?

3. Problems or commons
● Step 2: Give them 12 records that include invalid data

1. Could the error messages help you to identify invalid data?

2. Did you understand the meaning of these error messages?

3. Could you correct the invalid data based on error message?

4. Problems or comments

● Step 3: Test performance by inserting 500 records

1. Was the response time acceptable?

2. Problems or comments
● Reliability

Were there any problems during the testing?

● Was it easy to insert data by this approach?

● Would it be useful?
Before people have to send data to John, John fixes the data and load into Germinate.

If we provide such applications, how confident are you to loads it data by yourself?

● Accessibility

Insert data via internet (VPN)
Appendix C

Evaluation form for developers

Name:                             Job Title:

Experience of using VBA:

New
Occasional
Often

Experience of Web programming:

● Step 1: Import toolkits:

1. Were the instructions clear showing how to import the toolkit?

2. Did you import the Web service toolkit successfully?

3. Was it easy? any problems / comments
• Step 2: Connection to the “Web service"

Public Function WS_connect(hostName As String, Optional userName As String, Optional password As String)

1. Were the instructions clear showing how to use TK_initialisation() function?

2. Did your program connect to the Web service successfully?

3. Did the toolkit pass back useful error message?

4. Was it easy? any problems / Comments

• Step 3 : Setting Accession attributes

1. Were the instructions clear showing how to set values to accession attributes?

2. Did your program create an Accession object and set values successfully?

3. Is it easy? Any problems / comments
Step 4 : Insert data function

Public Function TK_Acc_insert(AccObj As Accession)

1. Were the instructions clear showing how to use the TK_Acc_insert function?

2. Did you insert data by the Web service successfully?

3. Did the toolkit pass back useful error messages when invalid data was detected?

4. Is it easy? any problems /Comments

Instruction documentation

1. Were the instructions clear / complete?

2. Was the sample code useful to build your application?

3. any problems /Comments
Toolkit Code

1. Is it easy to understand?

2. Any problems /Comments

Flexibility/Customizability/ Interoperability

1. Would you find it is easy way to customise your applications?

=> How confident are you to apply the Toolkit to another situation (e.g. different data layout)

2. Would you find it is easy to add new data transformation functions according to your local original data?

E.g. Transform original data into a required format.

Transform lookup data between different presentation styles.

3. Any problems /Comments
● Security

1. What security access issues do you see with this approach?

2. Any problems /Comments

● Maintenance /Deployment

1. Do you think this is a useful approach for developers to customize an end user application?

2. Do you think it is easy to maintain the client application code?

● Any comments