INTERNATIONALIZATION AND LOCALIZATION

FOR CHINESE SOFTWARE

By

HANGBO ZHANG

Bachelor of Engineering

Hunan University

ChangSha, China

1991

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
In Partial Fulfillment of
The Requirements for
The Degree of
MASTER OF SCIENCE
July 2000

INTERNATIONLIZATION AND LOCALIZATION

FOR CHINESE SOFTWARE

Thesis Approved:

H. Lu

Thesis Adviser

J Chandler

Paul Cubbill

Dean of the Graduate College

ACKNOWLEDGEMENTS

I sincerely thank my thesis advisor, Professor Huizhu Lu, for her intelligent supervision, constructive guidance, warm encouragement, and valuable time she has given me toward the completion of my thesis. My sincere appreciation extends to Professors Chandler and Nohpill Park for serving on my committee; their guidance, encouragement, assistance, and friendship are invaluable.

I would like to give my special thanks to my wife, Xinyue Zhu, for her love, encouragement, patience, and understanding throughout my study at Oklahoma State University. My respectful thanks go to my parents Binwen Zhang and Shiying Peng for their love and encouragement.

Finally, I would like to thank all the faculty of the Department of Computer Science for their support during my two and half year study here.

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

The largest software markets in the world have been the United States, Europe, and Japan. In recent years, China's economic expansion has created many software users and opened a huge software market. Additionally, the Spanish-speaking population using software products has also been increasing quickly. Software products are now expected to be available in the native languages of its customers. On the surface, this may look like a simple process, as some people would say, "It's easy, we just need to translate the English version into another language". But in fact, the issue is much more complicated. Every local market has its specific requirements, from both a cultural point of view and a technical point of view.

1.1 Motivation

During the 1990's, China has gradually become a sizable software market, partly because of the global information revolution. The Chinese society has quite different cultural traditions from the West. These traditions also differ among the Asian countries such as Japan and Korea. This thesis deals with the concepts of internationalization (also called globalization) and localization, as well as the associated technologies of encoding character sets. It will discuss how programs can be designed and coded to minimize the challenge of converting a software product from the English edition into, say, Chinese.

1.2 Objectives

The objectives of the thesis are to study the major approaches to software

internationalization and localization, to implement two applications in C++ and Java and finally to show in detail how to internationalize software products in general, and how to localize software products to Chinese in particular.

1.3 Organization of the Thesis

The thesis consists of six chapters, listed below:

- Chapter 1: Introduction
- Chapter 2: Literature Review
- Chapter 3: Internationalization Models
- Chapter 4: Localization
- Chapter 5: Application Design and Implementation
- Chapter 6: Summary and Future Work

Chapter II

LITERATURE REVIEW

2.1 Internationalization and Localization

Whenever a software program is written, it is usually localized to the culture and language of the programmer, mainly because that is what the programmer is familiar with. According to Dave Taylor [1], it is hard for software in a foreign language to succeed in the international marketplace where it is sold; applications and the underlying operating system need to work in the language of the user, not the developer. Basically. the users' need is the engine to run the business. This is the reason why we need to globalize and localize software products.

- Internationalization (or Globalization)

Globalization/Internationalization is the set of code design and changes that are made to ensure that a software product or a Web-site can be adapted to various local languages and can be used by customers in different countries. It's the process of developing a program (or Web-site) whose design features and code implementation aren't based on a single language. The goal of internationalization is to give users a consistent look and functionality among different language versions of a software product. Users may expect the localized software to have the same basic features as the native language version . To achieve this goal, it is necessary for developers to design and code the program so that it can be smoothly converted from one version to another. Obviously, a company will save money and time if their applications are designed

internationally from the beginning.

The following is a brief discussion of the major issues encountered by developers when producing an internationalized software product.

- Character Set

  A character set is, intuitively, just a collection of symbols which can be used to build languages, which are then used by humans and computers. Before the Double Byte Character Sets (DBCS) and Unicode (explained in section 2.2) were introduced into English programs, the programs generally used the ASCII set, which means that only 256 symbols can be used; they include the 26 upper and lower case English letters, commonly used punctuation marks, and certain other symbols. Using the ASCII set alone is clearly not enough if the software product is to be internationalized. For example, even the most basic collection of Chinese characters is in the thousands. Nowadays, several DBCSs, each of which contains several thousand symbols, are supported on all the Windows operating systems. Additionally Unicode, which can potentially contain over sixty thousand characters, is supported on the Windows NT operating system. It is now possible to create any language versions of a software product with the appropriate choice of one or several character sets.

- Sorting Methods

  In a Latin language environment, we sort text strings according to the order of the underlying alphabet. This clearly has to be done differently in other languages. For example, in the Far East edition of Windows, several different sorting methods are supported: kanji characters in Japanese are sorted by the order of the radical,

and Han characters can be sorted by the number of strokes used in characters.

- Bitmaps, Icons and Sounds

Using bitmaps and icons in a program to identify certain features is indeed a creative activity. However, when converting a software product from one language edition to another, it is generally a waste of time to recreate those same pictures. Therefore, programmers need to consider the portability of bitmaps and icons when designing them. For instance, we at least need to consider using colors and graphics that will not violate the target market's culture. Fortunately, many programs can take advantage of the common toolbar defined in Windows 95/98 and those international symbols. Using portable symbols in the program is an important step in producing global software. As for the use of sounds in programs, obviously the less the better, since we need to translate them for each localized version of the software.

- Menus and Dialogs

Menus and Dialogs are essential elements in any Windows-based application. When text in Menus and Dialog boxes are translated from English to another language, their layout length will often change. Programmers typically need to allow more space for menu bars, toolbars, and dialog boxes which contain text strings or are used as text edit boxes, just in case later translations of these text strings expand in length.

- Currency Symbols and Number Separators

Different countries use different currencies, and each currency has its own representative symbol. Similarly, each culture seems to have its own way of writing

long numbers and decimals. In the US, people write numbers in the form 34,567.89 and they immediately recognize the meaning of the comma and the period in the expression, while other cultures may well have different ways of presenting the same number. During programming, we need to avoid putting characters such as currency symbols and decimal separators in the implementation files. Instead, the resource files are better places to hold them.

- Date Format and Calendar

Again, different countries tend to use different date formats. For example, when dealing with date related issues, a typical software package in the US should interpret the expression 1/2/99 as January 2nd, 1999, while an application in England should understand it as February 1st, 1999. This date expression will probably produce an error message in a Chinese software package, because the date format in China is year/month/day.

- Hard-coded Literals

In windows programming with C/C++ and Java, if a programmer puts a text literal (such as a warning message) in one of the implementation files: *.c files, *.h files, *.cpp files, and *.java files, the practice is called hard-coding, and the text literal is said to be hard-coded. Since a commercial software product consists of hundreds, if not thousands, of files, it will be extremely difficult to locate those text strings for translation if the software is to be localized to a particular foreign language edition. Normally, in a properly internationalized C/C++ Windows-based application, there will be one or several files (called resource files) to collect dialogs, menus, toolbars, bitmaps, icons, and text strings used in the program. For

Java applications, those features are stored in properties files for different locales. When localizing a product, developers need only let translators access those resource or properties files. Hard-coded text strings, if not discovered in the translation process, will clearly produce undesired results during software run time, when the users of the translated programs cannot understand the displayed strings. Even if the translator is allowed to access source files, he or she may not be a programmer and therefore will have trouble distinguish between a hard-coded string (which needs translation) and a key word (which does not need translation). Thus the common practice to avoid hard-coding is to give a unique identifier to each string needed in the program, and then store them accordingly in the resource files. The source files then only use those identifiers.

- Non-ID strings

As we mentioned earlier, a careful Programmer assigns a unique identifier (ID) to each text string needed in the implementation files. There are also strings which are just displayed on the panel as static labels. The IDs for such strings are given automatically by the Visual C++ Resource Editor, like IDS_STATIC. This happens in Windows API (Application Programming Interface), too. Programmers do not pay much attention to assigning IDs in this situation. When writing global programs, this may cause problems in the localization process. The author will discuss the possible solutions in a later chapter.

- Political and Cultural References

To internationalize a software product, we need to identify not only the language-dependent elements, but also the culture-dependent and politics-dependent

elements. It happens often that software developers impose their own values and ideals upon the customers in foreign countries where applications and operating systems are sold. One needs to be particularly cautious with politics, as in some countries, politics or reference to the government is a very sensitive issue. Insensitivity in this area could present serious obstacles to shipping the products.

- Localization

In contrast to internationalization, localization is the process of adapting a program for a specific locale. It includes translating the user interfaces, resizing the dialog boxes, customizing certain features (if necessary), and testing the program to ensure that it works well in the desired locale. The goal of this process is to better match the product with the target culture and marketplace.

Nadine Kano [2] discusses seven increasing levels of localization, which are listed below:

1 Translate nothing

2 Translate documentation and packaging only

3 Enable code,

4 Translate software menus and dialogs

5 Translate online help, tutorials and samples and README files

6 Add support for locale-specific hardware

7 Customize feature for locale

Obviously, the higher the level of the localization, the more the cost of the development, both in time and money. For this reason, it is important that international software is designed properly right from the beginning. When the programmers take

internationalization into account during the process of planning and developing, localizations can then be done very effectively; the relationship is shown in Figure 2.1. In particular, software internationalization and localization should not be the step after designing, coding and testing, but should be one of the guiding principles throughout the development cycle.

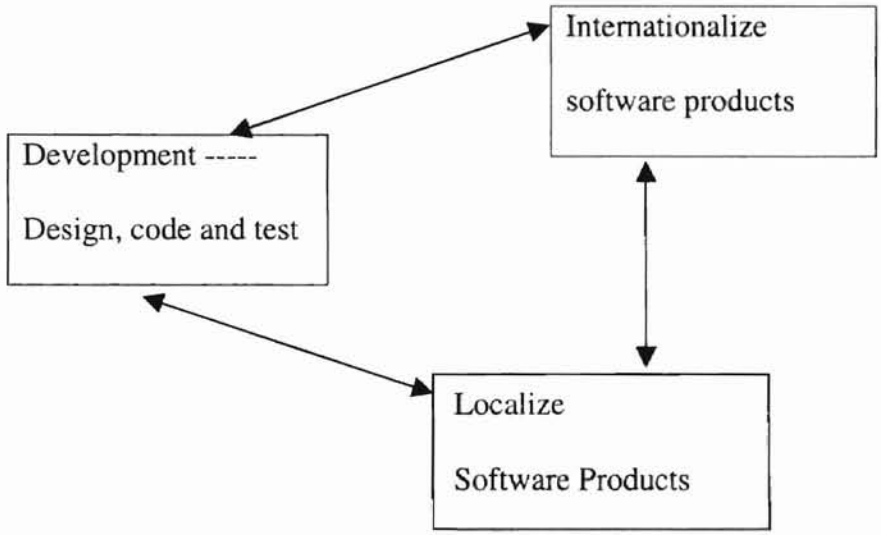

Figure 2.1 Internationalization and Localization in the Development Cycle

2.2 Encoding Character Sets

2.2.1 ASCII

ASCII is the acronym for American Standard Code for Information Interchange. In this standard, the numbers from 0 to 255 represent Latin letters, numbers, punctuation marks and other characters. The ASCII code was designed to facilitate transmitting text between computers, or between a computer and a peripheral device. Since it only contains 256 characters, it can not represent all the languages in the world.

2.2.2 DBCS

DBCS is the acronym for Double Byte Character Set. Double byte character sets were created to handle some East Asian languages that use ideographic characters such as Chinese, Japanese, and Korean. A character in a DBCS is usually represented by 2 bytes, or 16 bits. With 16 bits we can represent 65,536 characters, although far fewer characters are defined for the East Asian language. For instance, the Han character set today includes about 18,000 characters. In the areas where DBCSs are used — such as China, Japan and Korea, both single-byte and double-byte characters are used in the character set. The single-byte characters used in these areas are almost identical to the ASCII character set. For double-byte characters, the first byte is called the lead byte, the second called the trail byte. Non-DBCS-enabled software may behave strangely: the cursor can be put in the middle of a character, resulting in selecting half of a character or deleting half of a character. Thus it is necessary to DBCS-enable a program that handles, say, Han characters. Alternatively, we can have the program Unicode enabled, as discussed below.

2.2.3 Unicode

The Unicode standard was produced by an industry group called the Unicode Consortium, and has been adopted as the standard character encoding specification by the International Standard Organization (ISO). The Unicode is a character-encoding scheme that uses 2 bytes for every character. It uses the numbers from 0 to 65335 to represent characters or symbols from every language in the world. In fact, the value range has not run out, there are still some empty slots for later character creations. Unicode is fully supported by Windows NT. Unfortunately, operating systems earlier than Windows NT do not fully support Unicode. This is the reason that the DBCS technology still exists as a method of coding character sets. Some people do not like to use Unicode since a single

Latin character will occupy 16-bit of space instead 8 bits and they do not want to expand the extra disk space for their applications. The author will thus implement a Windows-based application using DBCS programming techniques.

2.3 Microsoft Visual C++ Studio 6.0 ™ and Microsoft Foundation Classes (MFC)

The author will use the Microsoft Visual Studio 6.0™ as the development environment to implement some Windows-based applications, and use the Microsoft Foundation Classes to help the implementation. The Microsoft Foundation Classes (MFC) form an "application framework" for programming in Windows [3]. Written in Microsoft Visual C++ Studio 6.0™, MFC provides much of the code necessary for managing windows, menus, dialogs boxes, performing basic input/output, storing collections of data objects, and so on. Generally speaking, the classes in MFC help create a skeleton for an application; programmers can then design the user interface elements visually through the resource editor. We can build the text implementation files and the resource files together into executables or Dynamic Link Libraries (DLL); we can also build them into separate DLLs. The Visual C++ compiler provides users with the ability to handle MFC code, it has the "symbiotic" relationship with MFC [4]. It is a very useful tool for developing Windows applications.

2.4 The Generic Approach to Designing a Global Software

Internationalizing software involves designing a user interface and a code base that are generic enough to apply to most of the product's target language editions, they are developed independently of the countries or languages of their users, and then localized for multiple countries or regions. Of course, some customization changes may be necessary during localization, but the fewer the changes, the better the initial design.

An important prerequisite to creating an internationalized code base for an application is that all language editions share the same source files. Maintaining separate source files for different language editions of the same product is error prone and unnecessary for code that has been properly internationalized. The following steps are usually taken when developing international software:

2.4.1 Planning an International Application

Programmers may need to have a written design specification before any code work begins. In this step, programmers take the internationalization issues into consideration. First, the product feature details need to be made clear. For instance, the function calls in the Japanese edition to handle the ideographic characters may not be supported by the US edition, so we need to specify the entry points for those functional calls, allowing the program to run properly in different languages. Second, the languages and locales to which the final product is targeted need to be known in advance, so that the developing team can address the international issues related to different locales. In addition to those, Nadine Kano [2] suggests that the developing team communicate with the marketing representatives and language specialists to make the specification parts related to the culture concise. Planning ahead reduces the cost of developing an international application rather than modifying it for international use later on.

2.4.2 Customizing Features

A well-designed international product minimizes the need for customizing features for different language editions. Some features might be essential for all language editions, yet they might need different implementation (or different algorithms). For example, there is no spell-checker for the Chinese version of Microsoft Word™ since the spelling

rules are too complicated. To implement a spell-checker in the Chinese version may not be worth the expense. Obviously, in this situation, it is important to create code that is flexible for programmers to add, remove or to customize functionality.

Designing an international user interface is also included in this step. Nadine Kano [2] suggests two rules: first, use simple generic graphics, and second, avoid crowding those graphics. The Windows Interface: An Application Design Rule [11] provides lots of recommendations for designing consistent interfaces which also meet the requirements for international markets. Tony Fernandes [5] gives readers some design rules in his book. From the internationalization point of view, the GUI design consists of two parts. The first part is designing the visual elements: bitmaps, icons, sounds, menus, dialogs; The other part concerns the language and locale related factors, such as date formats, currency formats, address formats, keyboard layouts, measurement systems, number formats, time formats and paper size. In this step developers also need to consider legal issues both in US exporting laws and laws in the foreign market.

2.4.3 Setting Up a Development Environment

An application consists of a user interface component and an application component [8]. The user interface component contains graphics, text strings and settings for various locales. The application component includes the source code which can be run for all the language editions. The application component processes the user-interface component. To make the application easier to maintain and localize, we normally separate the application source code files from user interface components. Therefore, we don't have to browse the source code to localize interface elements. This relationship is shown in Figure 2.1

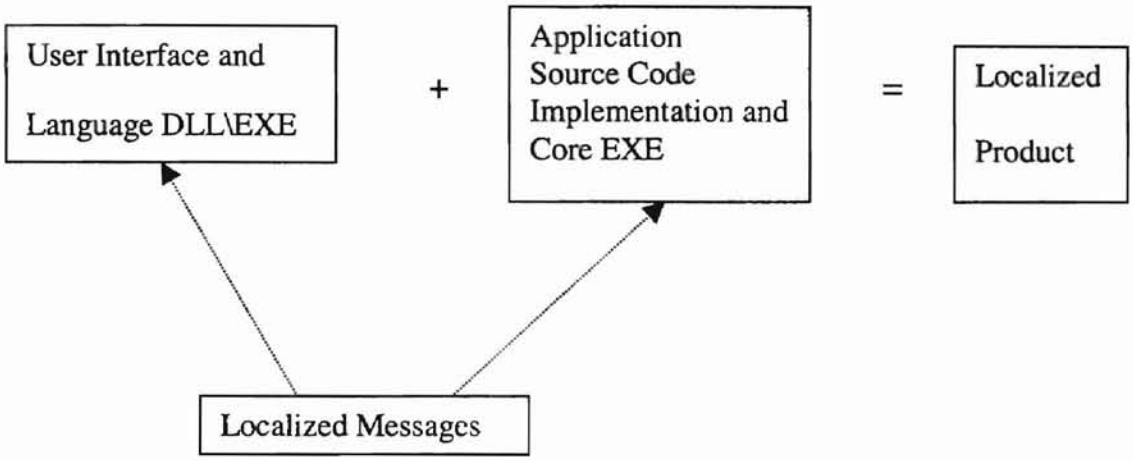

Figure 2.2: A Localized Product

The major program elements that need to be isolated from the resource files include certain algorithms for different locales, as well as dialogs, menus, prompts, sounds, status bars, and toolbars. Sometimes developers put message files in the language DLL/EXE files, or link localized messages to the Core EXE and Language DLL/EXE at run time as shown in Figure 2.2. On the other hand, putting strings or character literals in the main body of implementation source files (like .c, .cpp, .h file in C/C++ environment or .java file in java environment) may cause the so-called hard-coding problems as discussed previously. In order not to recompile the source code every time we build different language editions, putting the localizable resource for windows-based applications in windows resource files is the simplest way to set up an international application [2]. When we build the application, we can build the resource file into Dynamic Linked Libraries (DLLs), but these DLLs must be included in the same physical memory space of the source code execution files. Dale Rogerson [9] illustrates another

general way of dealing with resource files, that is, we can use the so-called Component Object Model (COM) technology to build the Core EXE and the resource files as a Client-Server relationship. In a COM-based EXE server part, we can use the resource across different processes. No matter which format we use for resource files, either EXEs or DLLs, we need only build the resource DLLs or register the EXEs when building the whole application, and that makes localization easier. Issac Varon [10] presents ideas to create resource DLLs for MFC applications. From the above discussion, we may have a file structure shown in Figure 2.2. Developers will focus only on the native version resource files (except certain algorithms), and translators complete the other language specific resource files.

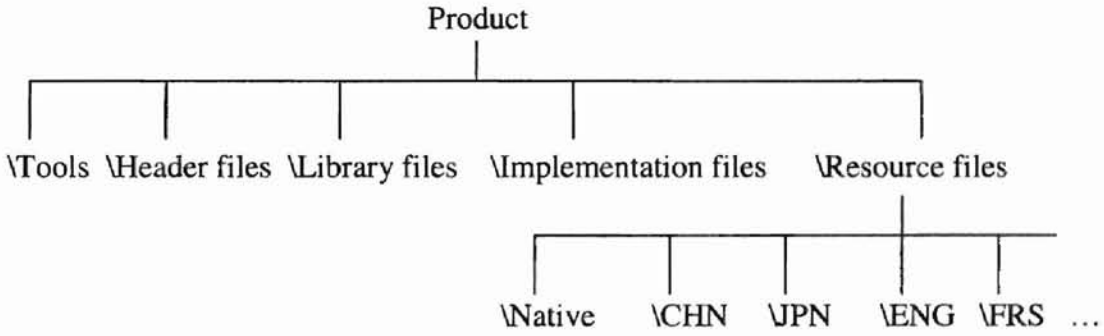

Figure 2.3: An Example of C/C++ File Directory

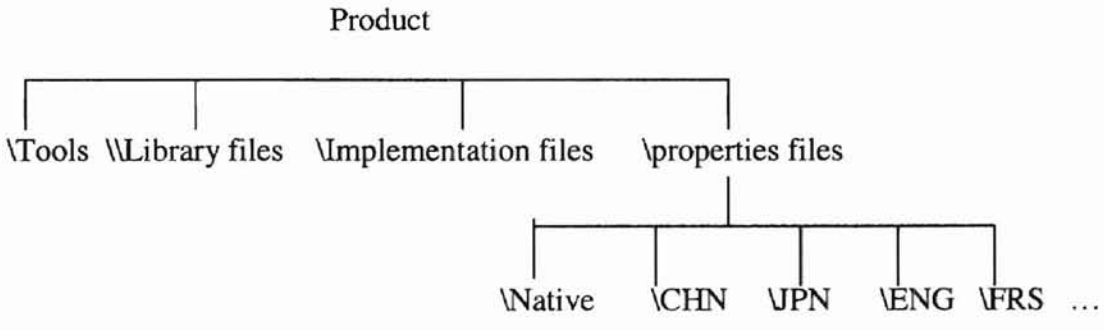

Figure 2.4 Java Application File Structure

Java applications have almost the same file structure as Windows programs do; Java applications use properties files instead of resource files. See Figure 2.4.

2.4.4 Coding

In real world programming, we have to deal with many intricate issues related to locale changes. Avoiding hard-coding localizable elements is the first issue that developers need to keep in mind. When dealing with internationalization, ordinary methods consist of eliminating compile dependencies, making buffer size large enough to hold translated text, using Unicode/DBCS techniques, etc. Some tricks may be related to specific projects. For example, developers may decide whether to use smart pointers or wrapper classes when working with COM-based EXEs or DLLs. The coding step is inevitably the most difficult part in global software programming.

2.4.5 Testing

Just like we should consider internationalization early in the product cycle, so should we consider testing of localized applications. Kory Srock in his article [7] discusses some general testing methods, such as extended character testing, DBCS testing and Pan-European testing. Basically, the checklist consists of the interface part and the functional part. The testing step always goes along with the developing cycle.

CHAPTER III

INTERNATIONALIZATION MODELS

Two approaches to internationalization are used in the industry today; the compile-time internationalization model and the run-time internationalization model. each used in different situations. For small-sized products, or products that are mainly sold in English-speaking countries and areas, especially when the companies' main focus is the market in those areas, compile-time internationalization may be appropriate. In this case, less time and money will be spent on training the developers how to work on a generic code base which can be used for different language versions. When products are sold to non-English-speaking areas to achieve a high level of profit margin, then run-time internationalization is probably the way to go, even when it means that higher costs will be incurred in educating the programmers how to build an internationalized code base for all the language versions; in the long run, companies still save time and money. The decision of choosing which model to use depends on the concerns about efficiency as well as the release date for the major market.

3.1 Compile-Time Internationalization Model

Roughly speaking, the compile-time internationalization model goes like this: "If you want it in Chinese, then write it that way". This is a reasonable model for developing some software. Obviously, it is easy for programmers. Two examples are given in the following.

3.1.1 Examples

```
void print_hi(void)
{
/* say "hi" on the printer, in the appropriate language */
FILE *printer;
If((printer=fopen(PRINT_DEVICE, "w")) == NULL) {
fprint(stderr, "could not open printer for printout\n");
        exit(1); //problem: when error, the prompting text is English.
}
#if LNAGUAGE=FRENCH
//can add other choices like French etc.
  fprint(printer, "baur");
        #else
        fprint(printer, "hi");
        #endif
        fclose(printer);
}
```

Another version isolates all the language and culturally sensitive elements into different portions of code to improve modularity.

```
void print_hi(void)
{
        FILE * printer;
        printer = open_file(); //get FILE*
        say_hi(printer);//language choices
        close_printer(printer);//close
        Exit(0);
}
```

3.1.2 Advantages and disadvantages of this approach

The advantages of this approach are that it offers the programmer the ability to have complete control of all aspects of the software, and there are no mysterious invocations of routines that might or might not work correctly. However, when using this approach for each locale, we need a different language version, and we need to modify the source code for each version, which is error prone. Moreover, we need to recompile the program and rebuild the executable file for each and every locale. All the modifications result in a lot of wasted time and money, which will drastically delay the product release for different locales.

3.2 Run Time Internationalization Model

This is the main approach for internationalization. The program source code is separated from the locale-related resources, one executable file can be used in different locales with different resource files, which can be DLLs or be contained in EXEs.

3.2.1 An Example

To compare with the Compile-Time Internationalization Model, we rewrite the previous example as follows.

```
void print_hi(void){
FILE* printer;
Choose_language();
Printer = open_printer();
Say_hi(printer);
Close_printer(printer);
}
```

```
void say_hi(FILE* fd){
fprinf(fd, get_message_from_db(1));
}
//the internationalization support code would be written like this:
void choose_language(void){
language = getenv("LANG");
open_language_db(language);
}
void get_message_from_db(int indcx){
rewind(db);
while(read_line(buffer, db) != EOF);
if(indexof(buffer) = = index)
return (char*) valueof(buffer));
return (char* ) NULL;}
//for error message
FILE* open_printer(void){
Static FILE* printer;
If(printer = fopen(PRINT_DEVICE, "w") = = NULL){
fprintf(stderr, get_message_from_db(2));
Exit(1);
Return (FILE*) printer);
}
//database for internationalization file
#language = English
1 hi
2 could not open printer for printout.
#language = Frenchs
        1  Baura
        2  No ...
```

The previous examples for the two internationalization models are just simple pseudo code for printing the hello message and reporting errors. We can see that the Run-Time Model approach is more complicated than the Compile-Time Model approach. This is true for developing a single language product. If we need to release a multi-language product, the run time model is more efficient.

3.2.2 Methods of Handling Resource File

Let us assume that we want to internationalize a product so that one executable file runs worldwide. We then have to properly handle the resource files and the source code to make the executable file interact with the appropriate resource files. Basically, there are three methods we can use to achieve this goal.

3.2.2.1 The resource file is part of the executable file. In this set up, if any change is made to the resource file, we have to recompile the whole project to produce a new executable file. This is not good for localization, but we can put multiple language translations in one resource file. For example, the resource file can contain English, Chinese and Japanese resources. The program can dynamically select different resources at runtime. For small-sized products, this is a good way to make the product work with all different locales.

3.2.2.2 The resource file is separated from executable file and compiled as a separate DLL with all languages in it. We put multiple language translations in one resource file as in method 3.2.2.1. The executable file links with the DLL resource file at run time, and it dynamically selects the appropriate locale resources according to different operating systems. Compared with the first method, the size of the executable file is smaller and changes in the resource file do not require the recompiling of source code.

3.2.2.3 The resource file is separated from executable file and compiled as a separated DLL for each different language. In this set up, each language has its own DLL. We load the specific resource DLL at runtime according to different locale, or install only the locale specific DLLs. Compared with the previous methods, this is the best way to internationalize a product. Nadine Kano [1] recommends this method as a general method for developing global software. In this method, developers have to handle different locales manually, rather than let the executable file itself choose the appropriate resource.

3.2.3 Advantages and disadvantages of the runtime internationalization model

The advantages of this approach are that not only can new languages be added without modification to the source code, but also the amount of disk space required equals the space for a single executable plus the database files, which are typically quite small. The best aspect about this solution is that programmers can be completely freed from the complexity of having to internationalize their code, thus they can focus on the development of the key algorithms for the product. In actuality, the database of international support becomes split into two pieces; the "static cultural data" including notational conventions, collation data, and transliteration data, and the "active program-specific data", which the programmer is responsible for creating and utilizing.

Compared with the compile time internationalization model, the advantages for run time mode are as follows:

- With the addition of localization data, the same executable file may run worldwide.
- Textual elements, such as status messages and the GUI component labels, are not hard-coded in the program. Instead, they are stored outside the source code and

retrieved dynamically.

- Modification for new languages does not require recompilation.
- Culturally-dependent data, such as dates and currencies, appear in formats that conform to the end user's region and language.
- It can be localized quickly.

CHAPTER IV

LOCALIZATION

4.1 The Localization Process

The process of localization or creating localized software usually involves communications among the product team and translators. Figure 4.1 below shows the basic interaction required for international product development.

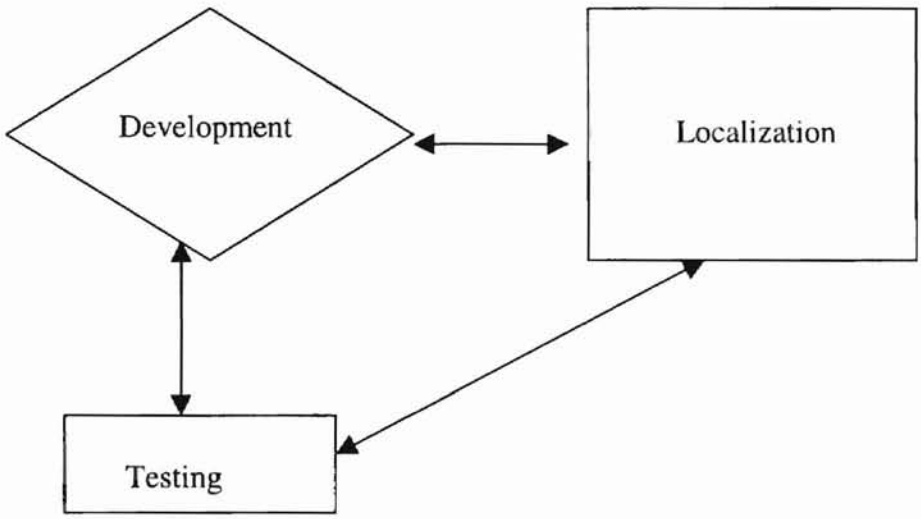

Figure 4.1 the Localization Process

During the core phases, the Development team provides files to the Localization team, which translates text, resizes dialog boxes, and hands files back for compilation, if necessary. The localized executable then goes to the Testing team, which reports functionality problems to Development and reports user interface problems to Localization. All three groups work together to resolve bugs, and the cycle continues.

4.2 Windows Resource Files

The Development team should place every element of the user interface that needs to be localized in one or several Windows resource files; this includes pictures, strings, messages, menus, dialog boxes, and version information. Figure 4.2 shows some resource elements defined by Windows.

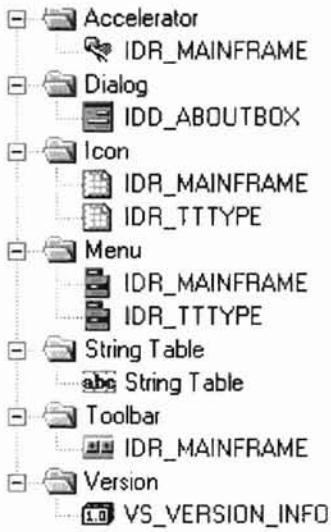

Figure 4.2 Typical Resource Elements

In addition to the elements shown in Figure 4.2, a Windows product may contain other resource elements such as bitmaps, fonts, message table, and user-defined resource files.

4.2.1 Compiling Resource Files

Virtual C++ 6™ is a Win32 SDK (software Development Kit), it is useful for creating and editing resource files. There are three steps in compiling a resource file in a WIN32 SDK environment. First, the RC compiler turns the .RC file into a .RES file; second, the .RES file is converted to an object file; finally, the object file is linked to the program executable. If we have multiple languages that span more than one Windows code page, then we usually need multiple .RC files. For example, we may want to expand our English-language application to include Chinese and Japanese user interface translations.

The standard code page for Japanese is 938 and the standard code page for Chinese is 936. Because the RC compiler can handle only one code page at a time, we need a separate RC file for each of these languages. The translators need to see the localized text as it will appear in the final product, so they will edit the Chinese file on a system that supports Chinese characters, the Japanese file on a system that supports Japanese. We can use the corresponding RC file for building the specific language resource file. To create multiple-language resource files, we may use some localization tool to speed the process and make fewer errors. This kind of tool isolates localizable resources from the rest of the .RC file format and displays only text that needs to be translated. Some localization tools can plug into translation database or allow developers to provide comments for strings so that translators know how to translate them.

4.2.2 Localizing Dialogs

The most time-consuming part of localizing dialog boxes is resizing them, particularly for applications that will ship in numerous languages and that contain a large number of dialog boxes. Developers can minimize the amount of resizing necessary by creating your native-language dialog boxes with as much room to spare as developers feel comfortable leaving. Extend text frames as far as possible to allow text to grow when it is translated. For example, in Figure 4-3 below, the frame surrounding the version text field in the About dialog box extends to the right until it reaches the OK button.

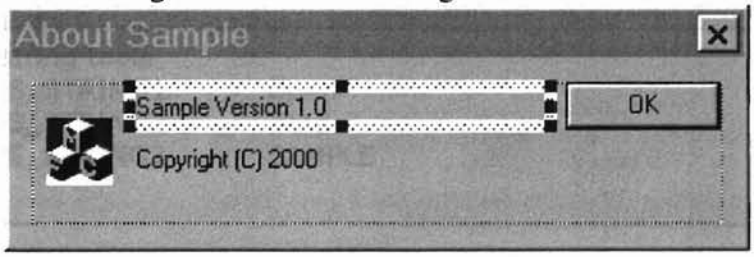

Figure 4.3 The About-Dialog

4.2.3 Storing Strings

For those character strings that will be displayed in the application, we need to avoid putting them in the implementation files, instead, placing them in the string table or message table. String tables are good for short strings and for strings containing only one replacement parameter; Message tables are more convenient for alert and error messages that contain more than one replacement parameter. As message tables support up to 99 parameters.The FormatMessage API function will substitute variables according to each place holder's numeric label and not according to its position in the string. Localization members can freely change a string's word order and FormatMessage will still return correct results. We can use FormatMessage with string tables as well as with message tables, but it is more efficient to use this function with message tables. FormatMessage can retrieve message table strings directly, but it cannot access string tables. To format a string from a string table, we would first have to retrieve it with the LoadString function and then pass it in a buffer to FormatMessage. The following are two examples for each type.

```
//sample.MC

MessageId=1 SymbolicName=IDS_SAMPLE
Language=English
Cannot open file %1
```

```
//string table
IDS_STRING1        Open
IDS_STRING2        Find
IDS_STRING3        SAMPLE
```

CHAPTER V

APPLICATION DESIGN AND IMPLEMENTATION

Many US companies have been very successful in exporting their products to Europe and some Asian regions (notably Japan, Taiwan, and Singapore). Other companies have been trying to improve their businesses in Europe and Asia, and are doing a lot of research in this regard [1][5]. The overseas success of software companies depends very much on their internationalization and localization efforts. For historical as well as economic reasons, most software internationalization and localization efforts so far have been focused on the European countries and Japan [2][5][6]. In particular, very little attention has been given to the characteristics of Chinese software localization. As China experiences rapid economic growth, there is no doubt that demand in software products localized in Chinese will increase dramatically.

In this thesis, the author designs and implements an editor using C++ with MFC. The author also implements a relatively simple editor using Java with JFC Swing. Through designing and implementing the Windows applications, the author will demonstrate how to internationalize software in general and how to localize software applications with C++ and Java in Chinese in particular. We first discuss the major steps in designing and implementing the application using C++. The Java internationalization approach will be discussed separately.

5.1 Planning an International Application

To demonstrate the internationalization and localization ideas, we consider the following major Windows program elements that require localization.

Certain algorithms	Messages
Constants	Prompts
Dialogs	Sounds
Macro Language	Status Bars
Menus	Toolbars

In the mean time, we also need to consider certain locale-dependent features such as

Date format	Time format
Currency format	Measurement units
Calendar formats	Screen font and printer font
Paper Size	Address format

The specifications for this application are presented as follows:

1 The application can be run under a Chinese version windows operating system, it can be also run under an English version of the operating system without recompiling the source code. The application will use localized common dialogs.

2 The application will provide help files both in English and in Chinese.

3 In general, the application uses bitmaps which are free of letters, and avoids punctuation marks that may be changed in the Chinese environment. More specifically, text is not included in bitmaps or icons unless the text does not need to be translated. Be cautious with representations of animals, religious and mythological symbols, national emblems, colors, people (like racial, cultural or

gender), hand gestures, and body language, which might be misinterpreted or might offend users in China. Keep the use of sounds to a minimum, since there are many local accents in China, and some areas may prefer their own accents to Mandarin, the official spoken language for the country.

4 Design menu bars, status bars, toolbars, title bars, and dialog boxes to allow text size to increase. The Chinese characters are larger than Latin characters in the same font type, so the translation is likely to warrant more text space. Some shortcuts will be created using the function keys (F3, F4 etc) or combinations of *Ctrl* and *Alt* with Latin characters. For example, use *ctrl* with character *s* as a shortcut for Save, use *ctrl* with character *a* as a shortcut for Save As.

5 The date and time formats in the application should be automatically selected according to what operating system the application is running on.

6 Different address formats will be used in different versions.

7 There are several font types popular in China, such as standard type, Songti, Heiti and so on. The application should get and set the screen font type and font size. It should allow users to set the different printer font type and size as well.

5.2 Setting up a Development Environment for the Run-time Internationalization Model

Two operating systems are used during the implementation: Windows NT 4.0 (simplified Chinese version) and Windows NT 4.0 (English version). The author will also use Visual C++ 6.0 as the Integrated Development Environment.

The author uses the MFC AppWizard (EXE file) to build the basic application executable file. The author also uses the MFC AppWizard (DLL file) to build the basic

DLL file that is to contain the resource file separated from the MFC executable. The whole application architecture is shown in figure 5.1. A localized executable file consists of compiled source code plus a localized user interface. Customized language code can be contained in a DLL. The basic features which the author can get from VC++ AppWizard (for executable) include the Single Document Interface (SDI) framework, with CEditView as the base class in the document-view architecture, and a help file. For the DLL the author will use MFC extension DLL (use shared DLL); this option is chosen just in case some MFC built-in classes are needed as base classes to build the DLL application.

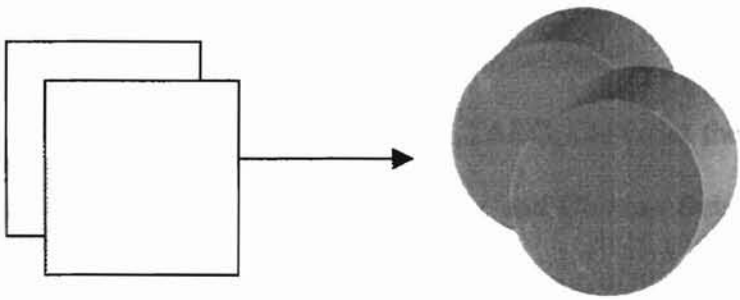

Figure 5.1 Various Components of an application

5.2.1 Handling the Resource File

Three ways to handle resource files were discussed in Chapter 4. We are going to use methods 1 and 3 respectively. First, the resource file will become part of the executable file.

For this method, the following steps are taken [10]:

- Create a default MFC AppWizard application (called "Project" in this thesis), and specify "U.S English" for the resource language, Using MFC library as a shared DLL.
- Insert into Project a default MFC AppWizard application as an independent project (called ResourceDll in the thesis), and select MFC extension DLL for the type of DLL.

- Remove and delete the RC file, the Resource.h file, ResourceDll.rc2 file and res directory from the ResourceDll project.
- Add CPROJECT.RC file to the ResourceDll project.
- For each additional language, Project has localized resources. Create both the release and debug configuration, copy the settings from Win32 debug and Win32 release (in the thesis, Chinese Debug and Chinese Release configurations are built). For both Chinese Release and Debug configuration, we need to set the preprocessor definition as AFX_RESOURCE_DLL and AFX_TARG_CHS. Set the output name for Win32 Debug, Win32 Release, Chinese Release and Chinese Debug to be ResourceDll.dll, ResourceDlld.dll, ResourceDllchs.dll and ResourceDllchsd.dll respectively.
- For all the configurations of the ResourceDll application, we need to add the path for all the Project application in the "additional include directory field".
- For all the configurations of the Project application, we also need to set the preprocessor definitions field, add AFX_RESOURCE_DLL; this setting will remove all of the resources from Project.exe
- In the ResourceDll workspace in the Visual C++ IDE, we need to copy all the resources including menus, toolbars, dialogs, string tables into another language version, and set the language type for another version to CHINESE (PRC).
- Work on the English resources under English Windows NT 4.0, work on the Chinese resources under Chinese Windows NT 4.0. After finishing all the work, compile and build the application in either operating system; the Project.exe file can be run under different operating systems and it chooses the appropriate language version resource

file dynamically. The result is that the executable file contains two resource file for two different locales. It will choose appropriate resource under different operating systems.

Using MS Visual C++ 6.0 IDE to open the executable file as resource, we can see what is contained in the EXE; see Figure 5.2

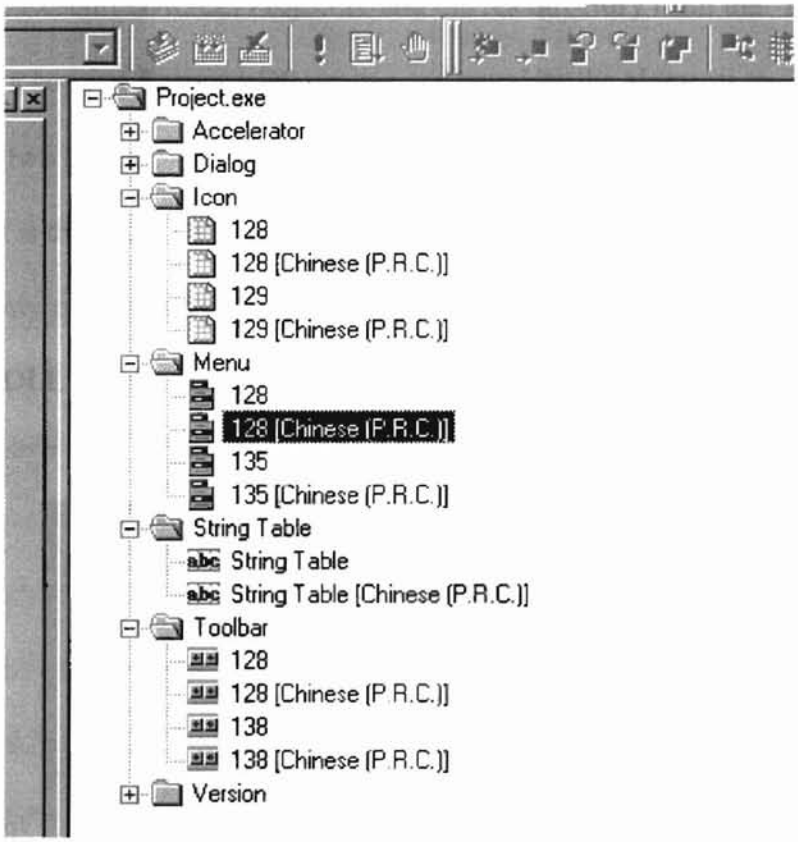

Figure 5.2 An EXE with Two Resource Files.

Next, the resource file is separated from executable and compiled as a separated DLL for each language version. This is the Method 3 mentioned in Chapter 3; the following steps need to be taken to build separated DLLs:

- Create the new project (called Project in this thesis) using MFC AppWizard (exe).

- Create another new project (called ResourceDll in the thesis) using MFC AppWizard (dll), the DLL project is not in the same workspace as that in the method 1.
- Delete Resource.h, ResourceDll.rc and the res directory from the ResourceDll project.
- Copy Project.rc (change the name to ResourceDll.rc) into ResourceDll project, and copy Resource.h and res directory into ResourceDll project as well.
- Delete ResourceDll.rc2, Project.rc and the res directory from the "Project" project.
- Make a copy of ResourceDll project and translate the elements to Chinese.
- Build two DLLs.

The result is that each DLL consists of one resource file for each language, the executable file can only run properly with the correct DLL. From Figure 5.3, Figure 5.4, we can see that each DLL contains only one language version resource.

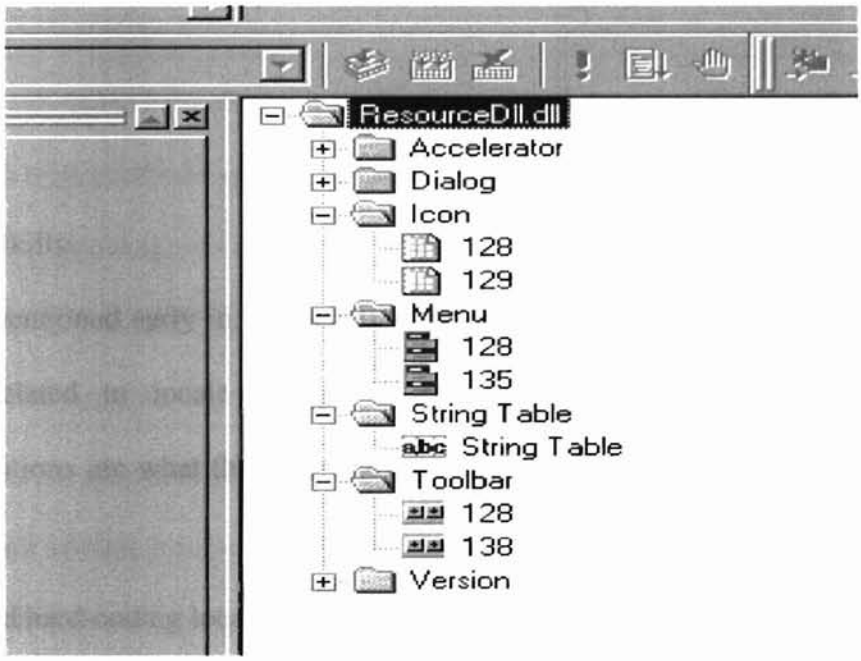

Figure 5.3 An English DLL

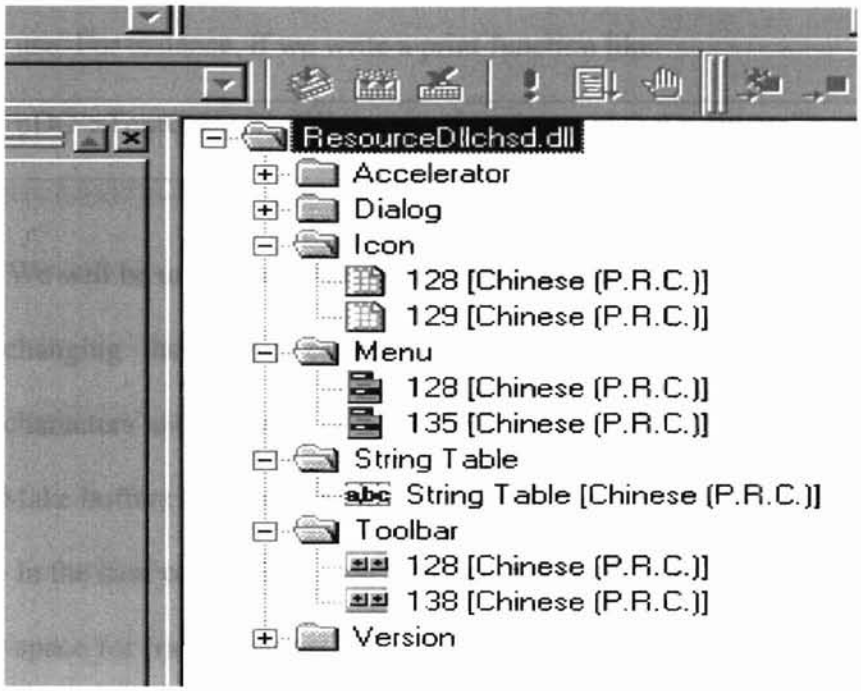

Figure 5.4 An Chinese DLL

5.3 Coding Skills

As mentioned early in Chapter II, developers have to deal with a lot of intricate issues related to locale change in real world programming. The following considerations are what the author always keeps in mind when coding international software.

1 Avoid hard-coding localizable elements.

Hard-coded strings, characters, constants, screen positions, filenames, and file paths are difficult to track and localize. From the beginning of designing a global

application, always remember that all the elements should be suitable for global use. For instance, if we write a print function like:

```
pDC->TextOut(1, 2, "We are implementing global software");
```

We will be unable to display it in Chinese under Chinese operating system without changing the source code, since the program cannot select the appropriate characters to display.

2. Make buffers large enough to hold translated text.

In the case of characters we always use the largest buffer size to avoid insufficient space for translated text. In the English version of the product, we normally use the following code:

```
char szOk[3];
GetCHSName(szOk);
```

This will cause a problem when OK is translated into two Chinese characters, which requires four bytes. A better way for global software writing is to use code like the following:

```
char szOk[MAXSIZE]; //MAXSIZE is 4096 in win 32 programming
GetCHSName(szOk);
```

3. Do not limit character parsing to Latin Script.

It is not advisable to parse the character string, and assume the characters are Latin script; we can use the Win32 function IsCharAlpha( ) to find out.

4. Do not assume that characters are always 8-bit.

If we write code that is based on 8-bit characters, the code certainly will not work with

double byte character sets or Unicode. For instance, the following code will not work in a generic code base.

```
int len = strlen(szString);
pBuffer = (char*)malloc(len);
```

If we want to make it work with different character sets, we might use code like:

```
int len = strlen(szString);
pBuffer = (TCHAR* )malloc(len*sizeof(TCHAR));
```

Keeping these items in mind will make the localization process a little easier. Besides, the possible mistakes mentioned above are easy to avoid as long as one is willing to pay some attention to internationalization issues while coding.

5.4 Testing

Testing is a very important step in the cycle of developing a software product; this is especially so for products to be released to different locales. Not only the functionality of the application should be tested carefully, the issues related to localization (such as the user interface) are also very important. From Appendix A we see how testing a global software product is different from testing a normal English-version product. Appendix A is the checklist for internationalization suggested by Nadine Kano [1].

5.5 Application Implementation

Planning ahead for internationalization always saves time for localization later on. Again, we treat the two models for internationalization separately.

I. Compiler-time Internationalization

A Chinese version application is implemented under the Simplified Chinese Windows NT 4.0 operating system; an English version application is implemented under the English version Windows NT 4.0 operating system. One application may not be able to run correctly under another operating system. The English language version application runs well under both operating system, but of course it does not display Chinese characters in the Chinese WinNT4.0, see Figure 5.5. The Chinese language version application runs well under the Chinese WinNT4.0, but it does not display English characters on the English WinNT4.0. Figure 5.6 shows the correct Chinese version application running under the Chinese WinNT 4.0 and Figure 5.7 shows it running under the English WinNT4.0.

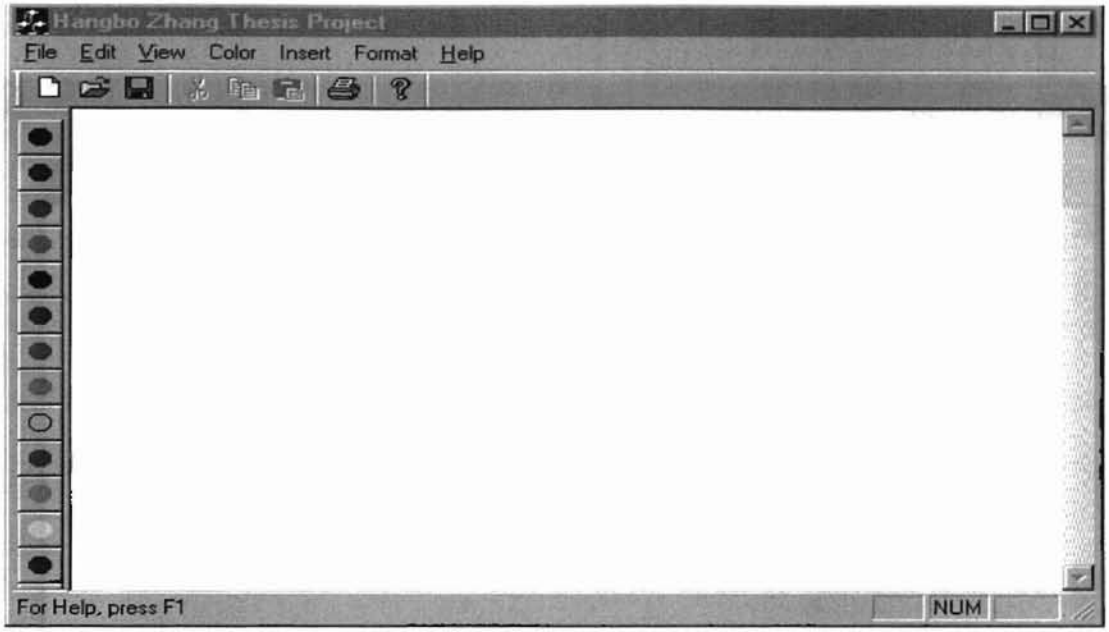

Figure 5.5 An English Application

Figure 5.6 An Chinese Application

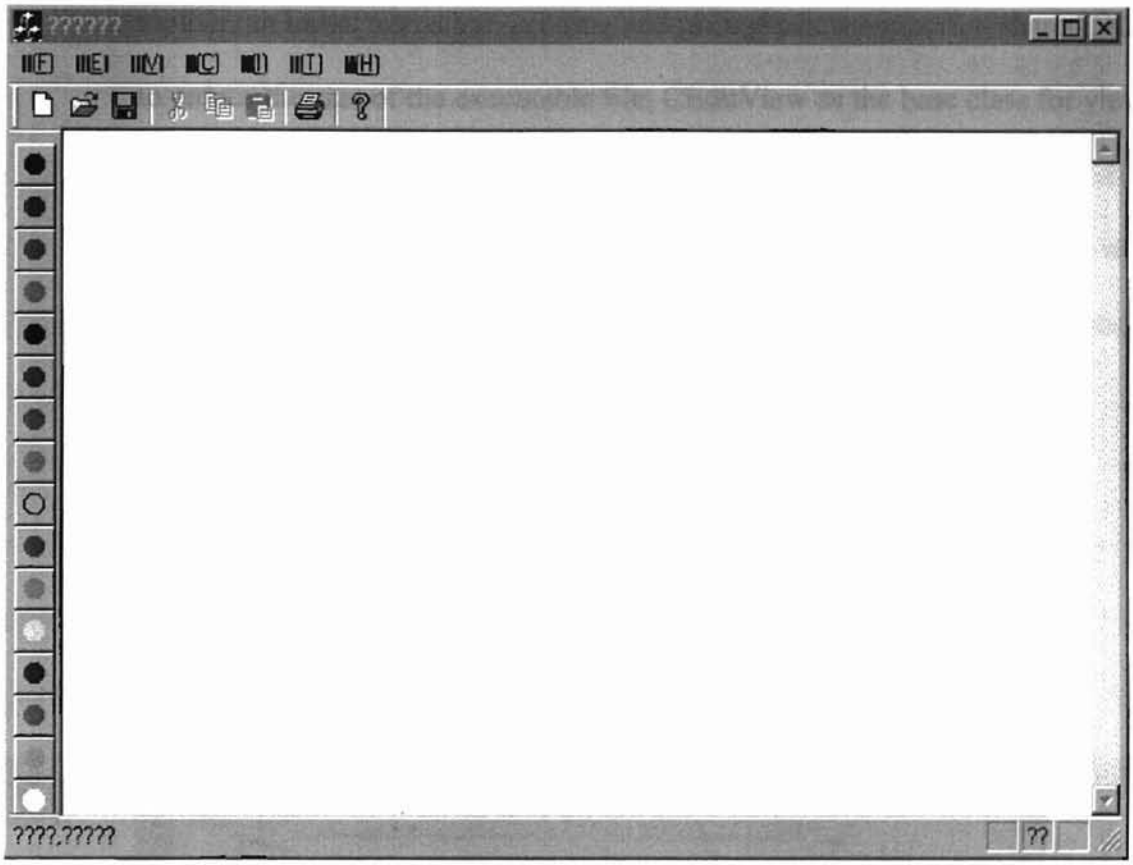

Figure 5.7 A Chinese Application on English WinNT Operating System

II. Run-time Internationalization

Using this model, the author implements an application which can be run under different operating systems. The application displays English characters under English WinNT as shown in Figure 5.5, and it displays Chinese characters under the Chinese WinNT as shown in Figure 5.6. The author also discusses some program details in the following sections.

The DBCS version implementation involves the following issues:

- Setting the basic environment: A single document application, a docking toolbar, an initial status bar, printing and print preview support, a shared DLL to reduce the size of the executable file, CEditView as the base class for view. Figure 5.8 summarizes the setup information.

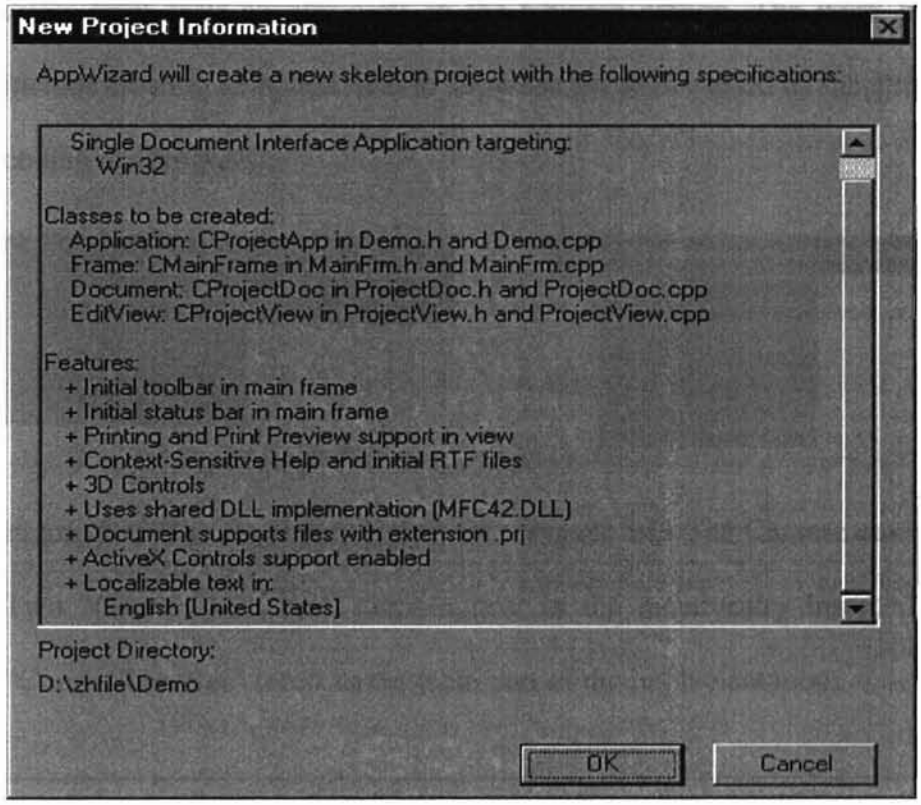

Figure 5.8 Basic Settings

- IME support. IME is the acronym for Input Method Editor. Different languages are supported by different IMEs, and there are IME's that are multi-language. In this implementation, the author uses the IME that comes with the Simplified Chinese version of Windows NT4.0. Figure 5.9 shows what the IME looks like.

Figure 5.9 Chinese Input Method Editor (IME)

- Working on Menus and toolbars. Menus and toolbars in the English edition differ from their counter-parts in the Chinese edition. The work here will include creating an option item to show the problem caused by inappropriately coding the program.

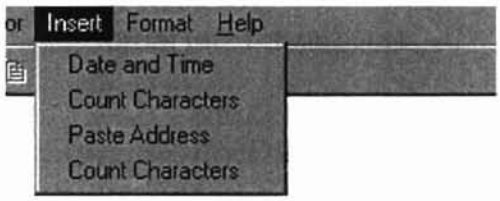

Figure 5.10 The English environment Figure 5.11 The Chinese environment

Two "Count Characters" items appear in the menu entry Insert. The first "Count Character" refers to the main part of the implementation:

```
CString str;
str.LoadString(IDS_COUNT_CHAR);
int nlen = GetBufferLength();
TCHAR buf[64];
wsprintf(buf, str, nlen);
AfxMessageBox((LPCTSTR)(buf));
```

The problem in this implementation is that it does not work with double byte character sets since it assumes the character string is Latin. Thus we implemented a second "count Character" menu item:

```
void CProjectView::OnInsertCharcount()
{
	CString str;
	str.LoadString(IDS_COUNT_CHAR);
	CEdit &edit = GetEditCtrl();
	int lines = edit.GetLineCount();
	TCHAR *charcnt;
	charcnt = (TCHAR*) malloc(100*sizeof(TCHAR));
	int count = 0;
	for(int i=0; i<lines; i++)
	{
		edit.GetLine(i, charcnt, 100);
		if(vfDBCS) //get this from constructor
		{
			for(count; *charcnt; charcnt = MyCharNext(charcnt) )
			{
				if(IsDBCSLeadByte(*charcnt))
					charcnt++;
				++count;
			}
		}
		else
		{
		for(count; (*charcnt); charcnt++)
				++count;
		}
	}
	TCHAR buf[64];
	wsprintf(buf, str, count);
	AfxMessageBox((LPCTSTR)(buf));
}
```

In this implementation the author considers two situations. One important thing we need to pay attention to is the function MyCharNext(char* pszStr). The implementation details are shown below:

```
char* CProjectView::MyCharNext(char* pszStr)
{
        BYTE bRange = 0;
        while((bRange < 12 ) && (vbLBRange[bRange] != NULL) )
        {
                if((*pszStr >= vbLBRange[bRange]) &&
                        (*pszStr <= vbLBRange[bRange + 1]) )
                        return (pszStr + 2); //skip two bytes
                bRange += 2;
        }
        return (pszStr + 1);
}
```

In this implementation, the author checks to see whether *pszStr is a lead byte, the constant 12 allows up to 6 pairs of lead-byte range values. The second "Count Character" sub menu item works well for both MBCS and SBCS.

- Working on bitmaps and icons: Use the resource editor to create different bitmaps and icons. Most of the icons created by the AppWizzard are good enough for our application.

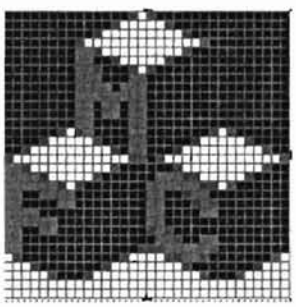

Figure 5.10 An Application icon

Figure 5.11 A Document icon

Some icons, as shown Figure 5.10, Figure 5.11 and Figure 5.12 should not be used in the implementation of an international software product, since the text on the icon is hard to

product, since the text on the icon is hard to translate during localization.

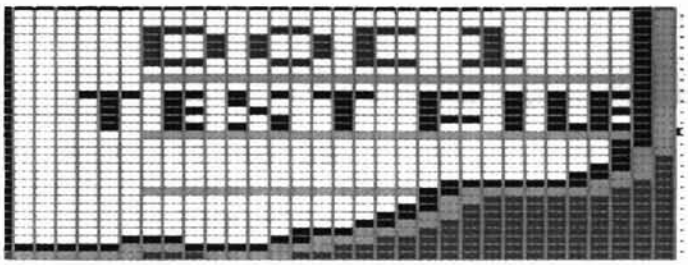

Figure 5.12 An Icon containing text

- Creating List Boxes for Date and Time String. Two list boxes will be created to demonstrate different formats in date and time.

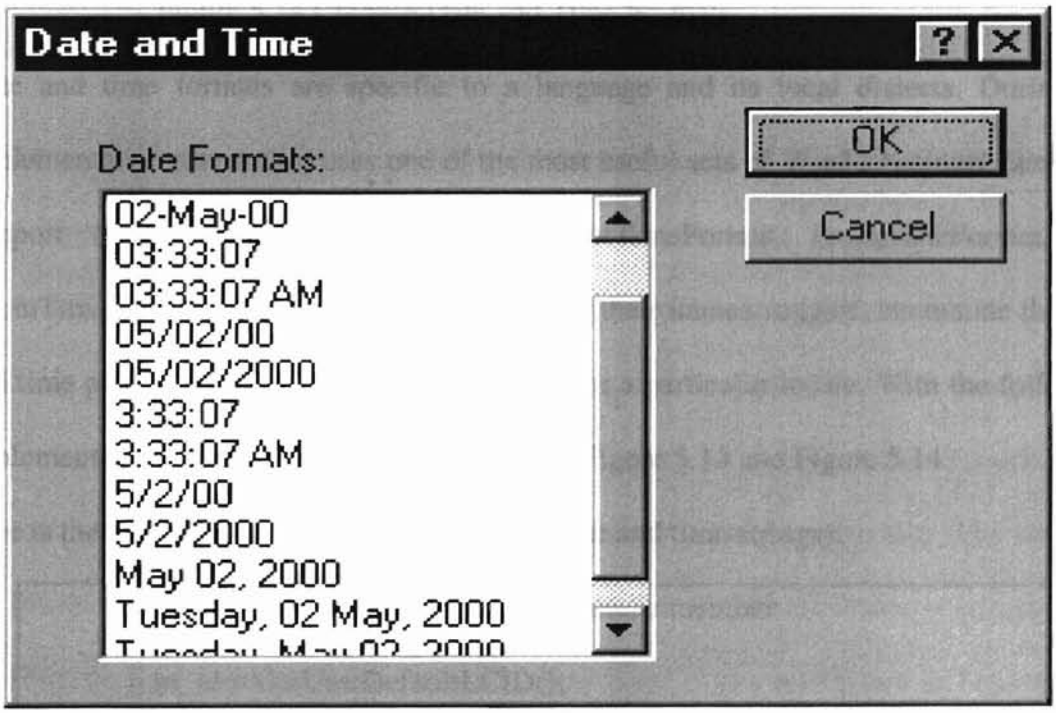

Figure 5.13 American Date and Time Formats

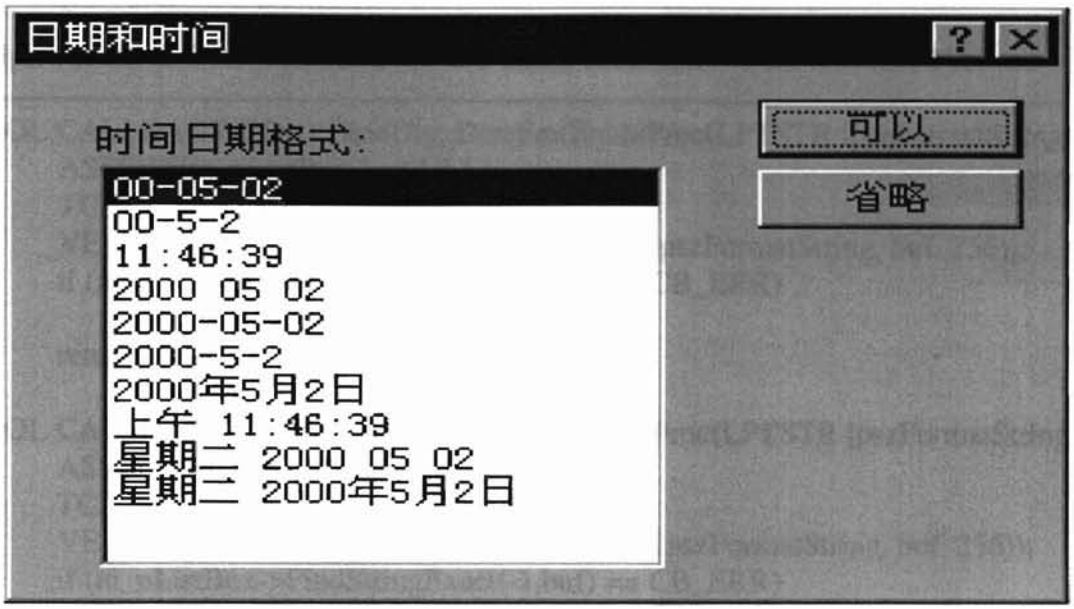

Figure 5.14 Chinese Date and Time Formats

Date and time formats are specific to a language and its local dialects. During the implementation, the author uses one of the most useful sets of Win32 National Language Support (NLS) API – GetDateFormat, GetTimeFormat, EnumDateFormat, and EnumTimeFormats. The latter two functions, as their names suggest, enumerate the date and time picture string that the system carries for a particular locale. With the following implementation we get the two list boxes in the Figure 5.13 and Figure 5.14.

Here is the core part for adding list elements (date and time strings):

```
m_pListBox = &m_listbox; // set static member
GetLocalTime(&m_time);
m_id = GetUserDefaultLCID();

EnumDateFormats(DateFmtEnumProc,m_id, DATE_SHORTDATE);
EnumDateFormats(DateFmtEnumProc, m_id, DATE_LONGDATE);
EnumTimeFormats(TimeFmtEnumProc, m_id, 0);

m_pListBox = NULL;
m_listbox.SetCurSel  (0);
```

Helper functions DateFmtEnumProc and TimeFmtEnumProc are also listed below:

```
BOOL CALLBACK CDateTimeDlg::DateFmtEnumProc(LPTSTR lpszFormatString)
{       ASSERT(m_pListBox != NULL);
        TCHAR buf[256];
        VERIFY(GetDateFormat(m_id, 0, &m_time, lpszFormatString, buf, 256));
        if (m_pListBox->FindStringExact(-1,buf) == CB_ERR)
                m_pListBox->AddString(buf);
        return TRUE; }

BOOL CALLBACK CDateTimeDlg::TimeFmtEnumProc(LPTSTR lpszFormatString)
{       ASSERT(m_pListBox != NULL);
        TCHAR buf[256];
        VERIFY(GetTimeFormat(m_id, 0, &m_time, lpszFormatString, buf, 256));
        if (m_pListBox->FindStringExact(-1,buf) == CB_ERR)
                m_pListBox->AddString(buf);
        return TRUE;}
```

- Message Boxes and Message Files: Message boxes are very useful to give users information about the current status of using the application. The author created some message boxes, and all the messages strings and the their translation will be put in a DLL and be loaded dynamically in run time. An example of such a string is "the character number is n", where n is the actual number. We need to store the string "the character number is %d" in the STRING Table of the resource file. The string is loaded during run time. The corresponding translated Chinese string is stored in the Chinese resource file. Some examples of message boxes are shown in Figures 5.15 5.16, 5.17 and 5.18.

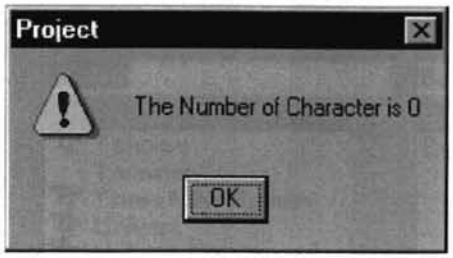

Figure 5.15 English Message Box 1

Figure 5.16 Chinese Message Box 1

Figure 5.17 English Message Box 2

Figure 5.18 Chinese Message Box 2

- Working on screen fonts and set printer fonts. For different locales the operating system supports different fonts for screen display and printer, because these fonts are locale dependent. Using the screen-fonts dialog and printer-fonts dialog, we can take advantage of some Windows built-in controls which have been localized well, see figure 5.19 and Figure 5.20. After we select a font, the editor will use the appropriate font to display characters.

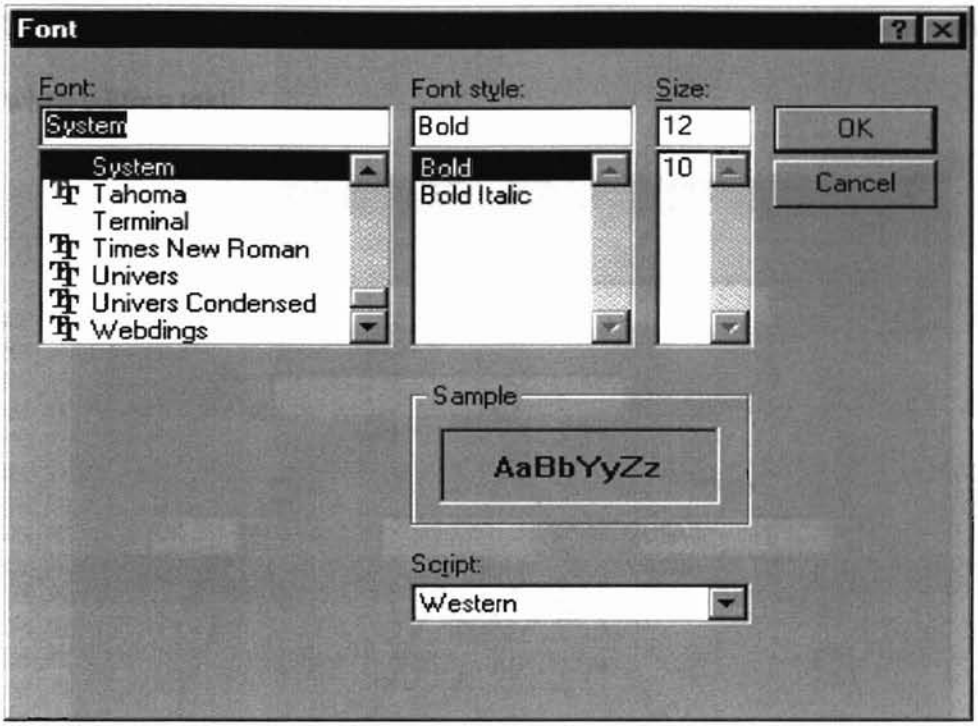

Figure 5.19 A Font Dialog for Printer and Screen Display

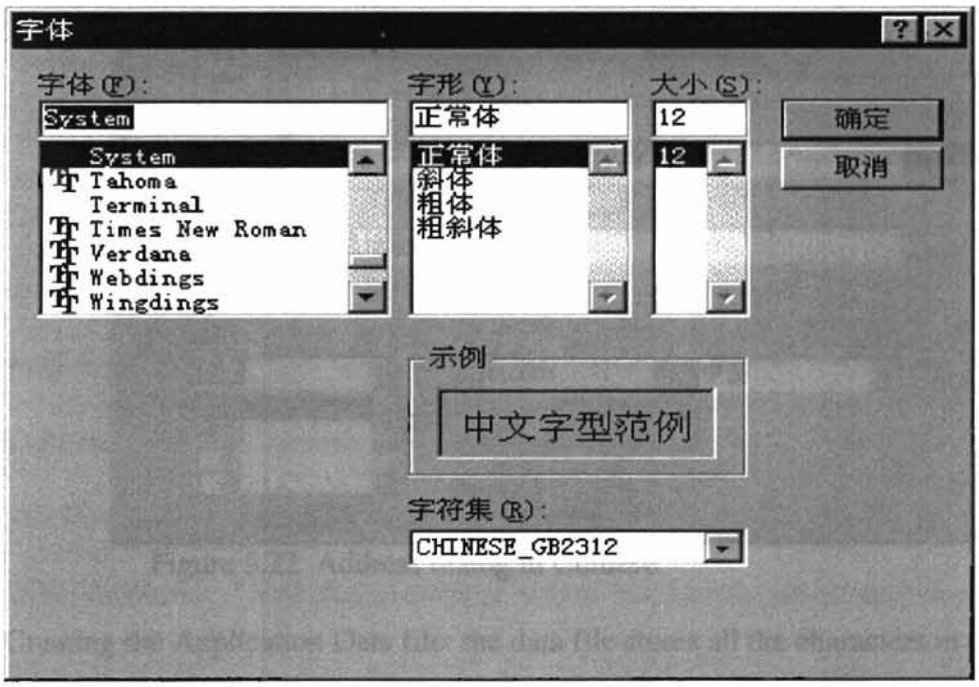

Figure 5.20 A Font Dialog for Printer and Screen Display in Chinese

- Set address dialog to get address information and display appropriate address format when editing text.

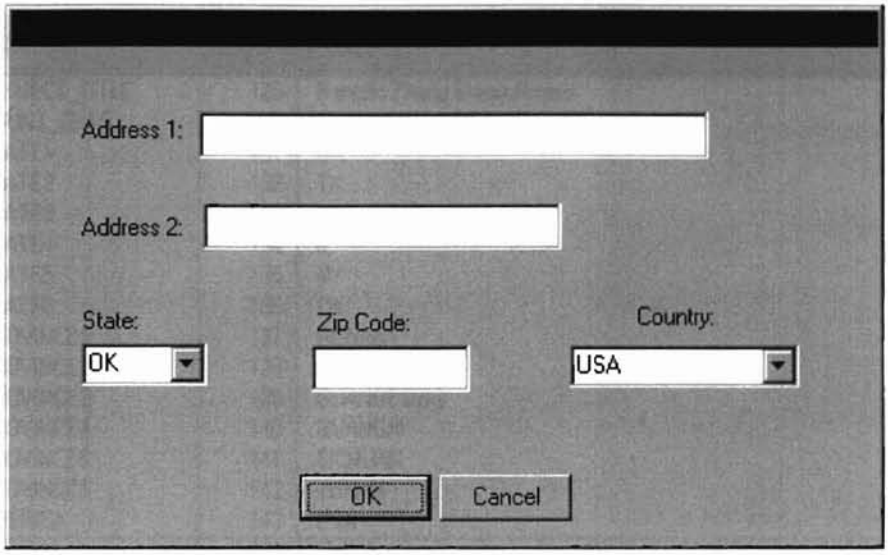

Figure 5.21 An Address Dialog in English

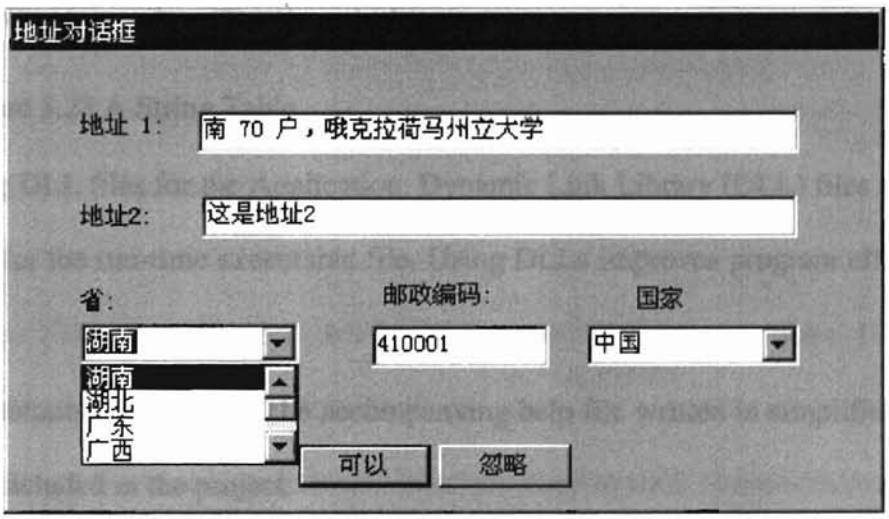

Figure 5.22 Address dialog in Chinese

- Creating the Application Data file: the data file stores all the characters in the editor.

- Construct all the strings in the string table of the program as shown in Figure 5.23.

ID	Value	Caption
IDR_MAINFRAME	128	Project\n\nProjec\nProjec Files (*.prj)\n.prj\nProject.Document\nProjec Document
IDS_PROJECT_TITLE	129	Hangbo Zhang Thesis Project
IDS_COUNT_CHAR	130	The Number of Character is %d
IDS_STATE1	131	OK
IDS_STATE2	132	TX
IDS_STATE3	133	KS
IDS_STATE4	134	IL
IDS_STATE5	135	NY
IDS_STATE6	136	CA
IDS_PROVINCE1	137	HUNAN
IDS_PROVINCE2	138	HUBEI
IDS_PROVINCE3	139	GUANGDONG
IDS_PROVINCE4	140	GUANGXI
IDS_PROVINCE5	141	SICHUAN
IDS_PROVINCE6	142	YUNAN
IDS_AMERICA	143	USA
IDS_CHINA	144	CHINA
IDS_JAPAN	145	JAPAN
IDS_KOREA	146	KOREA
IDS_CANADA	147	CANADA
IDS_NO_ADDRESS	148	No address has been entered!
IDS_INVALID_LINE	149	Line number is invalid
IDS_BIG_LINENUM	150	Line number %d is out of range

Figure 5.23 A String Table

- Creating DLL files for the Application. Dynamic Link Library (DLL) files need to be created for the run-time executable file. Using DLLs improves program efficiency as well.
- Documentation in Chinese: The accompanying help file written in simplified Chinese will be included in the project.
- Other Features: Add some features which makes the editor easy to use. In the implementation we have included the following Dialog classes: CGoLineDlg, CsetTabStops, CmySplashWnd. The last dialog as shown in Figure 5.24 is used to show the project information at the beginning of the project. This window also needs

to be localized.

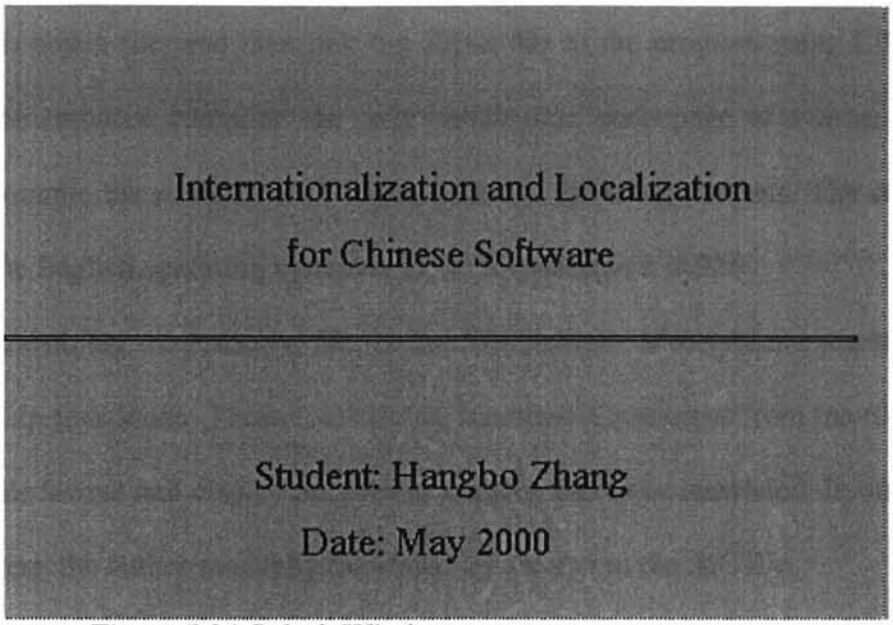

Figure 5.24 Splash Window

- Testing the Application: We follow general testing rules [7] and the check list in Appendix A.

5.6 Localization of Application

For compile time internationalization model, we do not have to deal with localization, since the localization is finished while the internationalization is done. For runtime internationalization, no matter what methods we have used to handle resource file, we do not have to touch the source code at all. What we need to do is to localize the Windows features contained in the resource file. As the author planned ahead before writing a single line of code, the source code for the application has been internationalized. An internationalized application is very easy to localize. The following procedures have been taken to localize the program.

1 Compiling the resource file. We use the build operation in Visual C++ 6.0 to compile the .RC file into a .res file, then use CVTRES to convert the .res file into an object file, and then link the object file to the program using LINK32. Since the resource compiler can only handle one code page at a time, we have to compile the resource files under different operating systems. The code page for the English operating system is 1252; for Chinese it is 936.

2 Localizing the resource file. In the localization industry there are many tools to help translation. Those tools isolate localizable resources from the rest of the .RC file format and display only the text that needs to be translated. In our application here, the author manually translated all the text in the .RC file.

3 Localizing Dialogs. The most time-consuming part of localizing dialog boxes is resizing them. For one or two locales, as is the situation here, this is not difficult. If a product will be released to a lot of locales, and if the application uses a large number of dialog boxes, resizing these dialogs could be a very tedious task. To ease the pain here, developers usually follow the rule that the English version text fields in a dialog box should employ as much room as they feel comfortable leaving. Figure 5.21 and figure 5.22 show the difference between the sizes of the same text field in Chinese and English.

4 Check the string table. We need to check the string translation. We also need to check to see if the messages have been correctly formatted.

5 We pack the executable file and all the DLLs together. This executable can then be run under different operating systems.

5.7 Java Internationalization and Localization

The basic Java internationalization issues are very much similar to the ones in C++. But Java has its special way of handling these issues. Listed below are some differences in internationalization issues between C++ and the current version of Java, Java 2.

1 Java supports the Unicode encoding technique, which makes life for international programmers much easier, since all the characters are represented using two bytes. When Single Byte Character Sets or Multiple Byte Character Sets are used, the Java Virtual Machine will handle the conversion between Unicode and SBCS (or MBCS).

2 A Java application also uses something like a resource file, which includes user interface features and locale specific elements. Visual C++ can save these elements visually, but Java saves all these features in a plain text file called the properties file.

3 With the pure OOP characteristics, Java also has two special classes to handle the properties file. These classes are called the PropertiesResourceBundle and the ListResourceBundle. Both classes are inherited from the base abstract class ResourceBundle. In C++ there is no class of this type to handle the resources.

4 Clearly there are other differences created by programming details related to the two different programming languages.

4.7 Java Application Design and Implementation

Our Java application in the thesis is a simple text editor. Basically, it will show us how to

internationalize Java applications. The following are the implementation details.

1 The base class for the Jthes application class is Jpanel. Basic Windows elements such as JTextComponet, HashTable, JMenuBar, Toolbar, Jcomponent (for status) are set in this class.

```
class Jthes extends JPanel {

   private static ResourceBundle resources;
        private JTextComponent editor;
   private Hashtable commands;
   private Hashtable menuItems;
   private JMenuBar menubar;
   private JToolBar toolbar;
   private JComponent status;
        // .....
```

2 Create the English properties file which can be used for any language as shown in Figure 5.25. A properties file stores information about the characteristics of a program or environment. A properties file is in plain-text format. For this application, locale-specific data must be tailored according to the conventions of the end user's language and region.

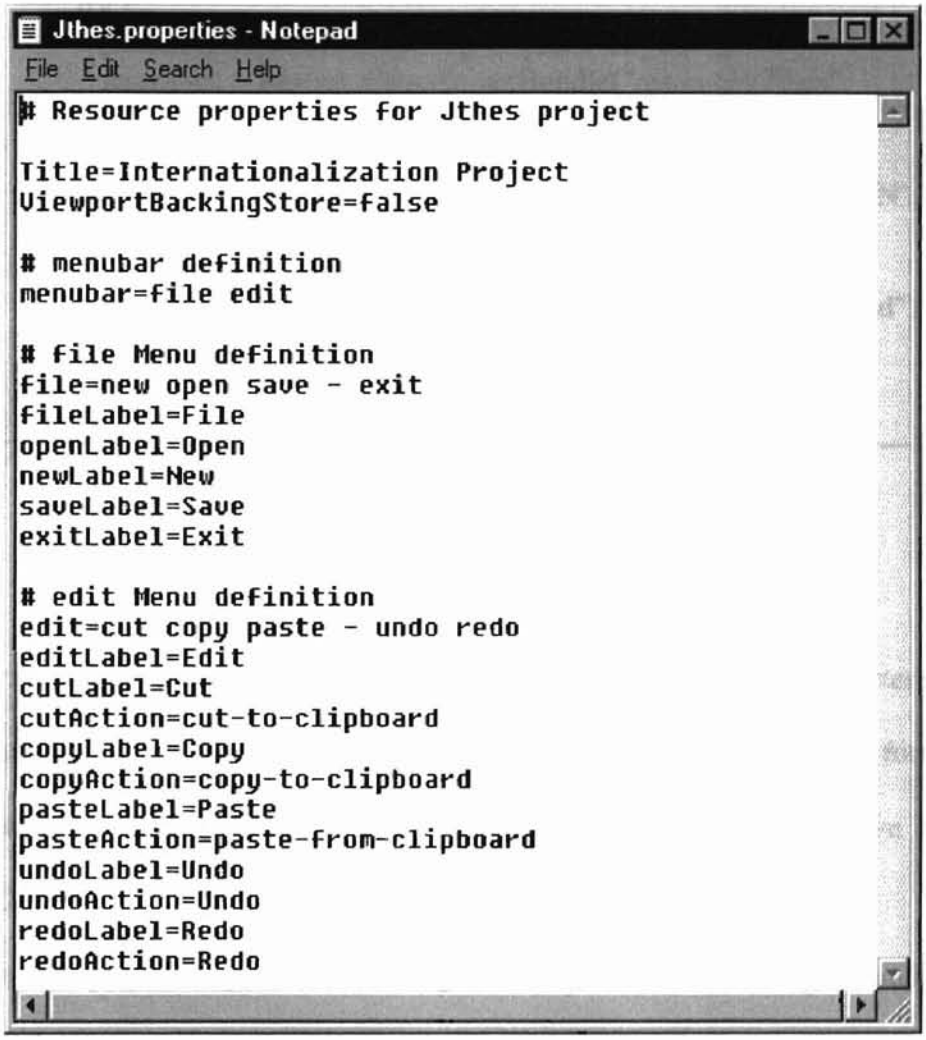

Figure 5.25 A Properties file for an English Project

3 Load the resource file into the program according to locale, and implement the event handler for all the menu items.

```
static {
        try {
        if(Locale.getDefault().toString().equals("en_US"))
          resources = ResourceBundle.getBundle("resources.Jthes",
                                        Locale.getDefault());
        else if(Locale.getDefault().toString().equals("zh_CN"))
        resources = ResourceBundle.getBundle("resource.Jthes_zh_CN",
                                        Locale.getDefault());
             } catch (MissingResourceException mre) {
               System.err.println("resources/Jthes.properties not found");
               System.exit(1);
            }
         }
```

4 Translate all localizable text into Chinese. Use the Java text converter function, native2ascii, to convert the Chinese properties file into Unicode format. The resulting properties file is named Jthes_zh_CN.properties, as shown in Figure 5.26.

5 Test the project. Figure 5.27 and Figure 5.28 present CUI in English and Chinese, respectively.

Figure 5.26 A Localized Properties File

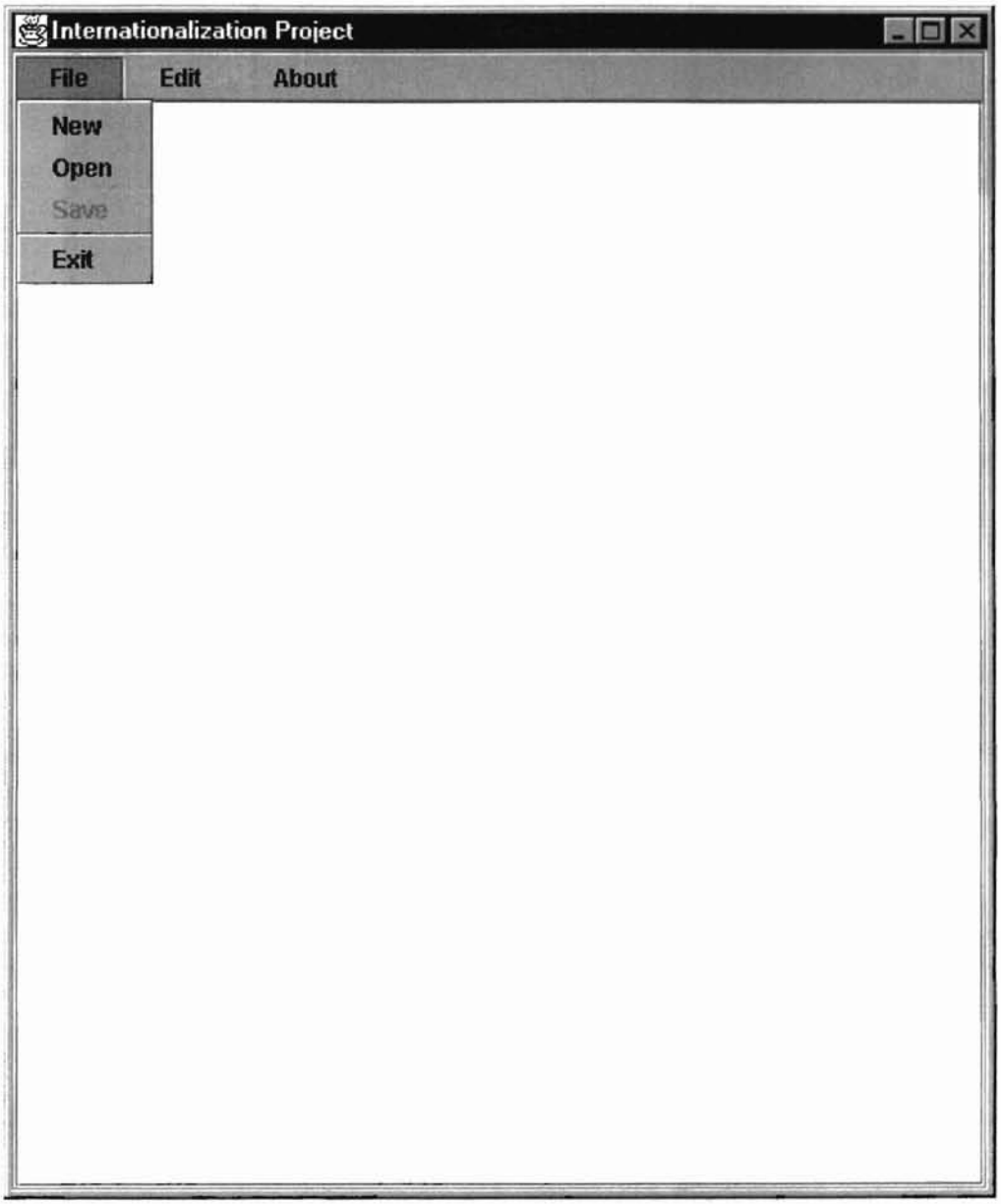

Figure 5.27 Java Application GUI in English

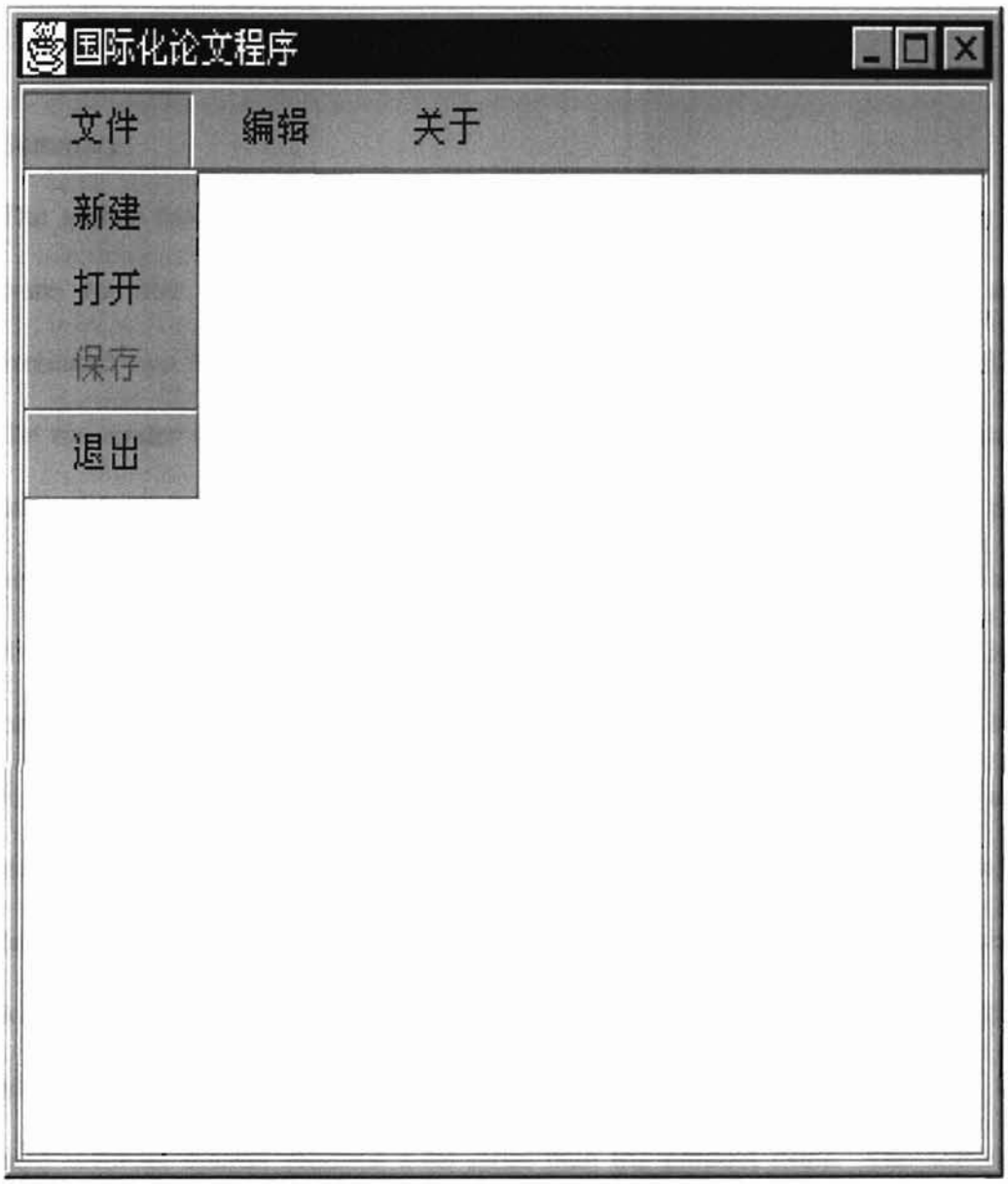

Figure 5.28 Java Application GUI in Chinese

CHAPTER VI

SUMMARY AND FUTURE WORK

6.1 Summary

The author has discussed the issues of internationalization and localization of software for the Chinese market. The study has successfully designed and implemented two Windows applications using C++ and Java. The executable files can be run under different operating systems dynamically loading the appropriate Dynamical Link Library (DLL). The applications have been adapted to English and Chinese without changes to the source code. Users can get consistent appearance and functionality in both versions of the applications. The study has also showed why it is important to handle the Windows resources and locale related features correctly and appropriately. The study used a generic approach to software internationalization and adapted the application to the Simplified Chinese locale. In general, software internationalization and localization is very complicated. To successfully market a software product outside the US, a well functioning feature design along with a friendly user interface is very essential. Furthermore, to succeed in entering the Chinese software market requires a lot more than the product itself. The software itself must be very sensitive to the Chinese cultural tradition and political system, both quite different from the western society.

6.2 Future Work

While working on the thesis, the author noticed some interesting subjects related to internationalization and localization. The author would like to share these with people who might be interested in this area.

1 Conversion tools between ASCII, DBCS and Unicode. Since Unicode is not fully supported on Windows 95/98, these tools should prove useful in internationalization/localization projects.

2 Web internationalization and localization. With the rapid development of the Internet, the need to localize web pages for all international audience has been increasing on a daily basis.

3 Internationalization and localization for various potential software markets. The Arabic world has very special culture. Some Southeast-Asian countries such as Thailand and Vietnam also have different cultures. To study the internationalization and localization issues for these potential software markets should be very interesting.

4 Localization tools. Localization tools are widely used to speed the localization process. There are a few commercial tools available in today's market.

REFERENCE

1 Dave Taylor, Global Software, Developing Applications for the International Market, Springer Verlag, 1992

2 Nadine Kano, Developing International Software for Windows 95 and Windows NT, Microsoft Press, 1995

3 David J. Kruglinski, Inside Visual C++ 4.0, Microsoft Press, 1995

4 Mike Blaszczak, Professional MFC with Visual C++ 5.0, Wrox Press, 1997

5 Tony Fernandes, Global Interface Design – A Guide to Design International User Interfaces, AP Professional, 1995

6 Microsoft, Programmer's Guide Documentation,
URL: http://www.microsoft.com/globaldev/gbl-gen/

7 Kory Srock, Planning for and Testing Global Software,
URL: http://www.microsoft.com/globaldev/gbl-gen/

8 Microsoft, Users' Guide for Visual C++
URL: http://www.msdn.com/dgdeveloping_international_application.html

9 Dale Rogerson, Inside COM, Microsoft Press, 1997

10 Issac Varon, Creating Localized Resource DLLs for MFC Applications, Microsoft Visual Studio Library Documentation, Article ID: Q198846

11 Microsoft, The Windows Interface: An Application Design Guide, Microsoft Press, 1991

APPENDIX A Check List for Internationalization

- Program specifications account for international considerations from the outset
- Features important to international markets are included
- Icons and bitmaps are generic, are culturally acceptable, and do not contain text
- Shortcut-key combinations are accessible on international keyboards
- Consistent English user interface terminology is used in strings
- Strings are documented using comments to provide context for translators
- Strings or characters that should not be localized are marked
- International laws affecting feature designs are considered
- Third-party agreements support international design issues
- Menu and dialog-box designs leave room for text expansion
- Text and messages are devoid of slang and specific cultural references
- Code is generic enough to work for several languages
- Code doesn't concatenate strings to form sentences
- Code doesn't use a given string variable in more than one context
- Code doesn't contain hard-coded character constants, numeric constants, screen positions, filenames, or pathnames that presume a particular language
- Buffers are large enough to handle translated words and phrases

- Program allows input of international data
- All language editions can read one another's documents
- Code contains support for locale-specific hardware, if necessary
- Features that don't apply to international markets can be removed easily

APPENDIX B Code for C++/MFC Application

```
// DateTimeDlg.cpp : implementation file
//

#include "stdafx.h"
#include "Project.h"
#include "MyDialog.h"
#include "DateTimeDlg.h"
#include "hlpids.h"
#include <winnls.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

SYSTEMTIME CDateTimeDlg::m_time;
LCID CDateTimeDlg::m_id;
CListBox* CDateTimeDlg::m_pListBox = NULL;

const DWORD CDateTimeDlg::m_nHelpIDs[] =
{
        IDC_DATETIME_LIST, IDH_PROJ_DATETIME,
        IDC_DATETIME_FORMAT, IDH_PROJ_DATETIME,
        IDOK, IDH_PROJ_DATETIME,
        IDCANCEL, IDH_PROJ_DATETIME,
        0 , 0
};

CDateTimeDlg::CDateTimeDlg(CWnd* pParent /*=NULL*/)
        : CMyDialog(CDateTimeDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CDateTimeDlg)
        m_selection = _T("");
        //}}AFX_DATA_INIT
}

void CDateTimeDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CDateTimeDlg)
        DDX_Control(pDX, IDC_DATETIME_LIST, m_listbox);
        DDX_LBString(pDX, IDC_DATETIME_LIST, m_selection);
        //}}AFX_DATA_MAP
```

```
}

BEGIN_MESSAGE_MAP(CDateTimeDlg, CMyDialog)
	//{{AFX_MSG_MAP(CDateTimeDlg)
ON_LBN_DBLCLK(IDC_DATETIME_LIST, OnDblclkDatetimeList)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CALLBACK CDateTimeDlg::DateFmtEnumProc(LPTSTR lpszFormatString)
{
	ASSERT(m_pListBox != NULL);
	TCHAR buf[256];
	VERIFY(GetDateFormat(m_id, 0, &m_time, lpszFormatString, buf, 256));
	if (m_pListBox->FindStringExact(-1,buf) == CB_ERR)
		m_pListBox->AddString(buf);
	return TRUE;
}

BOOL CALLBACK CDateTimeDlg::TimeFmtEnumProc(LPTSTR lpszFormatString)
{
	ASSERT(m_pListBox != NULL);
	TCHAR buf[256];
	VERIFY(GetTimeFormat(m_id, 0, &m_time, lpszFormatString, buf, 256));
	if (m_pListBox->FindStringExact(-1,buf) == CB_ERR)
		m_pListBox->AddString(buf);
	return TRUE;
}

void CDateTimeDlg::OnDblclkDatetimeList()
{
	// TODO: Add your control notification handler code here
	OnOK();
}

BOOL CDateTimeDlg::OnInitDialog()
{
	CMyDialog::OnInitDialog();

	m_pListBox = &m_listbox; // set static member
	GetLocalTime(&m_time);
	m_id = GetUserDefaultLCID();

	EnumDateFormats(DateFmtEnumProc, m_id, DATE_SHORTDATE);
	EnumDateFormats(DateFmtEnumProc, m_id, DATE_LONGDATE);
	EnumTimeFormats(TimeFmtEnumProc, m_id, 0);

	m_pListBox = NULL;
	m_listbox.SetCurSel(0);

	return TRUE;  // return TRUE unless you set the focus to a control
		// EXCEPTION: OCX Property Pages should return FALSE
}


// AddressDlg.cpp : implementation file
```

```
#include "stdafx.h"
#include "Project.h"
#include "AddressDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CComboBox* CAddressDlg::m_pComboCountry = NULL;
CComboBox* CAddressDlg::m_pComboState = NULL;
CAddressDlg::CAddressDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CAddressDlg::IDD, pParent)
{
        amastates[0] = IDS_STATE1;
        amastates[1] = IDS_STATE2;
        amastates[2] = IDS_STATE3;
        amastates[3] = IDS_STATE4;
        amastates[4] = IDS_STATE5;
        amastates[5] = IDS_STATE6;
        chsprovinces[0] = IDS_PROVINCE1;
        chsprovinces[1] = IDS_PROVINCE2;
        chsprovinces[2] = IDS_PROVINCE3;
        chsprovinces[3] = IDS_PROVINCE4;
        chsprovinces[4] = IDS_PROVINCE5;
        chsprovinces[5] = IDS_PROVINCE6;
        //{{AFX_DATA_INIT(CAddressDlg)
        m_strCty = _T("");
        m_strState = _T("");
        m_strAdr1 = _T("");
        m_strAdr2 = _T("");
        m_strZip = _T("");
        //}}AFX_DATA_INIT
}

void CAddressDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CAddressDlg)
        DDX_Control(pDX, IDC_COMBO_SATATE, m_comboState);
        DDX_Control(pDX, IDC_COMBO_COUNTRY, m_comboCty);
        DDX_CBString(pDX, IDC_COMBO_COUNTRY, m_strCty);
        DDX_CBString(pDX, IDC_COMBO_SATATE, m_strState);
        DDX_Text(pDX, IDC_EDIT_ADR1, m_strAdr1);
        DDX_Text(pDX, IDC_EDIT_ADR2, m_strAdr2);
        DDX_Text(pDX, IDC_EDIT_ZIP, m_strZip);
        //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAddressDlg, CDialog)
        //{{AFX_MSG_MAP(CAddressDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CAddressDlg::OnInitDialog()
```

```
{
	CDialog::OnInitDialog();
	m_pComboState = &m_comboState;
	m_pComboCountry = &m_comboCty; //set static member
	CString str;

	str.LoadString(IDS_AMERICA);
	m_pComboCountry->AddString(str);

	str.LoadString(IDS_CANADA);
	m_pComboCountry->AddString(str);

	str.LoadString(IDS_CHINA);
	m_pComboCountry->AddString(str);

	str.LoadString(IDS_JAPAN);
	m_pComboCountry->AddString(str);

	str.LoadString(IDS_CANADA);
	m_pComboCountry->AddString(str);

	int acp = ((CProjectApp*)(AfxGetApp()))->localeACP;
	if(acp == PROJ_UINT_ENGLISH)
	{
		for(int i=0; i<6; i++)
		{
			str.LoadString(amastates[i]);
			m_pComboState->AddString(str);
		}
		m_pComboState->SetCurSel(0);
		m_pComboCountry->SetCurSel(0);
	}

	else if (acp== PROJ_UINT_CHINESE)
	{
		for(int i=0; i<6; i++)
		{
			str.LoadString(chsprovinces[i]);
			m_pComboState->AddString(str);
		}
		m_pComboState->SetCurSel(0);
		m_pComboCountry->SetCurSel(2);
	}


	return TRUE;  // return TRUE unless you set the focus to a control
		// EXCEPTION: OCX Property Pages should return FALSE
}

// GoLineDlg.cpp : implementation file

#include "stdafx.h"
#include "Project.h"
#include "GoLineDlg.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CGoLineDlg dialog


CGoLineDlg::CGoLineDlg(CWnd* pParent /*=NULL*/)
        : CDialog(CGoLineDlg::IDD, pParent)
{
        //{{AFX_DATA_INIT(CGoLineDlg)
        m_nLine = 1;
        //}}AFX_DATA_INIT
}


void CGoLineDlg::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CGoLineDlg)
        DDX_Text(pDX, IDC_EDIT_GOLINE, m_nLine);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CGoLineDlg, CDialog)
        //{{AFX_MSG_MAP(CGoLineDlg)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()


void CGoLineDlg::OnCancel()
{

        CDialog::OnCancel();
}
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "Project.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

IMPLEMENT_DYNCREATE(CMainFrame, CFrameWnd)

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
```

```
        //{{AFX_MSG_MAP(CMainFrame)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        ON_WM_CREATE()
        //}}AFX_MSG_MAP
        // Global help commands
        ON_COMMAND(ID_HELP_FINDER, CFrameWnd::OnHelpFinder)
        ON_COMMAND(ID_HELP, CFrameWnd::OnHelp)
        ON_COMMAND(ID_CONTEXT_HELP, CFrameWnd::OnContextHelp)
        ON_COMMAND(ID_DEFAULT_HELP, CFrameWnd::OnHelpFinder)
END_MESSAGE_MAP()

static UINT indicators[] =
{
        ID_SEPARATOR,           // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

CMainFrame::CMainFrame()
{}

CMainFrame::~CMainFrame()
{}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CFrameWnd::OnCreate(lpCreateStruct) == -1)
                return -1;

        if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP
                | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
                !m_wndToolBar.LoadToolBar(IDR_MAINFRAME))
        {
                TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }

        if (!m_wndStatusBar.Create(this) ||
                !m_wndStatusBar.SetIndicators(indicators,
                  sizeof(indicators)/sizeof(UINT)))
        {
                TRACE0("Failed to create status bar\n");
                return -1;      // fail to create
        }

        if (!m_wndClrBar.Create(this, WS_CHILD | WS_VISIBLE | CBRS_TOP,
        IDR_MYCLR_TOOLBAR) ||
                !m_wndClrBar.LoadToolBar(IDR_MYCLR_TOOLBAR))
        {
                TRACE0("Failed to create color toolbar\n");
                return -1;      // fail to create
        }
```

```
	m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
	EnableDocking(CBRS_ALIGN_ANY);
	DockControlBar(&m_wndToolBar);

	m_wndClrBar.SetBarStyle(m_wndClrBar.GetBarStyle() |
					CBRS_TOOLTIPS | CBRS_FLYBY |
					CBRS_SIZE_DYNAMIC);
	m_wndClrBar.EnableDocking(CBRS_ALIGN_ANY);
	DockControlBar(&m_wndClrBar, AFX_IDW_DOCKBAR_LEFT);

	return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
	if( !CFrameWnd::PreCreateWindow(cs) )
		return FALSE;
	// TODO: Modify the Window class or styles here by modifying
	//  the CREATESTRUCT cs
	return TRUE;
}

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
	CFrameWnd::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
	CFrameWnd::Dump(dc);
}

#endif //_DEBUG

// MyDialog.cpp : implementation file
//

#include "stdafx.h"
#include "Project.h"
#include "MyDialog.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

CMyDialog::CMyDialog(UINT nIDTemplate, CWnd* pParentWnd)
	: CDialog(nIDTemplate, pParentWnd)
{
}

CMyDialog::CMyDialog(LPCTSTR lpszTemplateName, CWnd* pParentWnd)
	: CDialog(lpszTemplateName, pParentWnd)
{
```

```
}

CMyDialog::CMyDialog() : CDialog()
{
}


void CMyDialog::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CMyDialog)
                // NOTE: the ClassWizard will add DDX and DDV calls here
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CMyDialog, CDialog)
        //{{AFX_MSG_MAP(CMyDialog)
        //}}AFX_MSG_MAP
        ON_MESSAGE(WM_HELP, OnHelp)
        ON_MESSAGE(WM_CONTEXTMENU, OnHelpContextMenu)
END_MESSAGE_MAP()

LONG CMyDialog::OnHelp(UINT, LONG lParam)
{
        ::WinHelp( (HWND)((LPHELPINFO)lParam)->hItemHandle, AfxGetApp()-
>m_pszHelpFilePath,
                HELP_WM_HELP, (DWORD)(LPVOID)GetHelpIDs());
        return 0;
}

LONG CMyDialog::OnHelpContextMenu(UINT wParam, LONG)
{
        ::WinHelp((HWND)wParam, AfxGetApp()->m_pszHelpFilePath,
                HELP_CONTEXTMENU, (DWORD)(LPVOID)GetHelpIDs());
        return 0;
}

BOOL CMyDialog::OnInitDialog()
{
        CDialog::OnInitDialog();
        ModifyStyleEx(0, WS_EX_CONTEXTHELP);
        return TRUE;  // return TRUE unless you set the focus to a control
                                // EXCEPTION: OCX Property Pages should return FALSE
}

// MySplashWnd.cpp : implementation file
//

#include "stdafx.h"
#include "Project.h"
#include "MySplashWnd.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
#endif

BOOL CMySplashWnd::Create(CWnd * pParent)
{
	if (!CDialog::Create(CMySplashWnd::IDD, pParent))
	{
		TRACE0("Warning: creation of CSplashWnd dialog failed\n");
		return FALSE;
	}

	return TRUE;
}

CMySplashWnd::CMySplashWnd(CWnd* pParent /*=NULL*/)
	: CDialog(CMySplashWnd::IDD, pParent)
{
	//{{AFX_DATA_INIT(CMySplashWnd)
		// NOTE: the ClassWizard will add member initialization here
	//}}AFX_DATA_INIT
}

void CMySplashWnd::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CMySplashWnd)
		// NOTE: the ClassWizard will add DDX and DDV calls here
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CMySplashWnd, CDialog)
	//{{AFX_MSG_MAP(CMySplashWnd)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

BOOL CMySplashWnd::OnInitDialog()
{
	CDialog::OnInitDialog();
	CenterWindow();

	return TRUE;
}

// Project.cpp : Defines the class behaviors for the application.

#include "stdafx.h"
#include "Project.h"

#include "MainFrm.h"
#include "ProjectDoc.h"
#include "ProjectView.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
```

```
#endif

BEGIN_MESSAGE_MAP(CProjectApp, CWinApp)
	//{{AFX_MSG_MAP(CProjectApp)
	ON_COMMAND(ID_APP_ABOUT, OnAppAbout)
		// NOTE - the ClassWizard will add and remove mapping macros here.
		//    DO NOT EDIT what you see in these blocks of generated code!
	//}}AFX_MSG_MAP
	// Standard file based document commands
	ON_COMMAND(ID_FILE_NEW, CWinApp::OnFileNew)
	ON_COMMAND(ID_FILE_OPEN, CWinApp::OnFileOpen)
	// Standard print setup command
	ON_COMMAND(ID_FILE_PRINT_SETUP, CWinApp::OnFilePrintSetup)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CProjectApp construction

CProjectApp::CProjectApp()
{
	localeACP = GetACP();
}

CProjectApp theApp;

BOOL CProjectApp::InitInstance()
{
	AfxEnableControlContainer();
#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic(); // Call this when linking to MFC statically
#endif

	localeACP = GetACP();
	if(localeACP == PROJ_UINT_ENGLISH)
		m_hInstResDLL = LoadLibrary("ResourceDll.dll");
	else if (localeACP == PROJ_UINT_CHINESE)
		m_hInstResDLL = LoadLibrary("ResourceDllchsd.dll");
	//ASSERT(m_hInstResDLL != NULL);
	//it is not necessary to call AfxSetResourceHandle() at this point

	SetRegistryKey(_T("Local AppWizard-Generated Applications"));
	LoadStdProfileSettings();  // Load standard INI file options (including MRU)
	CProjectView::InitailizeView();
	CSingleDocTemplate* pDocTemplate;
	pDocTemplate = new CSingleDocTemplate(
		IDR_MAINFRAME,
		RUNTIME_CLASS(CProjectDoc),
		RUNTIME_CLASS(CMainFrame),	// main SDI frame window
		RUNTIME_CLASS(CProjectView));
	AddDocTemplate(pDocTemplate);

	EnableShellOpen();
	RegisterShellFileTypes(TRUE);
```

```
	CCommandLineInfo cmdInfo;
	ParseCommandLine(cmdInfo);

	if (!ProcessShellCommand(cmdInfo))
		return FALSE;

	m_pMainWnd->ShowWindow(SW_SHOW);
	m_pMainWnd->UpdateWindow();

	m_splash.Create(m_pMainWnd);
	m_splash.ShowWindow(SW_SHOW);
	m_splash.UpdateWindow();
	m_splash.SetTimer(1, 300, NULL);


	m_dwSplash = ::GetCurrentTime();
	m_pMainWnd->DragAcceptFiles();
	CString str;
	str.LoadString(IDS_PROJECT_TITLE);

	m_pMainWnd->SetWindowText((LPCTSTR)(str));

	return TRUE;
}

class CAboutDlg : public CDialog
{
public:
	CAboutDlg();

// Dialog Data
	//{{AFX_DATA(CAboutDlg)
	enum { IDD = IDD_ABOUTBOX };
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAboutDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	//{{AFX_MSG(CAboutDlg)
		// No message handlers
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
	//{{AFX_DATA_INIT(CAboutDlg)
	//}}AFX_DATA_INIT
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
```

```
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAboutDlg)
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
	//{{AFX_MSG_MAP(CAboutDlg)
		// No message handlers
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

// App command to run the dialog
void CProjectApp::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}

BOOL CProjectApp::OnIdle(LONG lCount)
{
	// call base class idle first
	BOOL bResult = CWinApp::OnIdle(lCount);

	// do our own work
	if (m_splash.m_hWnd != NULL)
	{
		if (::GetCurrentTime() - m_dwSplash > 1500)
		{
			// timeout expired, destroy the splash window
			m_splash.DestroyWindow();
			m_pMainWnd->UpdateWindow();
		}
		else
		{
			bResult = TRUE;
		}
	}

	return bResult;
}

int CProjectApp::ExitInstance()
{
	FreeLibrary(m_hInstResDLL);
	return CWinApp::ExitInstance();
}
// ProjectDoc.cpp : implementation of the CProjectDoc class
//

#include "stdafx.h"
#include "Project.h"

#include "ProjectDoc.h"

#ifdef _DEBUG
```

```
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CProjectDoc

IMPLEMENT_DYNCREATE(CProjectDoc, CDocument)

BEGIN_MESSAGE_MAP(CProjectDoc, CDocument)
        //{{AFX_MSG_MAP(CProjectDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
        //color handling
END_MESSAGE_MAP()

CProjectDoc::CProjectDoc()
{}

CProjectDoc::~CProjectDoc()
{}

BOOL CProjectDoc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;
        ((CEditView*)m_viewList.GetHead())->SetWindowText(NULL);
        return TRUE;
}

void CProjectDoc::Serialize(CArchive& ar)
{
        // CEditView contains an edit control which handles all serialization
        ((CEditView*)m_viewList.GetHead())->SerializeRaw(ar);
}

#ifdef _DEBUG
void CProjectDoc::AssertValid() const
{
        CDocument::AssertValid();
}

void CProjectDoc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG

// ProjectView.cpp : implementation of the CProjectView class
//

#include "stdafx.h"
#include "Project.h"
```

```
#include "ProjectDoc.h"
#include "ProjectView.h"
#include "DateTimeDlg.h"
#include "SetTabStops.h"
#include "AddressDlg.h"
#include "GoLineDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CProjectView

IMPLEMENT_DYNCREATE(CProjectView, CEditView)

BEGIN_MESSAGE_MAP(CProjectView, CEditView)
        //{{AFX_MSG_MAP(CProjectView)
        ON_COMMAND(ID_INSERT_DATEANDTIME, OnInsertDateandtime)
        ON_COMMAND(ID_CHOOSE_FONT, OnChooseFont)
        ON_COMMAND(ID_FORMAT_TABSTOP, OnFormatTabstop)
        ON_COMMAND(ID_INSERT_COUNTCHARACTERS, OnInsertCount)
        ON_COMMAND(ID_FORMAT_ADDRESS, OnFormatAddress)
        ON_COMMAND(ID_INSERT_PASTEADDRESS, OnInsertPasteaddress)
        ON_COMMAND(ID_EDIT_GOTO, OnEditGoto)
        ON_WM_CONTEXTMENU()
        ON_COMMAND(ID_FORMAT_SETPRINTERFONT, OnFormatSetprinterfont)
        ON_COMMAND(ID_INSERT_CHARCOUNT, OnInsertCharcount)
        //}}AFX_MSG_MAP
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CEditView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CEditView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CEditView::OnFilePrintPreview)
        ON_COMMAND_RANGE(ID_COLOR0, ID_COLOR15, OnColor)
        ON_UPDATE_COMMAND_UI_RANGE(ID_COLOR0, ID_COLOR15, OnUpdateColor)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CProjectView construction/destruction
UINT CProjectView::m_nDefTabStops;
UINT CProjectView::m_nDefTabStopsOld;
BOOL CProjectView::m_bDefWordWrap;
BOOL CProjectView::m_bDefWordWrapOld;
LOGFONT NEAR CProjectView::m_lfDefFont;
LOGFONT NEAR CProjectView::m_lfDefFontOld;
LOGFONT NEAR CProjectView::m_lfDefPrintFont;
LOGFONT NEAR CProjectView::m_lfDefPrintFontOld;

int CProjectView::s_ColorMap[] =
                                        {
                                                0,              //black
                                                1,              //dark red
                                                2,              //dark green
                                                3,              //light brown
```

```
                4,              //dark blue
                5,              //purple
                6,              //dark cyan
                12,     //gray
                7,              //light gray
                13,     //red
                14,     //green
                15,     //yellow
                16,     //blue
                17,     //magenta
                18,     //cyan
                19              //white
};

static TCHAR BASED_CODE szSettings[] = _T("Settings");
static TCHAR BASED_CODE szTabStops[] = _T("TabStops");
static TCHAR BASED_CODE szFont[] = _T("Font");
static TCHAR BASED_CODE szPrintFont[] = _T("PrintFont");
static TCHAR BASED_CODE szHeight[] = _T("Height");
static TCHAR BASED_CODE szWeight[] = _T("Weight");
static TCHAR BASED_CODE szItalic[] = _T("Italic");
static TCHAR BASED_CODE szUnderline[] = _T("Underline");
static TCHAR BASED_CODE szPitchAndFamily[] = _T("PitchAndFamily");
static TCHAR BASED_CODE szCharSet[] = _T("CharSet");
static TCHAR BASED_CODE szFaceName[] = _T("FaceName");
static TCHAR BASED_CODE szSystem[] = _T("System");
static TCHAR BASED_CODE szWordWrap[] = _T("WordWrap");

static void GetProfileFont(LPCTSTR szSec, LOGFONT* plf)
{
        CWinApp* pApp = AfxGetApp();
        plf->lfHeight = pApp->GetProfileInt(szSec, szHeight, 0);
        if (plf->lfHeight != 0)
        {
                plf->lfWeight = pApp->GetProfileInt(szSec, szWeight, 0);
                plf->lfItalic = (BYTE)pApp->GetProfileInt(szSec, szItalic, 0);
                plf->lfUnderline = (BYTE)pApp->GetProfileInt(szSec, szUnderline, 0);
                plf->lfPitchAndFamily = (BYTE)pApp->GetProfileInt(szSec, szPitchAndFamily, 0);
                plf->lfCharSet = (BYTE)pApp->GetProfileInt(szSec, szCharSet,
DEFAULT_CHARSET);
                CString strFont = pApp->GetProfileString(szSec, szFaceName, szSystem);
                lstrcpyn((TCHAR*)plf->lfFaceName, strFont, sizeof plf->lfFaceName);
                plf->lfFaceName[sizeof plf->lfFaceName-1] = 0;
        }
}

static void WriteProfileFont(LPCTSTR szSec, const LOGFONT* plf, LOGFONT* plfOld)
{
        CWinApp* pApp = AfxGetApp();

        if (plf->lfHeight != plfOld->lfHeight)
                pApp->WriteProfileInt(szSec, szHeight, plf->lfHeight);
        if (plf->lfHeight != 0)
        {
                if (plf->lfHeight != plfOld->lfHeight)
                        pApp->WriteProfileInt(szSec, szHeight, plf->lfHeight);
```

```
                if (plf->lfWeight != plfOld->lfWeight)
                        pApp->WriteProfileInt(szSec, szWeight, plf->lfWeight);
                if (plf->lfItalic != plfOld->lfItalic)
                        pApp->WriteProfileInt(szSec, szItalic, plf->lfItalic);
                if (plf->lfUnderline != plfOld->lfUnderline)
                        pApp->WriteProfileInt(szSec, szUnderline, plf->lfUnderline);
                if (plf->lfPitchAndFamily != plfOld->lfPitchAndFamily)
                        pApp->WriteProfileInt(szSec, szPitchAndFamily, plf->lfPitchAndFamily);
                if (plf->lfCharSet != plfOld->lfCharSet)
                        pApp->WriteProfileInt(szSec, szCharSet, plf->lfCharSet);
                if (_tcscmp(plf->lfFaceName, plfOld->lfFaceName) != 0)
                        pApp->WriteProfileString(szSec, szFaceName, (LPCTSTR)plf->lfFaceName);
        }
        *plfOld = *plf;
}

CProjectView::CProjectView()
{
        m_pGoLine = new CGoLineDlg;
        m_bAddress = FALSE;
        UINT lclACP = ((CProjectApp*)AfxGetApp())->localeACP;
        GetCPInfo(lclACP, &CPInfo);
        vbLBRange = CPInfo.LeadByte;
        vfDBCS = (CPInfo.MaxCharSize > 1);
        //check to see if the max length in bytes of page more than 1
}

CProjectView::~CProjectView()
{
        delete m_pGoLine;
}

BOOL CProjectView::PreCreateWindow(CREATESTRUCT& cs)
{
        BOOL bPreCreated = CEditView::PreCreateWindow(cs);
        cs.style &= ~(ES_AUTOHSCROLL|WS_HSCROLL);       // Enable word-wrapping
        return bPreCreated;
}

void CProjectView::OnDraw(CDC* pDC)
{
        CProjectDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);

        pDC->SetTextColor(m_crColor);

}

BOOL CProjectView::OnPreparePrinting(CPrintInfo* pInfo)
{
        // default CEditView preparation
        return CEditView::OnPreparePrinting(pInfo);
}

void CProjectView::OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo)
{
```

```
	// Default CEditView begin printing.
	CEditView::OnBeginPrinting(pDC, pInfo);
}

void CProjectView::OnEndPrinting(CDC* pDC, CPrintInfo* pInfo)
{
	// Default CEditView end printing
	CEditView::OnEndPrinting(pDC, pInfo);
}

#ifdef _DEBUG
void CProjectView::AssertValid() const
{
	CEditView::AssertValid();
}

void CProjectView::Dump(CDumpContext& dc) const
{
	CEditView::Dump(dc);
}

CProjectDoc* CProjectView::GetDocument() // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CProjectDoc)));
	return (CProjectDoc*)m_pDocument;
}
#endif //_DEBUG

/////////////////////////////////////////////////////////////////////////////
// CProjectView message handlers

void CProjectView::OnInsertDateandtime()
{
	CDateTimeDlg dlg;
	if (dlg.DoModal() == IDOK)
	GetEditCtrl().ReplaceSel(dlg.m_selection);
}

void CProjectView::OnChooseFont()
{
 // get current font description
  CFont* pFont = GetFont();
  LOGFONT lf;
  if (pFont != NULL)
		pFont->GetObject(sizeof(LOGFONT), &lf);
  else
		::GetObject(GetStockObject(SYSTEM_FONT), sizeof(LOGFONT), &lf);

	CFontDialog dlg(&lf, CF_SCREENFONTS|CF_INITTOLOGFONTSTRUCT);
	if (dlg.DoModal() == IDOK)
	{
		// switch to new font.
		m_font.DeleteObject();
		if (m_font.CreateFontIndirect(&lf))
		{
			CWaitCursor wait;
```

```
                    SetFont(&m_font);
                    m_lfDefFont = lf;

                }
        }
}

void CProjectView::InitailizeView()
{
        CWinApp* pApp = AfxGetApp();
        m_bDefWordWrap = pApp->GetProfileInt(szSettings, szWordWrap, 0);
        m_bDefWordWrapOld = m_bDefWordWrap;
        m_nDefTabStops = pApp->GetProfileInt(szSettings, szTabStops, 8*4);
        m_nDefTabStopsOld = m_nDefTabStops;
        GetProfileFont(szFont, &m_lfDefFont);
        m_lfDefFontOld = m_lfDefFont;
        GetProfileFont(szPrintFont, &m_lfDefPrintFont);
        m_lfDefPrintFontOld = m_lfDefPrintFont;
}

void CProjectView::OnFormatTabstop()
{
        CSetTabStops dlg;
        int l_itabs;
        dlg.m_nTabStops = m_nTabStops/4;
        l_itabs = dlg.m_nTabStops;

        if (dlg.DoModal() == IDOK)
        {
                CWaitCursor wait;
                if(dlg.m_Unit == 0)
                {
                        int tabs = (int) (dlg.m_nTabStops * 2.54 * 20) / 4;
                        l_itabs = tabs / 4;
                }
                else if(dlg.m_Unit == 1)
                {
                        int tabs = (dlg.m_nTabStops * 20)/4;
                        l_itabs = tabs / 4;
                }

                SetTabStops(l_itabs*4);
                m_nDefTabStops = m_nTabStops;
        }

}

void CProjectView::OnInitialUpdate()
{
        CEditView::OnInitialUpdate();
        //AfxGetMainWnd()-> SetWindowText("Hangbo Zhang Thesis Project ");
        //AfxGetMainWnd()->MoveWindow(20, 20, 600, 500);
}

void CProjectView::OnInsertCount()
{
```

```
	//int acp = ((CProjectApp*)AfxGetApp())->localeACP;
	int lines = GetEditCtrl().GetLineCount();
	char LINE[64];

	for(int i=0; i<lines; i++)
	{
		if( GetEditCtrl().GetLine(i, LINE) );
	}


	CString str;
	str.LoadString(IDS_COUNT_CHAR);
	int nlen = GetBufferLength();
	TCHAR buf[64];
	wsprintf(buf, str, nlen);
	AfxMessageBox((LPCTSTR)(buf));
}

void CProjectView::OnFormatAddress()
{
	CAddressDlg adrDlg;
	if(adrDlg.DoModal() == IDOK)
	{
		m_strAdr1 = adrDlg.m_strAdr1;
		m_strAdr2 = adrDlg.m_strAdr2;
		m_strState = adrDlg.m_strState;
		m_strzip = adrDlg.m_strZip;
		m_strCty = adrDlg.m_strCty;
		m_bAddress = TRUE;
	}
}

void CProjectView::OnInsertPasteaddress()
{
	if( !m_bAddress)
	{
		CString str;
		str.LoadString(IDS_NO_ADDRESS);
		AfxMessageBox(str);
		return;
	}

	CString str = "";
	int acp = ((CProjectApp*)AfxGetApp())->localeACP;
	if(acp == PROJ_UINT_ENGLISH)
	{
		str += m_strAdr1;
		str += "\r\n";
		str += m_strAdr2;
		str += "\r\n";
		str += m_strState + ",  " + m_strzip + "\r\n" + m_strCty;
	}

	if(acp == PROJ_UINT_CHINESE)
	{
		str += m_strCty + "  " + m_strzip;
```

```
                str += "\t" + m_strState;
                str += "  " + m_strAdr1;
                str += m_strAdr2;
        }

        GetEditCtrl().ReplaceSel(str);
}


void CProjectView::OnEditGoto()
{
        if(m_pGoLine->DoModal() != IDOK)
                return;

        if(m_pGoLine->m_nLine < 1)
        {
                CString str;
                str.LoadString(IDS_INVALID_LINE);
                AfxMessageBox(str);
                return;
        }
        CEdit &edit = GetEditCtrl();
        int i = edit.LineFromChar();          // this is the current line the cursor is on
        int nLine = m_pGoLine->m_nLine; // line number to go to

        if(nLine > (edit.GetLineCount()))
        {
                CString str2;

                str2.LoadString(IDS_BIG_LINENUM);
                TCHAR buf[64];
                wsprintf(buf, str2, nLine);
                 str2 = buf;
                AfxMessageBox(str2);
                return;
        }

        // move window and caret
        --nLine;                // edit control is zero based
        edit.LineScroll(nLine-i);  // new line number - the current line

        int idx;
        idx = edit.LineIndex(nLine);
        edit.SetSel(idx, idx);
}

void CProjectView::OnContextMenu(CWnd* pWnd, CPoint point)
{
                CMenu menuText;
                menuText.LoadMenu(IDR_TEXT_POPUP);
                CMenu* pMenuPopup = menuText.GetSubMenu(0);
                //menuText.RemoveMenu(0, MF_BYPOSITION);
                //return pMenuPopup->Detach();
                pMenuPopup->TrackPopupMenu(TPM_LEFTALIGN | TPM_LEFTBUTTON,
                        point.x, point.y,
                        AfxGetMainWnd(), NULL);
```

```
}
COLORREF CProjectView::GetColorRef(UINT nID)
{
        ASSERT( nID >= ID_COLOR0 );
        ASSERT( nID <= ID_COLOR15 );
        CPalette* pPal = CPalette::FromHandle(
                                                        (HPALETTE) GetStockObject(
DEFAULT_PALETTE ) );
        ASSERT( pPal != NULL );
        PALETTEENTRY pe;
        if( pPal->GetPaletteEntries( s_ColorMap[ nID-ID_COLOR0 ], 1, &pe ) != 0 )
        {
                return RGB( pe.peRed, pe.peGreen, pe.peBlue );
        }
        else
        {
                TRACE1( "Unable to find palette color entry for ID=%d\n", nID );
                return ::GetSysColor( COLOR_WINDOWTEXT );
        }
}


void CProjectView::OnColor(UINT nID)
{
        /*m_crColor = GetColorRef(nID);
        CDC dcScreen;
        dcScreen.Attach(::GetDC(NULL));
        dcScreen.SetTextColor(m_crColor);*/
}

void CProjectView::OnUpdateColor(CCmdUI *pCmdUI)
{

}

static void ScaleLogFont(LPLOGFONT plf, const CDC& dcFrom, const CDC& dcTo)
        // helper to scale log font member from one DC to another!
{
        plf->lfHeight = MulDiv(plf->lfHeight,
                dcTo.GetDeviceCaps(LOGPIXELSY), dcFrom.GetDeviceCaps(LOGPIXELSY));
        plf->lfWidth = MulDiv(plf->lfWidth,
                dcTo.GetDeviceCaps(LOGPIXELSX), dcFrom.GetDeviceCaps(LOGPIXELSX));
}

void CProjectView::OnFormatSetprinterfont()
{

        CWaitCursor wait;
        CFont* pFont = GetPrinterFont();
        LOGFONT lf;
        LPLOGFONT plf = NULL;
        if (pFont != NULL)
        {
                pFont->GetObject(sizeof(LOGFONT), &lf);
                plf = &lf;
        }
```

```
        // magic to get printer dialog that would be used if we were printing!
        CPrintDialog dlgPrint(FALSE);
        if (!AfxGetApp()->GetPrinterDeviceDefaults(&dlgPrint.m_pd))
        {
                AfxMessageBox(IDP_ERR_GET_DEVICE_DEFAULTS);
                return;
        }
        wait.Restore();
        HDC hdcPrint = dlgPrint.CreatePrinterDC();
        if (hdcPrint == NULL)
        {
                AfxMessageBox(IDP_ERR_GET_PRINTER_DC);
                return;
        }

        CDC dcScreen;
        dcScreen.Attach(::GetDC(NULL));
        CDC dcPrint;
        dcPrint.Attach(hdcPrint);

        if (plf != NULL)
        {
                // need to map initial logfont to screen metrics.
                ::ScaleLogFont(plf, dcPrint, dcScreen);
        }
        CFontDialog dlg(plf, CF_PRINTERFONTS, &dcPrint);
        if (dlg.DoModal() == IDOK)
        {
                // map the resulting logfont back to printer metrics.
                lf = dlg.m_lf;
                ::ScaleLogFont(&lf, dcScreen, dcPrint);

                SetPrinterFont(NULL);
                m_fontPrint.DeleteObject();
                if (m_fontPrint.CreateFontIndirect(&lf))
                {
                        SetPrinterFont(&m_fontPrint);
                        m_lfDefPrintFont = lf;

                }
        }
        ::ReleaseDC(NULL, dcScreen.Detach());
}

void CProjectView::OnInsertCharcount()
{

        CString str;
        str.LoadString(IDS_COUNT_CHAR);

        CEdit &edit = GetEditCtrl();
        int lines = edit.GetLineCount();

        TCHAR *charcnt;
        charcnt = (TCHAR*) malloc(100*sizeof(TCHAR));
```

```
	int count = 0;
	for(int i=0; i<lines; i++)
	{
		edit.GetLine(i, charcnt, 100);
		if(vfDBCS)
		{
			for(count; *charcnt; charcnt++)// = MyCharNext(charcnt) )
			{
			if((*charcnt != -51) && (*charcnt != -3) && (*charcnt != -111) )
			{
				if(IsDBCSLeadByte(*charcnt))
					charcnt++;
				++count;
				}
			}
		}
		else
		{
		for(count; (*charcnt); charcnt++)
			{
				if((*charcnt != -51) && (*charcnt != -3)
				&& (*charcnt != -111) )
				++count;
			}
		}
	}
	//int nlen = GetBufferLength();

	TCHAR buf[64];
	wsprintf(buf, str, count);
	AfxMessageBox((LPCTSTR)(buf));
}

char* CProjectView::MyCharNext(char* pszStr)
{
	BYTE bRange = 0;
	while((bRange < 12 ) && (vbLBRange[bRange] != NULL) )
	{
		if((*pszStr >= vbLBRange[bRange]) &&
			(*pszStr <= vbLBRange[bRange + 1]) )
			return (pszStr + 2); //skip two bytes
		bRange += 2;
	}
	return (pszStr + 1);
}

// SetTabStops.cpp : implementation file
//

#include "stdafx.h"
#include "Project.h"
#include "SetTabStops.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
```

```
static char THIS_FILE[] = __FILE__;
#endif

CSetTabStops::CSetTabStops(CWnd* pParent /*=NULL*/)
        : CDialog(CSetTabStops::IDD, pParent)
{
        //{{AFX_DATA_INIT(CSetTabStops)
        m_nTabStops = 0;
        m_Unit = 2;
        //}}AFX_DATA_INIT
}


void CSetTabStops::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        //{{AFX_DATA_MAP(CSetTabStops)
        DDX_Text(pDX, IDC_EDIT_TAB, m_nTabStops);
        DDV_MinMaxUInt(pDX, m_nTabStops, 1, 200);
        DDX_Radio(pDX, IDC_RADIO_INCH, m_Unit);
        //}}AFX_DATA_MAP
}


BEGIN_MESSAGE_MAP(CSetTabStops, CDialog)
        //{{AFX_MSG_MAP(CSetTabStops)
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CSetTabStops message handlers

BOOL CSetTabStops::OnInitDialog()
{
        CDialog::OnInitDialog();
//      m_nTabStops = 4;
//      Invalidate();
        return TRUE;  // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}
#if !defined(AFX_ADDRESSDLG_H__882DFA31_1793_11D4_9E17_006008AFEE67__INCLUDED_)
#define AFX_ADDRESSDLG_H__882DFA31_1793_11D4_9E17_006008AFEE67__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// AddressDlg.h : header file

class CAddressDlg : public CDialog
{
// Construction
private:
        UINT amastates[6];
        UINT chsprovinces[6];
public:
        static CComboBox* m_pComboState;
```

```
        static CComboBox* m_pComboCountry;
public:
        CAddressDlg(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CAddressDlg)
        enum { IDD = IDD_ADDRESS };
        CComboBox      m_comboState;
        CComboBox      m_comboCty;
        CString  m_strCty;
        CString  m_strState;
        CString  m_strAdr1;
        CString  m_strAdr2;
        CString  m_strZip;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CAddressDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CAddressDlg)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_ADDRESSDLG_H__882DFA31_1793_11D4_9E17_006008AFEE67__INCLUDED_)
#if
!defined(AFX_DATETIMEDLG_H__D1AE6AA5_1702_11D4_9E15_006008AFEE67__INCLUDED_)
#define AFX_DATETIMEDLG_H__D1AE6AA5_1702_11D4_9E15_006008AFEE67__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DateTimeDlg.h : header file
//

#include "MyDialog.h"
/////////////////////////////////////////////////////////////////////////////
// CDateTimeDlg dialog

class CDateTimeDlg : public CMyDialog
{
// Construction
```

```
public:
        CDateTimeDlg(CWnd* pParent = NULL);  // standard constructor
                // Attributes
        static SYSTEMTIME m_time;
        static LCID m_id;
        static CListBox* m_pListBox;
        static BOOL CALLBACK DateFmtEnumProc(LPTSTR lpszFormatString);
        static BOOL CALLBACK TimeFmtEnumProc(LPTSTR lpszFormatString);


// Dialog Data
        //{{AFX_DATA(CDateTimeDlg)
        enum { IDD = IDD_DATETIMEDLG };
        CListBox        m_listbox;
        CString m_selection;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CDateTimeDlg)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        static const DWORD m_nHelpIDs[];
        virtual const DWORD* GetHelpIDs() {return m_nHelpIDs;}

        // Generated message map functions
        //{{AFX_MSG(CDateTimeDlg)
        virtual BOOL OnInitDialog();
        afx_msg void OnDblclkDatetimeList();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_DATETIMEDLG_H__D1AE6AA5_1702_11D4_9E15_006008AFEE67__INCLUDED_)
#if !defined(AFX_GOLINEDLG_H__129B1510_184C_11D4_9E18_006008AFEE67__INCLUDED_)
#define AFX_GOLINEDLG_H__129B1510_184C_11D4_9E18_006008AFEE67__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// GoLineDlg.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CGoLineDlg dialog
```

```
class CGoLineDlg : public CDialog
{
// Construction
public:
	CGoLineDlg(CWnd* pParent = NULL);  // standard constructor

// Dialog Data
	//{{AFX_DATA(CGoLineDlg)
	enum { IDD = IDD_GOLINE_DIALOG };
	int		m_nLine;
	//}}AFX_DATA


// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CGoLineDlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:

	// Generated message map functions
	//{{AFX_MSG(CGoLineDlg)
	virtual void OnCancel();
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_GOLINEDLG_H__129B1510_184C_11D4_9E18_006008AFEE67__INCLUDED_)
// MainFrm.h : interface of the CMainFrame class
//
/////////////////////////////////////////////////////////////////////////////

#if !defined(AFX_MAINFRM_H__D1AE6A7D_1702_11D4_9E15_006008AFEE67__INCLUDED_)
#define AFX_MAINFRM_H__D1AE6A7D_1702_11D4_9E15_006008AFEE67__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CMainFrame : public CFrameWnd
{

protected: // create from serialization only
	CMainFrame();
	DECLARE_DYNCREATE(CMainFrame)

// Attributes
public:
```

```
// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMainFrame)
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:  // control bar embedded members
        CStatusBar  m_wndStatusBar;
        CToolBar    m_wndToolBar;
        CToolBar          m_wndClrBar;

// Generated message map functions
protected:
        //{{AFX_MSG(CMainFrame)
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_MAINFRM_H__D1AE6A7D_1702_11D4_9E15_006008AFEE67__INCLUDED_)
#ifndef _MYDIALOG_H
#define _MYDIALOG_H

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MyDialog.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CMyDialog dialog

class CMyDialog : public CDialog
{
// Construction
public:
        CMyDialog();
```

```
        CMyDialog(LPCTSTR lpszTemplateName, CWnd* pParentWnd = NULL);
        CMyDialog(UINT nIDTemplate, CWnd* pParentWnd = NULL);

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMyDialog)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        virtual const DWORD* GetHelpIDs() = 0;
        // Generated message map functions
        //{{AFX_MSG(CMyDialog)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        afx_msg LONG OnHelp(UINT wParam, LONG lParam);
        afx_msg LONG OnHelpContextMenu(UINT wParam, LONG lParam);
        DECLARE_MESSAGE_MAP()
};

#endif
#if
!defined(AFX_MYSPLASHWND_H__6DBD2731_172C_11D4_9E16_006008AFEE67__INCLUDED_)
#define AFX_MYSPLASHWND_H__6DBD2731_172C_11D4_9E16_006008AFEE67__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// MySplashWnd.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CMySplashWnd dialog

class CMySplashWnd : public CDialog
{
// Construction
public:
        CMySplashWnd(CWnd* pParent = NULL);   // standard constructor
        BOOL Create(CWnd* pParent);

// Dialog Data
        //{{AFX_DATA(CMySplashWnd)
        enum { IDD = IDD_SPLASH };
                // NOTE: the ClassWizard will add data members here
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CMySplashWnd)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support
```

```
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CMySplashWnd)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_MYSPLASHWND_H__6DBD2731_172C_11D4_9E16_006008AFEE67__INCLUDED_)
// Project.h : main header file for the PROJECT application
//

#if !defined(AFX_PROJECT_H__D1AE6A79_1702_11D4_9E15_006008AFEE67__INCLUDED_)
#define AFX_PROJECT_H__D1AE6A79_1702_11D4_9E15_006008AFEE67__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
        #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"       // main symbols
#include "MySplashWnd.h"

/////////////////////////////////////////////////////////////////////////////
// CProjectApp:

#define PROJ_UINT_CHINESE 936
#define PROJ_UINT_ENGLISH 1252

class CProjectApp : public CWinApp
{
public:
        HINSTANCE m_hInstResDLL;
        CProjectApp();
        int localeACP;
private:
        DWORD m_dwSplash;
        CMySplashWnd m_splash;
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CProjectApp)
        public:
        virtual BOOL InitInstance();
        virtual BOOL OnIdle(LONG lCount);
        virtual int ExitInstance();
```

```
        //}}AFX_VIRTUAL

// Implementation
        //{{AFX_MSG(CProjectApp)
        afx_msg void OnAppAbout();
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_PROJECT_H__D1AE6A79_1702_11D4_9E15_006008AFEE67__INCLUDED_)
// ProjectDoc.h : interface of the CProjectDoc class
//

#if !defined(AFX_PROJECTDOC_H__D1AE6A7F_1702_11D4_9E15_006008AFEE67__INCLUDED_)
#define AFX_PROJECTDOC_H__D1AE6A7F_1702_11D4_9E15_006008AFEE67__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000


class CProjectDoc : public CDocument
{
protected: // create from serialization only
        CProjectDoc();
        DECLARE_DYNCREATE(CProjectDoc)

// Attributes
public:

// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CProjectDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        //}}AFX_VIRTUAL

// Implementation
public:
        virtual ~CProjectDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
```

```
protected:

// Generated message map functions
protected:
        //{{AFX_MSG(CProjectDoc)
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG

        DECLARE_MESSAGE_MAP()
};
//{{AFX_INSERT_LOCATION}}

#endif //
!defined(AFX_PROJECTDOC_H__D1AE6A7F_1702_11D4_9E15_006008AFEE67__INCLUDED_)
// ProjectView.h : interface of the CProjectView class
//

#if !defined(AFX_PROJECTVIEW_H__D1AE6A81_1702_11D4_9E15_006008AFEE67__INCLUDED_)
#define AFX_PROJECTVIEW_H__D1AE6A81_1702_11D4_9E15_006008AFEE67__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CGoLineDlg;

class CProjectView : public CEditView
{
private:
        BOOL m_bAddress;
        CPINFO CPInfo;
        BYTE *vbLBRange;
        BOOL vfDBCS;
public:
        CGoLineDlg* m_pGoLine;
        COLORREF      m_crColor;
protected: // create from serialization only
        CProjectView();
        DECLARE_DYNCREATE(CProjectView)

// Attributes
public:

        static void InitailizeView();
        //static void TerminateView();
        CProjectDoc* GetDocument();
        char* MyCharNext(char* pszStr);
// Operations
public:

// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CProjectView)
        public:
```

```
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual void OnInitialUpdate();
        protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);
        //}}AFX_VIRTUAL

// Implementation
public:

        virtual ~CProjectView();
        static int s_ColorMap[];
        static COLORREF GetColorRef(UINT nID);
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

        UINT m_uTimerID; // ==0 when no outstanding

        static LOGFONT NEAR m_lfDefFont;
        static LOGFONT NEAR m_lfDefFontOld;
        CFont m_font;

        static LOGFONT NEAR m_lfDefPrintFont;
        static LOGFONT NEAR m_lfDefPrintFontOld;
        CFont m_fontPrint;

        static UINT m_nDefTabStops;
        static UINT m_nDefTabStopsOld;
        static BOOL m_bDefWordWrap;
        static BOOL m_bDefWordWrapOld;

        UINT m_nPreviewPage;
        CTime m_timeHeader;
        CTime m_timeFooter;


        CString m_strAdr1;
        CString m_strAdr2;
        CString m_strState;
        CString m_strzip;
        CString m_strCty;

// Generated message map functions
protected:
        //{{AFX_MSG(CProjectView)
        afx_msg void OnInsertDateandtime();
        afx_msg void OnChooseFont();
        afx_msg void OnFormatTabstop();
        afx_msg void OnInsertCount();
        afx_msg void OnFormatAddress();
```

```
        afx_msg void OnInsertPasteaddress();
        afx_msg void OnEditGoto();
        afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
        afx_msg void OnFormatSetprinterfont();
        afx_msg void OnInsertCharcount();
        //}}AFX_MSG

        afx_msg void OnColor(UINT nID);
        afx_msg void OnUpdateColor(CCmdUI* pCmdUI);
        DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in ProjectView.cpp
inline CProjectDoc* CProjectView::GetDocument()
   { return (CProjectDoc*)m_pDocument; }
#endif

/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_PROJECTVIEW_H__D1AE6A81_1702_11D4_9E15_006008AFEE67__INCLUDED_)
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by Project.rc
//
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_PROJECTYPE                  129
#define IDS_PROJECT_TITLE               129
#define IDD_DATETIMEDLG                 130
#define IDS_COUNT_CHAR                  130
#define IDD_SET_TABSTOPS                131
#define IDS_STATE1                      131
#define IDD_SPLASH                      132
#define IDS_STATE2                      132
#define IDD_ADDRESS                     133
#define IDS_STATE3                      133
#define IDS_STATE4                      134
#define IDD_GOLINE_DIALOG               134
#define IDS_STATE5                      135
#define IDR_TEXT_POPUP                  135
#define IDS_STATE6                      136
#define IDS_PROVINCE1                   137
#define IDS_PROVINCE2                   138
#define IDR_MYCLR_TOOLBAR               138
#define IDS_PROVINCE3                   139
#define IDS_PROVINCE4                   140
#define IDS_PROVINCE5                   141
#define IDS_PROVINCE6                   142
#define IDS_AMERICA                     143
#define IDS_CHINA                       144
#define IDS_JAPAN                       145
#define IDS_KOREA                       146
```

```
#define IDS_CANADA                147
#define IDS_NO_ADDRESS            148
#define IDS_INVALID_LINE          149
#define IDS_BIG_LINENUM           150
#define IDC_DATETIME_LIST         1000
#define IDC_DATETIME_FORMAT       1001
#define IDC_STATIC_TAB            1002
#define IDC_EDIT_TAB              1003
#define IDC_STATIC_1              1004
#define IDC_STATIC_2              1005
#define IDC_STATIC_3              1006
#define IDC_STATIC_4              1007
#define IDC_STATIC_ADR1           1008
#define IDC_STATIC_ADR2           1009
#define IDC_EDIT_ADR1             1011
#define IDC_EDIT_ADR2             1012
#define IDC_COMBO_SATATE          1013
#define IDC_COMBO_COUNTRY         1014
#define IDC_STATIC_STATE          1015
#define IDC_STATIC_COUNTRY        1016
#define IDC_STATIC_ZIP            1017
#define IDC_EDIT_ZIP              1018
#define IDC_STATIC_GOLINE         1019
#define IDC_EDIT_GOLINE           1020
#define IDC_RADIO_INCH            1021
#define IDC_RADIO_CM              1023
#define IDC_MEASURE_GROUP         1024
#define IDC_RADIO_DEF             1025
#define ID_INSERT_DATEANDTIME     32771
#define ID_FORMAT_BULLETSTYLE     32773
#define ID_FORMAT_TABSTOP         32774
#define ID_CHOOSE_FONT            32775
#define ID_INSERT_COUNTCHARACTERS 32776
#define ID_FORMAT_ADDRESS         32777
#define ID_INSERT_PASTEADDRESS    32778
#define ID_EDIT_GOTO              32780
#define ID_COLOR0                 32781
#define ID_COLOR1                 32783
#define ID_COLOR2                 32784
#define ID_COLOR3                 32785
#define ID_COLOR4                 32786
#define ID_COLOR5                 32787
#define ID_COLOR6                 32788
#define ID_COLOR7                 32789
#define ID_COLOR8                 32790
#define ID_COLOR9                 32791
#define ID_COLOR10                32792
#define ID_COLOR11                32793
#define ID_COLOR12                32794
#define ID_COLOR13                32795
#define ID_COLOR14                32796
#define ID_COLOR15                32797
#define ID_FORMAT_SETPRINTERFONT  32814
#define ID_FORMAT_PAGESETUP       32815
#define ID_INSERT_CHARCOUNT       32816
#define IDP_ERR_GET_DEVICE_DEFAULTS 61446
```

```
#define IDP_ERR_GET_PRINTER_DC        61447

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_3D_CONTROLS                1
#define _APS_NEXT_RESOURCE_VALUE        140
#define _APS_NEXT_COMMAND_VALUE         32819
#define _APS_NEXT_CONTROL_VALUE         1026
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
#if !defined(AFX_SETTABSTOPS_H__12E9ED91_1714_11D4_9E15_006008AFEE67__INCLUDED_)
#define AFX_SETTABSTOPS_H__12E9ED91_1714_11D4_9E15_006008AFEE67__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// SetTabStops.h : header file
//

/////////////////////////////////////////////////////////////////////////////
// CSetTabStops dialog

class CSetTabStops : public CDialog
{
// Construction
public:
        CSetTabStops(CWnd* pParent = NULL);   // standard constructor

// Dialog Data
        //{{AFX_DATA(CSetTabStops)
        enum { IDD = IDD_SET_TABSTOPS };
        UINT    m_nTabStops;
        int             m_Unit;
        //}}AFX_DATA


// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CSetTabStops)
        protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support
        //}}AFX_VIRTUAL

// Implementation
protected:

        // Generated message map functions
        //{{AFX_MSG(CSetTabStops)
        virtual BOOL OnInitDialog();
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
};
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif //
!defined(AFX_SETTABSTOPS_H__12E9ED91_1714_11D4_9E15_006008AFEE67__INCLUDED_)
// ResourceDll.cpp : Defines the initialization routines for the DLL.
//

#include "stdafx.h"
#include <afxdllx.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif


static AFX_EXTENSION_MODULE ResourceDllDLL = { NULL, NULL };

extern "C" int APIENTRY
DllMain(HINSTANCE hInstance, DWORD dwReason, LPVOID lpReserved)
{
        // Remove this if you use lpReserved
        UNREFERENCED_PARAMETER(lpReserved);

        if (dwReason == DLL_PROCESS_ATTACH)
        {
                TRACE0("RESOURCEDLL.DLL Initializing!\n");

                // Extension DLL one-time initialization
                if (!AfxInitExtensionModule(ResourceDllDLL, hInstance))
                        return 0;
                new CDynLinkLibrary(ResourceDllDLL);
        }
        else if (dwReason == DLL_PROCESS_DETACH)
        {
                TRACE0("RESOURCEDLL.DLL Terminating!\n");
                // Terminate the library before destructors are called
                AfxTermExtensionModule(ResourceDllDLL);
        }
        return 1;  // ok
}
```

APPENDIX C Code for Java Application

```
import java.awt.*;
import java.awt.event.*;
import java.beans.*;
import java.io.*;
import java.net.URL;
import java.util.*;

import javax.swing.text.*;
import javax.swing.undo.*;
import javax.swing.event.*;
import javax.swing.*;

class Jthes extends JPanel {

   private static ResourceBundle resources;

   static {
      try {
                  resources = ResourceBundle.getBundle("resources.Jthes",
                              Locale.getDefault());
System.out.println(Locale.getDefault());
            } catch (MissingResourceException mre) {
         System.err.println("resources/Jthes.properties not found");
         System.exit(1);
      }
   }

   Jthes() {
  super(true);

try {
    UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName())
} catch (Exception exc) {
   System.err.println("Error loading L&F: " + exc);
 }

 setBorder(BorderFactory.createEtchedBorder());
 setLayout(new BorderLayout());
```

```
editor = createEditor();
editor.setFont(new Font("monospaced", Font.PLAIN, 12));
editor.getDocument().addUndoableEditListener(undoHandler);

commands = new Hashtable();
Action[] actions = getActions();
for (int i = 0; i < actions.length; i++) {
Action a = actions[i];
commands.put(a.getValue(Action.NAME), a);
 }

 JScrollPane scroller = new JScrollPane();
 JViewport port = scroller.getViewport();
 port.add(editor);
 try {
    String vpFlag = resources.getString("ViewportBackingStore");
    Boolean bs = new Boolean(vpFlag);
    port.setBackingStoreEnabled(bs.booleanValue());
 } catch (MissingResourceException mre) {
}

 menuItems = new Hashtable();
 menubar = createMenubar();
 add("North", menubar);
 JPanel panel = new JPanel();
 panel.setLayout(new BorderLayout());
panel.add("Center", scroller);
add("Center", panel);
add("South", createStatusbar());
add("South", createStatusbar());
  }

  public static void main(String[] args) {
     try {
     String vers = System.getProperty("java.version");
     if (vers.compareTo("1.1.2") < 0) {
        System.out.println("!!!WARNING: Swing must be run with a " +
                    "1.1.2 or higher version VM!!!");
     }
     JFrame frame = new JFrame();
     frame.setTitle(resources.getString("Title"));
frame.setBackground(Color.lightGray);
frame.getContentPane().setLayout(new BorderLayout());
frame.getContentPane().add("Center", new Jthes());
frame.addWindowListener(new AppCloser());
frame.pack();
frame.setSize(500, 600);
     frame.show();
     } catch (Throwable t) {
        System.out.println("uncaught exception: " + t);
        t.printStackTrace();
     }
 }
public Action[] getActions() {
return TextAction.augmentList(editor.getActions(), defaultActions);
 }
```

```
protected JTextComponent createEditor() {
eturn new JTextArea();
 }

protected JTextComponent getEditor() {
 return editor;
   }

protected static final class AppCloser extends WindowAdapter {
      public void windowClosing(WindowEvent e) {
    System.exit(0);
 }
   }

protected Frame getFrame() {
 for (Container p = getParent(); p != null; p = p.getParent()) {
    if (p instanceof Frame) {
   if (p instanceof Frame) {
 return (Frame) p;
    }
}
return null;
  }

protected JMenuItem createMenuItem(String cmd) {
JMenuItem mi = new JMenuItem(getResourceString(cmd + labelSuffix));
     URL url = getResource(cmd + imageSuffix);
if (url != null) {
   mi.setHorizontalTextPosition(JButton.RIGHT);
  // mi.setIcon(new ImageIcon(url));
}
String astr = getResourceString(cmd + actionSuffix);
if (astr == null) {
    astr = cmd;
 }
 mi.setActionCommand(astr);
 Action a = getAction(astr);
 if (a != null) {
    mi.addActionListener(a);
    a.addPropertyChangeListener(createActionChangeListener(mi));
    mi.setEnabled(a.isEnabled());
 } else {
    mi.setEnabled(false);
 }
 menuItems.put(cmd, mi);
 return mi;
   }

protected JMenuItem getMenuItem(String cmd) {
protected JMenuItem getMenuItem(String cmd) {
return (JMenuItem) menuItems.get(cmd);
  }

 protected Action getAction(String cmd) {
return (Action) commands.get(cmd);
  }
```

```
 protected String getResourceString(String nm) {
String str;
try {
   str = resources.getString(nm);
} catch (MissingResourceException mre) {
   str = null;
}
return str;
 }

 protected URL getResource(String key) {
String name = getResourceString(key);
if (name != null) {
   URL url = this.getClass().getResource(name);
   return url;
return null;
 }

 protected Container getToolbar() {
return toolbar;
 }

 protected JMenuBar getMenubar() {
return menubar;
 }

protected JButton createToolbarButton(String key) {
URL url = getResource(key + imageSuffix);
    JButton b = new JButton(new ImageIcon(url)) {
      public float getAlignmentY() { return 0.5f; }
};
    b.setRequestFocusEnabled(false);
    b.setMargin(new Insets(1,1,1,1));

String astr = getResourceString(key + actionSuffix);
if (astr == null) {
   astr = key;
}
Action a = getAction(astr);
if (a != null) {
    b.setActionCommand(astr);
    b.addActionListener(a);
 } else {
    b.setEnabled(false);
 }

 String tip = getResourceString(key + tipSuffix);
 if (tip != null) {
    b.setToolTipText(tip);
 }

     return b;
  }
protected String[] tokenize(String input) {
 Vector v = new Vector();
```

```
 StringTokenizer t = new StringTokenizer(input);
 String cmd[];

 while (t.hasMoreTokens())
    v.addElement(t.nextToken());
 cmd = new String[v.size()];
 for (int i = 0; i < cmd.length; i++)
    cmd[i] = (String) v.elementAt(i);

 return cmd;
   }
protected JMenuBar createMenubar() {
 JMenuItem mi;
 JMenuBar mb = new JMenuBar();

 String[] menuKeys = tokenize(getResourceString("menubar"));
 for (int i = 0; i < menuKeys.length; i++) {
    JMenu m = createMenu(menuKeys[i]);
   JMenu m = createMenu(menuKeys[i]);
   if (m != null) {
  mb.add(m);
   }
}
return mb;
  }

protected JMenu createMenu(String key) {
String[] itemKeys = tokenize(getResourceString(key));
JMenu menu = new JMenu(getResourceString(key + "Label"));
for (int i = 0; i < itemKeys.length; i++) {
   if (itemKeys[i].equals("-")) {
  menu.addSeparator();
   } else {
  JMenuItem mi = createMenuItem(itemKeys[i]);
  menu.add(mi);
   }
}
 return menu;
  }

protected PropertyChangeListener createActionChangeListener(JMenuItem b)
 return new ActionChangedListener(b);
  }

private class ActionChangedListener implements PropertyChangeListener {
     JMenuItem menuItem;

     ActionChangedListener(JMenuItem mi) {
        super();
        this.menuItem = mi;
     }
     public void propertyChange(PropertyChangeEvent e) {
        String propertyName = e.getPropertyName();
        if (e.getPropertyName().equals(Action.NAME)) {
           String text = (String) e.getNewValue();
           menuItem.setText(text);
```

```
        } else if (propertyName.equals("enabled")) {
            Boolean enabledState = (Boolean) e.getNewValue();
            menuItem.setEnabled(enabledState.booleanValue());
        }
    }
}

private JTextComponent editor;
private Hashtable commands;
private Hashtable menuItems;
private JMenuBar menubar;
private JToolBar toolbar;
private JComponent status;
private JFrame elementTreeFrame;
protected ElementTreePanel elementTreePanel;

protected FileDialog fileDialog;
protected UndoableEditListener undoHandler = new UndoHandler();
protected UndoManager undo = new UndoManager();
public static final String imageSuffix = "Image";
public static final String labelSuffix = "Label";

public static final String actionSuffix = "Action";

public static final String tipSuffix = "Tooltip";
public static final String openAction = "open";
public static final String newAction  = "new";
public static final String saveAction = "save";
public static final String exitAction = "exit";
public static final String showElementTreeAction = "showElementTree";

class UndoHandler implements UndoableEditListener {

public void undoableEditHappened(UndoableEditEvent e) {
  undo.addEdit(e.getEdit());
  undoAction.update();
  redoAction.update();
}
 }

class StatusBar extends JComponent {

    public StatusBar() {
  super();
  setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
}

    public void paint(Graphics g) {
  super.paint(g);
}

 }

private UndoAction undoAction = new UndoAction();
private RedoAction redoAction = new RedoAction();
```

```
 private Action[] defaultActions = {
 new NewAction(),
 new OpenAction(),
 new ExitAction(),
 new ShowElementTreeAction(),
    undoAction,
   redoAction
 };

 class UndoAction extends AbstractAction {
public UndoAction() {
  super("Undo");
  setEnabled(false);
}

public void actionPerformed(ActionEvent e) {
  try {
 undo.undo();
  } catch (CannotUndoException ex) {
 System.out.println("Unable to undo: " + ex);
 ex.printStackTrace();
  }
  }
  update();
  redoAction.update();
}

protected void update() {
  if(undo.canUndo()) {
 setEnabled(true);
 putValue(Action.NAME, undo.getUndoPresentationName());
  }
  else {
 setEnabled(false);
 putValue(Action.NAME, "Undo");
  }
}
 }

 class RedoAction extends AbstractAction {
public RedoAction() {
  super("Redo");
  setEnabled(false);
}


public void actionPerformed(ActionEvent e) {
  try {
 undo.redo();
  } catch (CannotRedoException ex) {
 System.out.println("Unable to redo: " + ex);
 ex.printStackTrace();
  }
  update();
  undoAction.update();
}
```

```
protected void update() {
   if(undo.canRedo()) {
  setEnabled(true);
  putValue(Action.NAME, undo.getRedoPresentationName());
    }
    else {
  setEnabled(false);
  putValue(Action.NAME, "Redo");
    }
}
  }

 class OpenAction extends NewAction {

OpenAction() {
   super(openAction);
}

     public void actionPerformed(ActionEvent e) {
   Frame frame = getFrame();
   if (fileDialog == null) {
 fileDialog = new FileDialog(frame);
    }
   fileDialog.setMode(FileDialog.LOAD);
   fileDialog.show();

   String file = fileDialog.getFile();
   if (file == null) {
 return;
    }
   String directory = fileDialog.getDirectory();
   File f = new File(directory, file);
   if (f.exists()) {
 Document oldDoc = getEditor().getDocument();
 if(oldDoc != null)
    oldDoc.removeUndoableEditListener(undoHandler);
 if (elementTreePanel != null) {
    elementTreePanel.setEditor(null);
 }
 getEditor().setDocument(new PlainDocument());
 frame.setTitle(file);
 Thread loader = new FileLoader(f, editor.getDocument
 loader.start();
    }
}
  }

 class NewAction extends AbstractAction {

NewAction() {
   super(newAction);
}

NewAction(String nm) {
   super(nm);
```

```
}


    public void actionPerformed(ActionEvent e) {
   Document oldDoc = getEditor().getDocument();
   if(oldDoc != null)
   oldDoc.removeUndoableEditListener(undoHandler);
   getEditor().setDocument(new PlainDocument());
   getEditor().getDocument().addUndoableEditListener(undoHandler);
   revalidate();
 }
  }
class ExitAction extends AbstractAction {

 ExitAction() {
   super(exitAction);
 }

    public void actionPerformed(ActionEvent e) {
   System.exit(0);
 }
class ShowElementTreeAction extends AbstractAction {

 ShowElementTreeAction() {
   super(showElementTreeAction);
 }

 ShowElementTreeAction(String nm) {
   super(nm);
 }

  public void actionPerformed(ActionEvent e) {
  if(elementTreeFrame == null) {
  // Create a frame containing an instance of
  // ElementTreePanel.
  try {
     String   title = resources.getString
              ("ElementTreeFrameTitle");
     elementTreeFrame = new JFrame(title);
  } catch (MissingResourceException mre) {
     elementTreeFrame = new JFrame();
  }

  elementTreeFrame.addWindowListener(new WindowAdapter() {
     public void windowClosing(WindowEvent weeee) {
   elementTreeFrame.setVisible(false);
     }
  });
  Container fContentPane = elementTreeFrame.getContentPane();

  fContentPane.setLayout(new BorderLayout());
  elementTreePanel = new ElementTreePanel(getEditor());
  fContentPane.add(elementTreePanel);
  elementTreeFrame.pack();
   }
   elementTreeFrame.show();
```

```
}
}
```

VITA

Hangbo Zhang

Candidate for the Degree of

Master of Science

Thesis: INTERNATIONALIZATION AND LOCALIZATION FOR CHINESE SOFTWARE

Major Field: Computer Science

Biographical:

Personal Data: Born in Hunan, China on October 16, 1970, the second son of Binwen Zhang and Shiying Peng

Education: Graduated from the Chemical Engineering Department of Hunan University (China) in July, 1991, and received the Bachelor's degree of Chemical Engineering. Completed the requirements of the Master of Science at Oklahoma State University in July 2000.

Professional Experience: Employed by KaiDi Chemical (HK) Ltd. Shenzhen, China, as a Chemical Engineer, 1991 to 1997; employed by Sterling Software San Rafael, CA, as a Software Co-Engineer, July 1999 to December 1999; employed by Oklahoma State University, Department of Computer Science, as a Research Assistant, March 1999 to July 1999, and January 2000 to May 2000.