

Université Catholique de Louvain

Faculté des Sciences Appliquées

Département d'Ingénierie Informatique



Reverse engineering of Graphical User Interfaces based on Resource Files

Promoteur :

Pr. J. Vanderdonckt

*Mémoire présenté en vue
de l'obtention du grade de*

licencié en informatique

par Julien Marion

Louvain-la-Neuve
Année académique 2005-2006

Parmi les personnes qui m'ont permis de réaliser ce mémoire, je tiens tout particulièrement à remercier M. Jean Vanderdonckt et M. Laurent Bouillon pour leurs conseils, leur disponibilité et leur gentillesse d'avoir accepté de m'encadrer dans ce travail. Je tiens à remercier ma famille, et en particulier mes parents pour leur soutien tout au long de mes études.

Contents

1	Introduction	5
2	State of the art	7
3	Reverse engineering method	9
3.1	General method	9
3.2	Methodological choices.....	13
3.3	Examples used and decompilation tools	16
3.4	Selected method	17
4	Windows resource files	19
4.1	Examples of resources.....	19
4.1.1	First example from SciTE	19
4.1.2	Second example from 7-Zip File Manager	20
4.1.3	Third example from RGB Editor	21
4.1.4	Fourth example from WinDirStat	22
4.1.5	Fifth example from the Common Dialog Library	23
4.1.6	Sixth example from TablEdit	24
4.2	Resource files modelization	26
4.2.1	Menus class diagram	26
4.2.2	Dialog boxes class diagram.....	26
4.2.3	Documentation	28
4.2.4	Constraints.....	33
4.3	Resource files structure	35
4.3.1	Resources of type dialog box	35
4.3.1.1	Dialog box template	35
4.3.1.2	Controls definition.....	39
4.3.2	Shortcut notations in the dialog box template.....	45
4.3.3	Resource of type menu	46
4.3.4	Other types of resources	47
4.4	About the decompilation tools	48
5	UsiXML	50
5.1	Structure of UsiXML	50
5.2	Concrete User Interface model.....	55
6	Importing resource files in GrafiXML	59
6.1	Transformation of resources into CUI	59
6.1.1	Correspondences table.....	59
6.1.1.1	Resources of type dialog box	60
6.1.1.2	Resources of type menu	66
6.1.2	Derivation rules	66

6.2	Plug-in development	70
6.2.1	How to run it?.....	70
6.2.2	How to use it?.....	71
6.2.3	Specifications	72
6.2.4	Description of my implementation.....	73
7	Conclusion	83
	Bibliography	85
	Appendix A <i>Example of complete resource file</i>	
	Appendix B <i>Comparison of the resource files given by the decompilers</i>	
	Appendix C <i>Documentation of UML diagrams modeling a resource file</i>	
	Appendix D <i>Source code</i>	

1 Introduction

Software development as it's commonly practiced today doesn't look much like engineering, but it should. Methods, models and automated tools composing software engineering practices are required to deliver software of high quality¹, and with low costs. The major source of software problems is poor requirements specification. The extraction of these requirements is the hardest function of software engineering, as the most important one since the cost of late corrections is prohibitive. The user interface in particular rarely satisfies the real expectations at the first time, it's from the look and the manipulation of the final product that we see really what is wrong. The requirements are also volatile. A feedback of users during the development is needful to provide a pleasant interface, statically (the presentation²) as dynamically (the dialog³). Traditional process models to lay out the different activities in the development life cycle are also not always appropriated, especially for the conception of user interfaces. For example, the waterfall model is a simple sequential composition of the activities (requirements, specifications, design, implementation, integration, maintenance). This model is too general and unrealistic to design user interfaces particularly because it not address early validation. To reduce the risk of inappropriate user interface, better models still make it possible the creation of interface prototypes and then more interactions with the users during the development. For example, exploratory models allow successive revisions of the software specifications based on prototypes quickly implemented from which we can obtain users feedback (when the expectations are reached, we then apply the other traditional software engineering phases). Another example is the spiral model allowing an iterative and incremental development (certainly the stronger process model to minimize risks).

The life duration of a product is an important aspect. We wish that the code we write for big applications will run for a long time, not that it becomes out-dated after few years. It's also needful to be able to produce a vast range of products (that is, multiple versions), and not only one product. Software's are subject to continual changes in their life cycle, and organizations devote significant resources to their maintenance and evolution. [The problem is to quickly adapt the user interface of interactive applications to these changes.]As the technologies evolve (mobile devices, e-commerce...), the requirements evolve too (additional versions, interface migration to the web ...). The today constant evolution of the variety of computing platform (a specific software and hardware environment) requires more efforts to cope with software portability⁴.

¹ The usability of a product is a factor of quality.

² For example, the service has to be easily visible and the critical services easily accessible.

³ For example, a good scheduling of the actions to be performed has to be defined.

⁴ The portability of a product is another factor of quality.

Graphical user interfaces are important parts of today's interactive applications. If we examine the code of interactive applications, a large part is generally devoted to the user interface portion. The development of user interface can then be strong component in the global cost engendered (dominated by the maintenance). It is then normal that software engineering also include well established methods to develop the user interface, uncoupling it from the application. The various aspects to take into consideration (such as the interactive task to perform, the domain, the context of use⁵, the presentation of information and the dialog between the user and the system) are generally captured by means of models. These specific models explicitly capture knowledge about the user interface with appropriate abstraction. The importance of having user interface engineering models justifies the need of providing appropriate automated tools to create each of these models. The design can be rapidly prototyped and implemented (possibly even before the application code is written) and it's easier to incorporate changes discovered through user testing. The motivation is also to facilitate the creation of user interfaces that work on any platform available today.

The goal of this thesis is to develop a tool aimed to capture the essence of an application's graphical user interface and recreate a model at a more abstract level (specifying the user interface without the multiple details linked to the implementation in a specific language). It will serve principally as automated support for reengineering activities such as the redesign of existing interfaces for another platform.

A method which performs abstraction is reverse engineering. The results can be then reused in a forward engineering phase to regenerate user interface code, achieving a reengineering cycle. The originality is that the model will not be extracted by the examination of the entire source code, but from the resource file of an application storing user interface information.

My approach to reuse an existing application's graphical user interface especially for different platforms can be stated as *the specification of this interface starting from the resource file of the application and describing it and at a higher level of abstraction, allowing a subsequent reengineering process adapted to any platform.*

⁵ The platform, the user and its environment define together the context of use.

2 State of the art

I will mention here some related works to the reverse engineering of application's user interfaces to show that this method is commonly used today. Many organizations for example are choosing to reengineer their critical applications to better fit their needs and to take advantage of the new technologies. Most of these researches lead to reverse engineering of legacy systems⁶ user interfaces.

MORPH [1] is a process for reengineering the user interfaces of text-based legacy systems to graphical user interfaces. The resulting model is used to transform the abstractions in the model to a specific graphical widget toolkit. The process is composed of three steps. In the detection step, the source code is analyzed to identify user interaction components in the legacy system thanks to a detection engine. In the representation step, an abstract model is build expressing the existing user interface (as derived from the detection step). The model is then stored in the knowledge base. In the transformation step, human analyst can refine the model thanks to the transformation engine which makes it possible to manipulate, augment and restructure of the resulting model to a graphical environment. The transformation stage suggests specific graphical implementations and integrates them for user interface abstractions into the legacy code.

The **AIUDL** [2] environment was a pioneer approach in user interface (UI) reengineering. The original UI is first translated in AUIDL (Abstract UI Description Language). This language is able to represent UI objects in terms of both structure and behaviour. Different levels of abstractions can be defined. To translate the interface in AUIDL, user actions and system responses have to be identified using pattern matching techniques⁷. The Milner's process algebra is used to map out the behaviour of the system. The spatial organization of display objects is explicitly described with two mechanisms: containment and importation of attributes from other objects⁸. An abstract syntax tree is first obtained from the source code of the system. A module extracts UI fragments from it, and the abstract specification of the UI is then constructed. This last step is semi-automated: a part of the code is parsed and automatically abstracted and the remaining is left to the programmer. Finally, the AUIDL specifications are translated in the EASEL language (this language allows the automated

⁶ A legacy system is a computer system or application program which has been in place for a long time and which continues to be used because an organization does not want to support the costs of replace or redesign it.

⁷ The code of a UI is parsed to build a manipulable representation of it. Interface fragments are then extracted from this representation, and a pattern matcher identifies syntactic patterns in the fragments. Using the code fragments as a basis, details about modes of interaction and conditions of activation are identified with control flow analysis.

⁸ For example, an object imports the x-coordinate of another object to describe an alignment for these objects.

generation of screens for the IBM 3270 environment) and the generated code is linked with application core.

Since there is an increased demand to make legacy systems accessible through the web in order to support e-commerce activities, the problem of these systems' user interface migration to web-accessible platforms is becoming important. Instead of analysing the code to extract a model of its structure, the **Cellest** [3] process analyse traces of the system-user interaction to model the behaviour of the user interface. This produces a state-transition model which specifies the screens (as state) and the possible commands leading from one screen to another (as transitions between the states). To capture user interaction during the use of the legacy system, Cellest is composed of a recorder and a pilot allowing. The recorder localizes aspects of the legacy system relevant to a specific task. In this approach, an interface is viewed as a collection of screens, each allowing a set of transition actions to other screens. The screens are captured by the recorder that records each session (visited screens, actions in order to achieve specific tasks). The pilot translates the transition actions in functionally equivalent actions in a new graphical user interface. The code is not needed in this approach and the necessary information is extracted to complete the tasks on the old system. The model produced is used to construct models of the tasks performed in the system. These models are then used to develop a new web-accessible interface for executing these tasks (in a forward engineering phase).

The reverse engineering step is not only used for the reengineering of a UI. For example, I could still mention a process named **GUI Ripping** [4] which creates a model from a graphical UI for testing. To detect defects in UIs, test cases can then be automatically generated.

3 Reverse engineering method

I present in this chapter the reverse engineering method that will be used to obtain the specifications of an existing graphical user interface.

3.1 General method

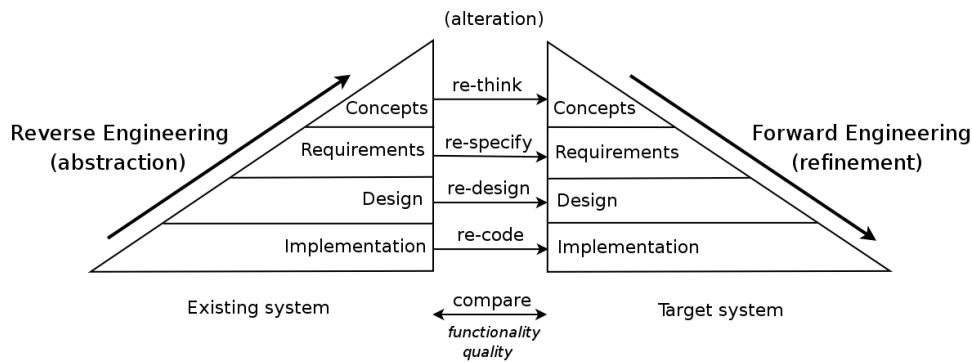
Reverse engineering is the general process of analyzing a technology to see how it works. Originally, it's a practice taken from industries that is now frequently used on computer software. In the automobile industry, for example, a manufacturer may purchase a competitor's vehicle, disassemble it, and examine the components to enhance their vehicles with similar components. Reverse engineering as a method can also be used in the development of software, including the user interface (UI) part giving access to its core functionalities. Such method does not involve changing the existing source UI or creating a new UI based on the reverse engineered source UI. It is a process of examination, not change or replication.

To give a simple definition⁹, reverse engineering is *"the process of analysing an existing system to identify its components and their interrelationships and create representations of the system in another form or at a higher level of abstraction. Reverse engineering is usually undertaken in order to redesign the system for better maintainability or to produce a copy of a system without access to the design from which it was originally produced."*

In black box reverse engineering, systems are observed without examining internal structure. For example, one might take the executable code of a program, run it to study how it behaved with different input and then attempt to write a program which behaved identically (or better). In white box reverse engineering the inner workings of the system are inspected.

We could think that reverse engineer a program is reversing a program's machine code back into the source code that it was written in. But it's a familiar use of the term. Here is a diagram [5] of the most traditional form of reverse engineering, which is an in-depth analysis (usually starting with source code) performing abstraction:

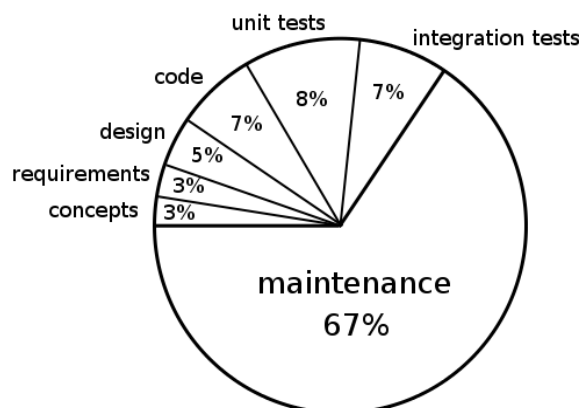
⁹ From the free on-line dictionary of computing <http://foldoc.doc.ic.ac.uk/foldoc/>.



The most traditional method of development is referred to as forward engineering. In the construction of UI, designers develop a product by implementing engineering concepts and abstractions. By contrast, reverse engineering begins with final product, and works backward to recreate the engineering concepts.

Due to the rapid changes and evolution of organisations and their business activities, the challenge is to quickly adapt their UIs of their interactive applications to these changes. Rather than create a new system, reengineering can be used to redesign the existing system. Note that reengineering involves reverse engineering followed by forward engineering. It is not a super type of the two.

The goal of maintenance in software engineering is to obtain a new version which responds some modification requests. There are three types of modification: a *correction* of detected faults, an *improvement* to have a better quality (performance, usability...) and an *adaptation* due to different, new or restricted requirements (called a variation, an extension and a contraction respectively).¹⁰ Reengineering is important since maintenance takes a great place in software production. Actually, the distribution of efforts (and then of costs) is often inadequate, and as consequence the cost to manage multiple versions for example is exorbitant. According to observations [6], the average distribution of the total cost of production (including development and maintenance costs) of a typical project can be represented by this pie-chart:



¹⁰ The new version is a revision when the modification is a correction or an improvement, and a variant when the modification is an adaptation.

The UI is rarely perfectly designed at the first shot, and the UI code has often to be rewritten completely when the software needs to be transformed for another platform. A substantial part of the code written for interactive applications is devoted to the user interface. This code has to be identified and replaced. UI design is then time-consuming and expensive in the development life cycle. Techniques for reengineering are important since too few time and effort are usually put in the specification and the conception of UI reliable, portable and maintainable. Moreover, the source code of the software may not always be available. When purchasing an application, the executable is often supplied to an organisation without any source code. Or the source code can also be lost. If we want to get a version with an interface in a more adapted language, correct errors and limitations in the existing interface or still transform or redesign an obsolete product into a useful one by adapting them to a new platform, the only way to present the information in a language that a programmer can understand is to have recourse to the software decompilation (translating binary code into source code). But the interpretation is a tedious task since original comments and specifications are not present in the resulting code.

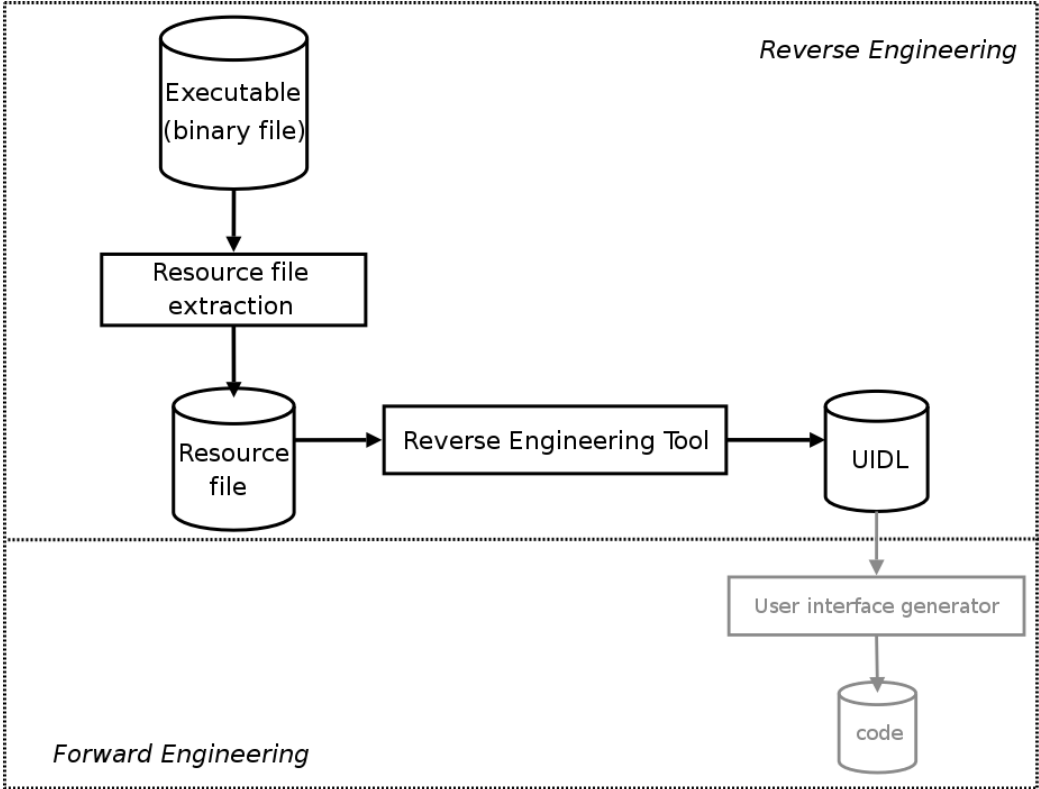
The general purpose of reverse engineer a UI, which is the subject of my thesis, is to examine an existing application's UI in order to extract an abstract representation. The method will be able to provide an automated way to extract relevant information (and in case of poor or non-existent documentation of the code, this also economizes efforts), to generate an additional view more synthetic of the UI (that will aid its review or modification) and to facility reuse. This contributes to the purpose of reengineer a UI, which is the transformation of the UI for another platform.

I describe in this thesis an approach to reverse engineer the UI of applications designed for a desktop machine directly from the executable file. A subsequent forward engineering process can be triggered. Instead of looking in the source code (composed of the functional core and the UI expressed in some programming language) for the part related to the UI, it's sometime possible to explore resources used and extract them from the executable file.

In addition to source code, most applications include resources used at runtime. Resources can be composed of a wide range of elements, including elements that are necessary for the UI but are not part of the application code itself and custom resources that contain data the application needs. Resources are additional binary data stored in an executable file. They do not reside in the executable program's data area (that is, the resources are not immediately addressable by variables in the program's data segment), but are appended to the executable in a separate area. When a program is loaded into memory for execution, it usually leaves resources on disk. A particular resource is loaded into memory when an application needs to use it for the first time. Just as multiple instances of the same program share the same code, multiple instances also usually share resources. Without the concept of resources, a binary file such as an icon for example would probably have to reside in a separate file that the executable would read into memory to use. Or the icon would have to be defined in the

program as an array of bytes (which make it difficult to visualize the actual icon image). As a resource, the icon is stored in a separate file and is bound into the executable file during the build process. All the resources are usually defined in a main file which is added, once compiled, to the application's executable file.

The following illustrates the general method that I'll used to reverse engineer the UI. The process starts from an executable file (a binary file containing a program in machine language which is ready to be executed).



The input of the tool to develop is a text-based resource file containing user interface information extracted from an executable.

The extraction can be made by some decompilation tool (that is, a program that converts the resource definition from machine language to some text programming language). Decompilation can be reverse engineering, since it is increasing the level of abstraction. It's just that it starts lower and ends lower (with the implementation, i.e. the source code) than most reverse engineering. Then decompilation would be part of my reverse engineering (although compilation is not considered part of forward engineering, since it is an automatic step).

The given UI has to be described at a level of abstraction that is higher than the level where code is manipulated. The output will be the UI expressed in a User Interface Description Language (UIDL). A specification language has to satisfy some requirements to be qualified

as UIDL. The UIDL should be precise to enable automatic specifications processing. It should be expressive enough to support software engineering techniques (such as model derivation). It should be compact, expressed in a standard format and as much as possible human-readable to allow designers and developers to exchange their specifications.

This transformation completes the reverse engineering part. The UIDL file then serves as input for forward engineering to generate UI code for a target computing platform.

3.2 Methodological choices

This section presents the choices that I've made to instantiate the general method illustrated in the previous section.

The reverse engineering tool presented in this thesis is limited to operate on UI of MS Windows' applications. Its **input** is any Windows resource script file (*.rc), from which high-level requirements will be extracted.

I've chosen to focus my investigation on Windows UIs since this operating system has broken into the market. But the scope of the analysed UIs could be enlarged by considering other formats of resource files. Resources such as nib (*.nib) or images (*.icns) resource files can also be packaged with Mac OS-based applications. Nib resource files store UI information, including windows, dialogs, and UI elements such as buttons, sliders, text objects, and help tags for these elements.¹¹ They are a bundle, which means they are really a directory structure, not a single file. I can mention also the screen definition files which are text files describing the layout of the display.¹² The method proposed could be generalized to these types of files.

A resource file associate to an executable MS Windows programs can include various types of resources, such as principally accelerator table, bitmap, cursor, menu, dialog, icon, string table and version information. An accelerator table is used to define key combinations that generate a command message (often to duplicate the action of common menu options). Some programs use customized mouse cursors to represent different operations of the program. Programs usually use menus, dialog boxes and customized icons (such as the icon displayed in the upper left corner of the title bar of the application window or shown as a shortcut on the desktop). The data in a resource can also describe character strings not defined as variables in the source code. The text used in a UI is isolated in one file to make easier the translation of a program in other languages and to reduce memory space. In addition to the standard

¹¹ An application can use several nib files, with one of them designated as the main nib file (containing the main menu and any windows appearing when the application starts up). Other nib files can be loaded whenever needed. A good rule for creating nib files is to use one nib for each separate kind of window in the application.

¹² They learn each window (screen) in an application and each object within that window.

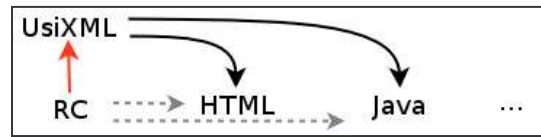
resources, a user-defined resource (also called a custom resource) can be used to attach any data in a convenient format required by a specific application.

These resource are defined in a resource script file (*.rc). This file is an ASCII text file. It contains representations of the resources that can be expressed in text, such as menus and dialog boxes. It also contains references to binary files that contain non text resources, such as icons and customized mouse cursors. All resources in a resource script file are defined using identifiers that, together with the type of the resource being referenced, uniquely identify each resource in an application. Once compiled, the resources are binary data. The binary compiled resource file (*.res) is then added to the executable file.

The resources appended to an executable can be extracted thanks to existing freeware utilities. I propose in the next section those that I've used. Thanks to these tools, image resources (such as cursor, icon, bitmap, gif, avi, and jpg files) can be viewed, audio resources (such as wav and midi files) can be played and the other type of resources (such as accelerator table, menu, dialog, string table and version information) can be viewed as decompiled resource scripts. Menus and dialog boxes can also be viewed as they would appear in a running application. These programs are usually used to edit the resources (modify, add and delete resources). The resources can also be saved as image files (*.ico, *.bmp etc.), as binary resource files (*.res) or as resource script files (*.rc). This last functionality giving a text representation of the UI is integrated in my reverse engineering process. An example of resource script file generated by one of these tools is given in appendix A.

The reverse engineering tool will provide as **output** a UsiXML (User Interface eXtensible Markup Language) description of the given resource script file. This language gives an XML UI description. It has been chosen to express the abstract representation of the UI because it's a UIDL that can be used to specify a platform-independent UI and enabling multi-path UI development. UsiXML can express various models depending on the level of abstraction. In my case, the UI will be reverse engineered at a level of abstraction dependent of the modality of interaction since it's a graphical UI. The model produced at this level is the Concrete User Interface (CUI) model. It can represent in a sufficient degree of expressiveness the elements perceived in a UI code in any specific language. A set of forward engineering tools using this language are also today available. The UsiXML language and its level of details and abstractions are introduced in chapter 5. XML is frequently used to describe and design UI for any device, any target language and any operating system on the device. Another well-known language is UIML, which is a relatively simple markup language (a little over two dozen tags). Various other related languages exist today, such as XUL (Xml-based User interface Language), XIIML (eXtensible Interface Markup Language), AUIML (Abstract User Interface Markup Language), useML (Useware Markup Language) or XIICL (eXtensible user Interface and user interface Components Language).

To accomplish the transformation of a UI specification from one format to another, I've decided to implement a new plug-in into the **GrafiXML** tool.¹³ In order to facilitate the edition of a new UI in UsiXML (at the CUI level), GrafiXML has been developed as a graphical tool to draw UIs. It's an editor based on a classical element-based approach (that is, all the elements are directly described by a physical form we can identify and assign a meaning). The user



can draw in direct manipulation a UI by placing elements and editing their properties in the composer, which are instantly reflected in the XML editor.¹⁴ At any time, the user can see the corresponding UsiXML specifications and edit it (and if a tag or the elements are modified, the changes are propagated to the graphical representation). The user can then save the UI in the UsiXML format. I will use this tool in order to reverse engineer a resource script file and obtain a CUI model. By importing a resource script file into GrafiXML, the layout of the specified UI will have to be displayed in the composer and the UsiXML specifications will have to be generated in the XML editor. The UsiXML file can then for example serve as input in forward engineering to generate UI code designed for another platform.

Using GrafiXML in my method has several useful advantages. Firstly, this will considerably facilitate my implementation: I can use built-in methods to generate and manage elements at the desired level of abstraction after having defined transformations rules. Then, the designer can view the result in a graphical form when reverse engineering a resource script file, and not only a XML specification. This gives some confidence in the reliability of the tool without having to implement some code giving the rendering. This graphical representation will also help me to validate my tool in the testing phase. The actual result can be rapidly compared with the expected one and the produced model can be easily analysed. Finally, GrafiXML is able to automatically generate UI code in HTML, XHTML (Extensible HyperText Markup Language¹⁵), XUL and Java thanks to a series of plug-ins. Reengineering of a resource script file can then be automated to obtain the code in these high-level languages.

The development of a new plug-in enhances also the functionalities of GrafiXML. Existing UIs can be reused and incorporated in a project current edited in the tool. We can see the utility of the plug-in that I will develop from two different points of view. As usual, GrafiXML can be used as an editor to specify UIs in UsiXML (at the CUI level) where Windows UIs can be imported. Or it can be used as a reverse engineering tool.

¹³ This free and open source software is available on <http://www.usixml.org/>.

¹⁴ I will show an illustration in the chapter 5.

¹⁵ It's a reformulation of HTML in XML.

3.3 Examples used and decompilation tools

I list here the UIs that will guide my analysis throughout my thesis. I will show in the next chapter how they are specified in a resource script file (they are also illustrated in appendix B). I list after the tool that I've used to extract the resource script files.

I've selected five 32bit Windows applications¹⁶: **SciTE** v1.62 (a source code editor), **7-Zip File Manager** v4.15 (an file archiver), **Windirstat** v1.1.0 (a disk usage statistics viewer and cleanup tool), **RBG Editor** v3.9 (a small utility for selecting and creating RGB - Red, Green and Blue – colors) and **TableEdit** v2.60d5 (a program for creating, editing, printing and listening to tablature and sheet music). For each of these applications, I've pick up one resource to analyse.

Resources are additional data accompanying a Windows application. They are usually stored with the executable. They can also be stored in a separate file named Dynamic Link Libraries (*.dll). As the name suggests, these libraries are not linked into the executable (*.exe) file during its creation, rather they get loaded dynamically into the system memory at runtime (that is, they are linked to applications are loaded in memory). Windows is operating system which uses dynamic linking. This means that the same block of library code can be shared between several applications rather than each application containing copies of code. Another exemple is not a resource from a particular application, but from the Windows **Common Dialog Library** (comdlg32.dll) that provide common dialog boxes used by Windows applications (such as the 'open file' and 'print' dialog boxes).

A decompiler can be used to offer a thorough look at all of the resources in the compiled executable file. To bring into relief eventual variances in the informations they produce, I've used four resource editors¹⁷ (including a decompilation tool): **Resource Tuner** v1.95, **Restorator** v3.00, **Resource Builder** v2.3.0.8 and **Resource Hacker** v3.4.0. Sush utilities enable to view, modify, add or delete resources in executables, but also to extract them in a file with a .rc extention, displaying readable information about the structure of the individual resource. I discuss the differences in the files they generate at the end of the next chapter.

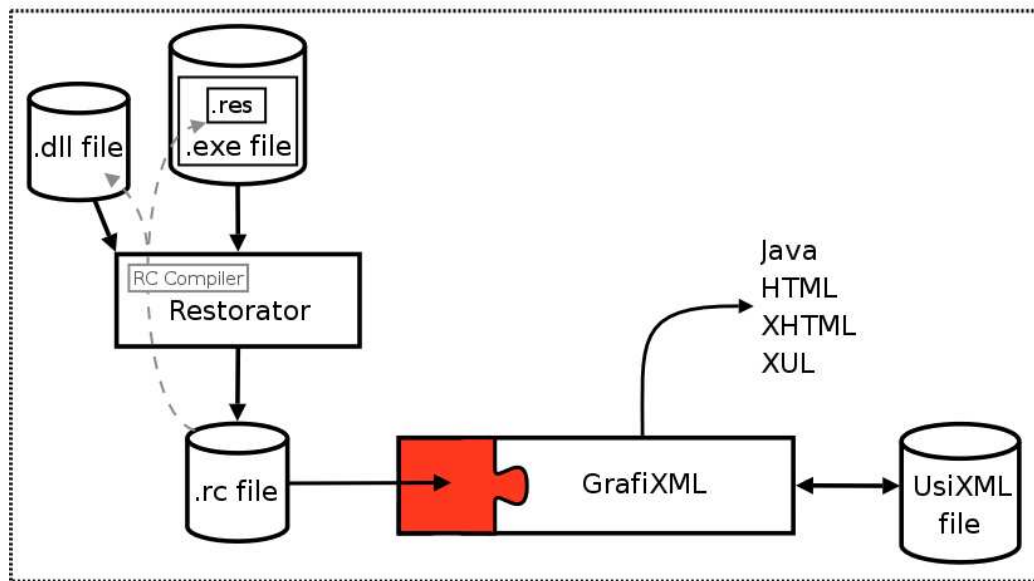
I can already see the link between the two text files describing a Windows UI (a resource file generated by a decompilation tool and another file expressed in the UsiXML language): by reproducing manually the UI in the GrafiXML editor, I can obtain a first specification. The two types of file obtained, describing on its own manner the interface, are always a textual representation of the resource. They can be then compared.

¹⁶ They can be found on the net. The three first applications are sharewares (from <http://www.framasoft.net> and the two others are free for try.

¹⁷ They can be downloaded on <http://www.restuner.com>, <http://www.bome.com/Restorator> (a shareware version limited to a 30-day trial period), <http://www.resource-builder.com> (also limited to a 30-day trial period) and <http://www.angusj.com/resourcehacker> respectively.

3.4 Selected method

The goal to achieve in this thesis can be reformulate as the development of a reverse engineering tool which will read any Windows resource script file and generate into GrafiXML the corresponding UsiXML specifications (at the CUI level). Here is an illustration of the selected method:



We have beforehand to extract the resource script file from an executable (Windows uses the filename extension ".exe"), here for example with Restorator. Note that these tools incorporate also an internal resource script compiler (called a RC compiler). Once a resource compiled, the resources are modified in the application's executable file.

4 Windows resource files

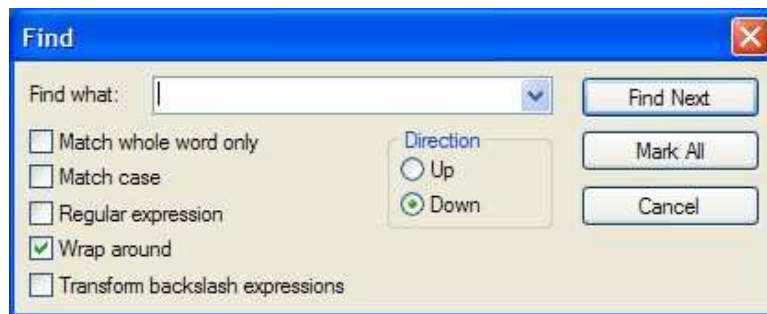
In the previous chapter I've stated my problem which is the reverse engineering of Windows resource script files. I see in this chapter the content of these source files and give a meta-model describing them. In the next chapter, I present the target language which will be used to specify these file in a higher level of abstraction.

4.1 Examples of resources

In this section I show the examples that I've chosen and how they are described in the resource script file (*.rc). Each resource is extracted by one particular decompiler. To compare the information generated, the resources given by the other decompilers are in appendix B. In the fourth section I'll make a discussion over the different decompilers and suggest my favourite. Each example come from one application mentioned in the chapter 3: the files that have been used in the decompilation tools to obtain these resources are SciTE.exe, 7zFM.exe, windirstat.exe, rgb.exe, comdlg32.dll and tabledit.exe.

4.1.1 First example from SciTE

The first graphical object is a dialog box called up by selecting "Find" from the "Search" menu, and will serve as main illustration for this chapter:



With a resource editor, the resources can be saved in an ASCII resource script file. This is what is generated by Restorator:

```

400 DIALOG 30, 73, 275, 84
STYLE DS_SETFONT | DS_MODALFRAME | DS_3DLOOK | WS_POPUPWINDOW | WS_CAPTION
CAPTION "Find"
FONT 8, "MS Shell Dlg"
{
    LTEXT "Fi&nd what:", -1, 5, 7, 45, 8
    COMBOBOX 222, 50, 5, 145, 50, CBS_DROPDOWN | CBS_AUTOHSCROLL
    AUTOCHECKBOX "Match &whole word only", 232, 5, 22, 120, 10, WS_GROUP
    AUTOCHECKBOX "Match &case", 233, 5, 34, 130, 10, WS_GROUP
    AUTOCHECKBOX "Regular &expression", 239, 5, 46, 120, 10, WS_GROUP
    AUTOCHECKBOX "Wrap aroun&d", 240, 5, 58, 120, 10, WS_GROUP
    AUTOCHECKBOX "Transform &backslash expressions", 241, 5, 70, 160, 10, WS_GROUP
    GROUPBOX "Direction", -1, 135, 22, 60, 34, WS_GROUP
    AUTORADIOBUTTON "&Up", 234, 140, 30, 45, 12, WS_GROUP | NOT WS_TABSTOP
    AUTORADIOBUTTON "&Down", 235, 140, 42, 45, 12, NOT WS_TABSTOP
    DEFPUSHBUTTON "&Find Next", 1, 205, 5, 65, 14, WS_GROUP
    PUSHBUTTON "&Mark All", 245, 205, 23, 65, 14
    PUSHBUTTON "Cancel", 2, 205, 41, 65, 14
}

```

The first line gives a name to the dialog box (in this case, 400). The name is followed by the keyword DIALOG and four numbers. The first two numbers are the x- and y-coordinates of the dialog box when the dialog box is invoked by the program. The second two numbers are the width and height of the dialog box.¹⁸ These coordinates and sizes are not pixels, but are based on the size of a system font character (in this case, an 8-point MS Shell Dlg font¹⁹): x-coordinates and width are expressed in units of 1/4 of an average character width and y-coordinates and height are expressed in units of 1/8 of a character height.²⁰ Because system font characters are often approximately twice as high as they are wide, the units on both the x- and y-axes are about the same. The second line defines some style for the dialog box (a combination of window style and dialog style). Within the left and right brackets are defined the child window controls that will appear in the dialog box. This dialog box uses six types of child window controls: a left-justified text, a combo box, a group box, radio buttons and push buttons. The first number is a value that the child window uses to identify itself when communicating to its parent (the dialog box window).²¹ The next four numbers set the position of the child window control (relative to the upper left corner of the dialog box's client area) and the size. Some style flag can also follow to define more precisely the appearance and functionality of the control.

4.1.2 Second example from 7-Zip File Manager

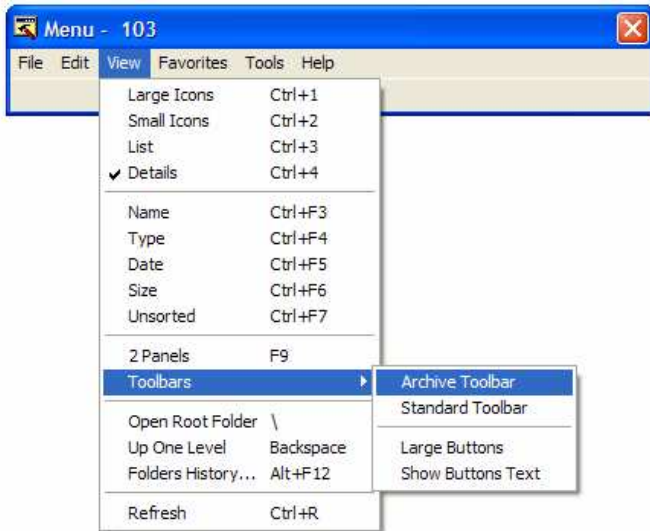
I take an interest here to the menu bar of an application. With a resource editor (here Resource Hacker), menus and Dialogs can also be viewed as they would appear in a running application. The following is the extracted menu resource.

¹⁸ Notice that if the dialog box has a caption bar (which is the case here), these measurements concern the dialog box's client area, and the caption bar will be shown above the y-coordinate.

¹⁹ In fact, MS Shell Dlg is not a physical font, rather a face name of a nonexistent font. It ensures the previous Windows operating system compatibility. It can be specified in either the Setup file during the installation process or when customizing a local system by double-clicking Control Panel's Regional Options icon. Because the font generated using MS Shell Dlg is different on different versions of Windows, the dialog box can look different depending on the version.

²⁰ This allows using coordinates and sizes that will retain the general dimensions and look of the dialog box regardless of the resolution of the video display.

²¹ Because the text control does not send messages back to its parent, this value is set to -1.



```

103 MENU
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
{
POPUP "&File" {...}
POPUP "&Edit" {...}
POPUP "&View"
{
MENUITEM "Large Icons\tCtrl+1", 410
MENUITEM "Small Icons\tCtrl+2", 411
MENUITEM "&List\tCtrl+3", 412
MENUITEM "&Details\tCtrl+4", 413, CHECKED
MENUITEM SEPARATOR
MENUITEM "Name\tCtrl+F3", 420
MENUITEM "Type\tCtrl+F4", 421
MENUITEM "Date\tCtrl+F5", 422
MENUITEM "Size\tCtrl+F6", 423
MENUITEM "Unsorted\tCtrl+F7", 424
MENUITEM SEPARATOR
MENUITEM "&2 Panels\tF9", 450
POPUP "Toolbars"
{
MENUITEM "Archive Toolbar", 461
MENUITEM "Standard Toolbar", 460
MENUITEM SEPARATOR
MENUITEM "Large Buttons", 462
MENUITEM "Show Buttons Text", 463
}
MENUITEM SEPARATOR
MENUITEM "Open Root Folder\t\\", 430
MENUITEM "Up One Level\tBackspace", 431
MENUITEM "Folders History...\tAlt+F12", 432
MENUITEM SEPARATOR
MENUITEM "&Refresh\tCtrl+R", 440
}
POPUP "F&avorites" {...}
POPUP "&Tools" {...}
POPUP "&Help" {...}
}

```

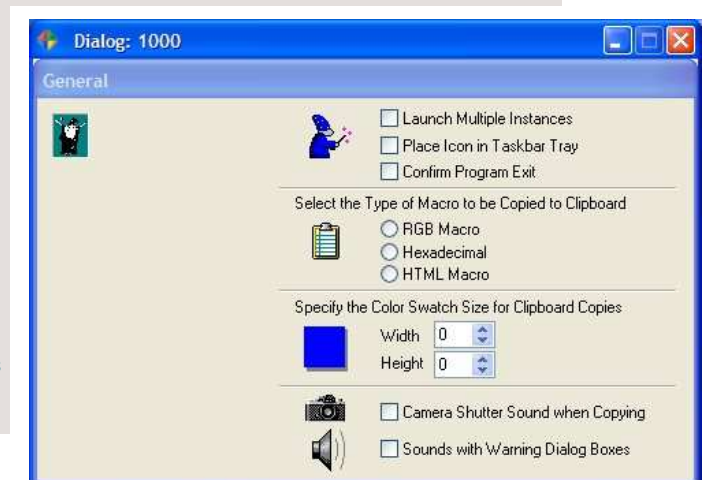
4.1.3 Third example from RGB Editor

The next example is the options dialog from the “File” menu of the application. It’s now Resource Tuner that is used to see the resource:

```

1000 DIALOG 0, 0, 320, 172
STYLE DS_SETFONT | DS_MODALFRAME | WS_CAPTION | WS_POPUP
CAPTION "General"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Sans Serif"
{
CONTROL 1999,-1,"STATIC",SS_BITMAP|SS_REALSIZEIMAGE|SS_SUNKEN,7,7,18,20
ICON 2655, -1, 132, 7, 18, 20
AUTOCHECKBOX "&Launch Multiple Instances", 108, 167, 5, 99, 8
AUTOCHECKBOX "&Place Icon in Taskbar Tray", 111, 167, 17, 99, 8
AUTOCHECKBOX "Confirm Program &Exit", 112, 167, 29, 79, 8
LTEXT -1, 118, 40, 195, 1, NOT WS_GROUP | SS_ETCHEDHORZ
LTEXT "Select the Type of Macro to be Copied to Clipboard",-1,125,43,165,8
ICON 2535, -1, 130, 55, 18, 20
AUTORADIOBUTTON "&RGB Macro", 101, 167, 55, 52, 8, WS_GROUP
AUTORADIOBUTTON "He&xadecimal", 102, 167, 65, 56, 8
AUTORADIOBUTTON "HTML &Macro", 103, 167, 75, 56, 8
LTEXT -1, 118, 86, 195, 1, SS_ETCHEDHORZ
LTEXT "Specify the Color Swatch Size for Clipboard Copies",-1,125,90,165,8
ICON 2000, -1, 130, 103, 18, 20
LTEXT "&Width", -1, 167, 103, 22, 8
EDITTEXT 104, 193, 100, 30, 12, ES_AUTOHSCROLL | ES_NUMBER
CONTROL "Spin1", 106, "msctls_updown32", 0x000000B7, 222, 100, 10, 12
LTEXT "&Height", -1, 167, 115, 22, 8
EDITTEXT 105, 193, 114, 30, 12, ES_AUTOHSCROLL | ES_NUMBER
CONTROL "Spin2", 107, "msctls_updown32", 0x000000B7, 222, 114, 10, 12
LTEXT -1, 118, 129, 195, 1, NOT WS_GROUP | SS_ETCHEDHORZ
ICON 2815, -1, 130, 131, 18, 20
AUTOCHECKBOX "&Camera Shutter Sound when Copying",109,167,137,133,8
ICON 2525, -1, 131, 149, 18, 20
AUTOCHECKBOX "&Sounds with Warning Dialog Boxes",110,167,154,126,8
}

```

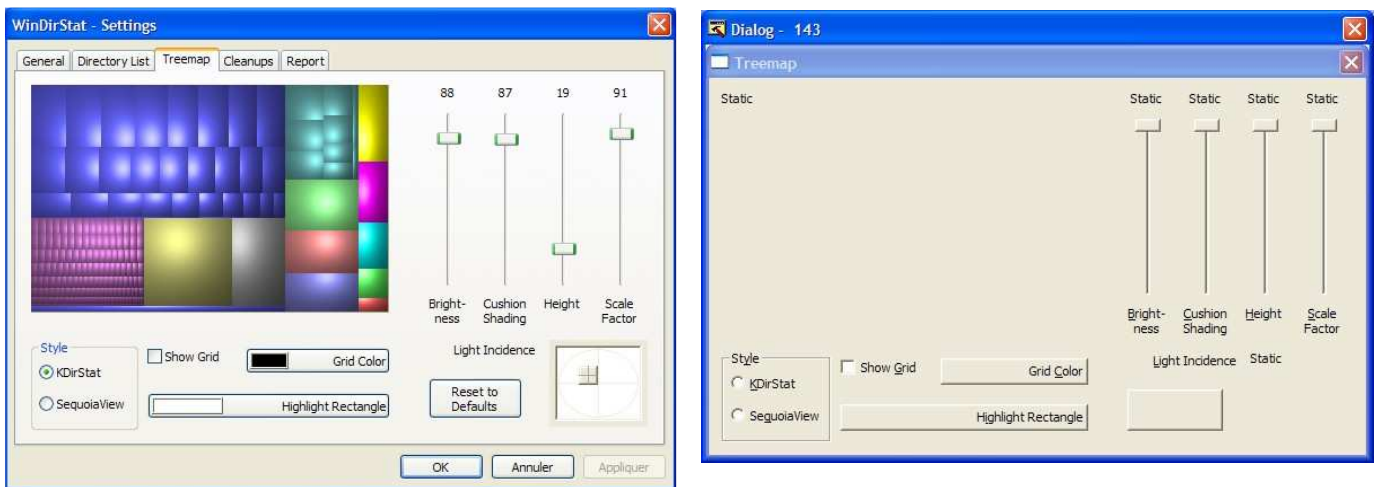


It happens that styles are expressed with hexadecimal number instead of flag identifiers. In reality, a flag correspond to a bit (or a group of bits for some mutual exclusive flags) positioned in a word of 32 bits in memory (as in the binary .res resource file): 1 if the style is present, 0 else. In this case, the hexadecimal number B7 for the up-down control to the binary

number 10110111, which correspond to the UDS_WRAP | UDS_SETBUDDYINT | UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS | UDS_NOTHOUSANDS flags combination. Note also that the bitmap (the first control) has not the same dimension in Resource Tuner that in the application (see in appendix).

4.1.4 Fourth example from WinDirStat

This example corresponds to the dialog box called up by selecting “Configure WindirStat...” from the “Options” menu. Notice that the tab control doesn’t appear to the right. In fact, there is one dialog box specification in the resource file for each tab item (the dialog box called “WinDirStat” which aggregates them is missing).



The following is the related part of the resource script file (from Resource Hacker):

```

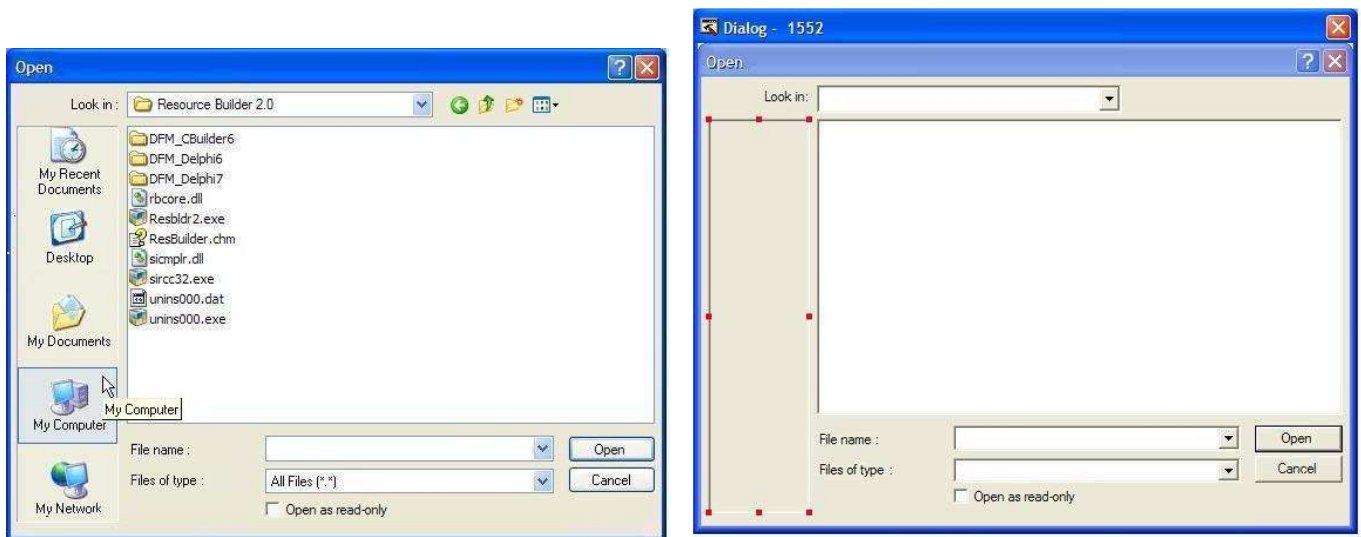
143 DIALOGEX 0, 0, 380, 202
STYLE DS_FIXEDSYS | WS_CHILD | WS_DISABLED | WS_CAPTION | WS_SYSMENU
CAPTION "Treemap"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Shell Dlg", FW_NORMAL, FALSE, 1
{
    CONTROL "&Bright-\nness", -1, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 237, 122, 32, 18
    CONTROL "", 1212, "msctls_trackbar32", TBS_VERT | TBS_BOTH | TBS_NOTICKS | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 245, 17, 16, 104
    CONTROL "&Cushion\nShading", -1, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 271, 122, 32, 18
    CONTROL "", 1211, "msctls_trackbar32", TBS_VERT | TBS_BOTH | TBS_NOTICKS | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 279, 17, 16, 104
    CONTROL "&Height", -1, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 305, 122, 32, 18
    CONTROL "", 1210, "msctls_trackbar32", TBS_VERT | TBS_BOTH | TBS_NOTICKS | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 313, 17, 16, 104
    CONTROL "&Scale\nFactor", -1, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 339, 122, 32, 18
    CONTROL "", 1209, "msctls_trackbar32", TBS_VERT | TBS_BOTH | TBS_NOTICKS | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 347, 17, 16, 104
    CONTROL "&Light Incidence", -1, STATIC, SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP, 246, 147, 59, 8
    CONTROL "Static", 1220, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 313, 146, 58, 48
    CONTROL "", 1034, BUTTON, BS_PUSHBUTTON | BS_MULTILINE | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 242, 167, 55, 22
    CONTROL "St&ytle", -1, BUTTON, BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 7, 146, 63, 49
    CONTROL "&KDirStat", 1213, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 12, 159, 42, 10
    CONTROL "Se&quoiaView", 1214, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 12, 176, 53, 10
    CONTROL "Show &Grid", 1022, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 76, 150, 54, 10
    CONTROL "Grid &Color", 1030, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 134, 150, 85, 14, 0x00001000
    CONTROL "H&ighlight Rectangle", 1202, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 76, 175, 143, 14, 0x00001000
    CONTROL "Static", 1219, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 339, 7, 32, 8
    CONTROL "Static", 1218, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 305, 7, 32, 8
    CONTROL "Static", 1217, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 271, 7, 32, 8
    CONTROL "Static", 1216, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 237, 7, 32, 8
    CONTROL "Static", 1215, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 7, 7, 211, 124
}

```

In this case, the two hexadecimal numbers 1000 means that the thirteenth bit from the right²² is set to 1, which correspond to the flag `WS_EX_RIGHT` (since it is an extended dialog, the controls can have extended styles that occur here to the end of a line). Using this style with a push button control has the same effect as using the `BS_RIGHT` style that right-justifies the text in the button rectangle.

4.1.5 Fifth example from the Common Dialog Library

I've noticed that the 'open file' dialog box was not present in the resource of SciTE. In fact, the application uses one of the Windows dialog boxes stored in a specific file (`comdlg32.dll`) that many applications import in their interfaces.²³ This is precisely the dialog often displayed by selecting 'open' from the 'file' menu (or by pressing `CTRL + O`) that serve as example here:



```
1552 DIALOGEX 0, 0, 370, 237
STYLE DS_MODALFRAME | DS_CONTEXTHELP | WS_POPUP | WS_VISIBLE | WS_CLIPCHILDREN | WS_CAPTION | WS_SYSMENU
CAPTION "Open"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Shell Dlg"
{
    CONTROL "Look &in :", 1091, STATIC, SS_LEFT | SS_NOTIFY | WS_CHILD | WS_VISIBLE | WS_GROUP, 4, 7, 57, 8, 0x00001000
    CONTROL "", 1137, COMBOBOX, CBS_DROPDOWNLIST|CBS_OWNERDRAWFIXED|CBS_HASSTRINGS|WS_CHILD|WS_VISIBLE|WS_VSCROLL|WS_TABSTOP,
    66, 4, 174, 300
    CONTROL "", 1088, STATIC, SS_LEFT | WS_CHILD, 248, 4, 80, 14
    * CONTROL "", 1184, "ToolBarWindow32", 0x50012B4C, 4, 22, 58, 208, 0x00000200
    CONTROL "", 1120, LISTBOX, LBS_NOTIFY | LBS_NOINTEGRALHEIGHT | LBS_MULTICOLUMN | WS_CHILD | WS_BORDER | WS_HSCROLL, 66, 22, 300, 156
    CONTROL "File &name :", 1090, STATIC, SS_LEFT | SS_NOTIFY | WS_CHILD | WS_VISIBLE | WS_GROUP, 67, 187, 71, 8
    CONTROL "", 1152, EDIT, ES_LEFT | ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 144, 184, 164, 12
    CONTROL "", 1148, "ComboBoxEx32", 0x50210042, 144, 184, 164, 150
    CONTROL "Files of &type :", 1089, STATIC, SS_LEFT | SS_NOTIFY | WS_CHILD | WS_VISIBLE | WS_GROUP, 67, 203, 71, 8
    CONTROL "", 1136, COMBOBOX, CBS_DROPDOWNLIST | WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 144, 201, 164, 100
    CONTROL "Open as &read-only", 1040, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 144, 217, 160, 8
    CONTROL "&Open", 1, BUTTON, BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 316, 184, 50, 14
    CONTROL "Cancel", 2, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 316, 200, 50, 14
}

```

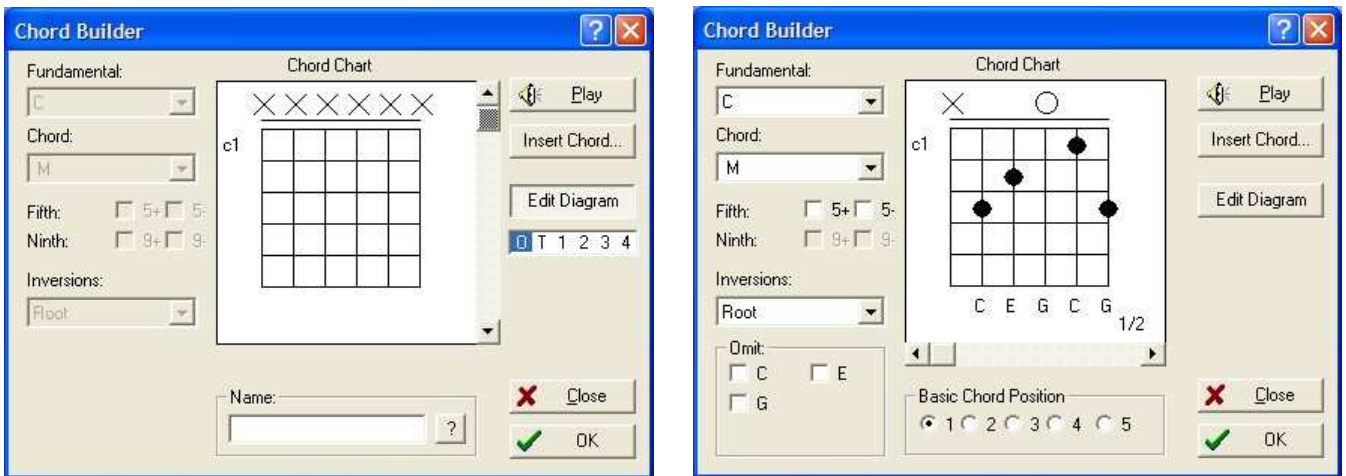
The first static control has also the number 1000 as extended style corresponding to `WS_EX_RIGHT`, which for static controls has the same effect as using `SS_RIGHT`. For the toolbar control, the number 50012B4C is equivalent to the styles `WS_CHILD`, `WS_VISIBLE`, `WS_TABSTOP`, `CCS_NORESIZE`, `CCS_NOPARENTALIGN`, `CCS_NODIVIDER`, `TBSTYLE_TOOLTIPS`,

²² The number 1000 in hexadecimal is equivalent to 1000000000000 in binary.
²³ I've found another decompiler (eXeScope) which lists in addition the .dll files used by an application.

TBSTYLE_WRAPABLE, TBSTYLE_CUSTOMERASE and TBSTYLE_FLAT, and the number 200 is equivalent to the extended style WS_EX_CLIENTEDGE. For the extended combo box, the number 50210042 is equivalent to WS_CHILD, WS_VISIBLE, WS_VSCROLL, WS_TABSTOP, CBS_DROPDOWN and CBS_AUTOHSCROLL. This control superposes the edit control (see the size numbers).²⁴

4.1.6 Sixth example from TabEdit

The last example is a dialog box that appears when clicking the “New” push button from the dialog “Chord Manager” called up from the “Edit” menu.



When pushing on “Edit Diagram”, the dialog box changes of look. But all the new emerged controls (and those that disappear) belong to the same dialog box, and are specified in sequence in the resource script file. It makes it difficult to know what the dialog look like with only the information below. The program determines if a child window is currently hidden and disabled (it’s not enough to be not visible, it has to no longer respond to mouse or keyboard input), for example by a call to the Windows functions ShowWindow (with SW_HIDE as one of the parameter) and EnableWindow (with FALSE as one of the parameter).

There is the resource with Resource Builder:

```

22 DIALOG 76, 29, 284, 174
STYLE DS_3DLOOK | DS_SETFONT | DS_MODALFRAME | DS_NOIDLEMSG | DS_CONTEXTHELP | WS_POPUP | WS_SYSMENU | WS_CAPTION
CAPTION "Chord Builder"
FONT 8, "MS Sans Serif"
LANGUAGE LANG_NEUTRAL, 0
BEGIN
CONTROL "&Fundamental:", 24, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 4, 60, 10
CONTROL "", 101, "COMBOBOX", CBS_DROPDOWNLIST | WS_CHILD | WS_VSCROLL | WS_TABSTOP | WS_VISIBLE, 8, 15, 76, 115
CONTROL "&Chord:", 25, "STATIC", SS_LEFT | WS_CHILD | WS_GROUP | WS_VISIBLE, 8, 31, 54, 10
CONTROL "", 102, "COMBOBOX", CBS_DROPDOWNLIST | WS_CHILD | WS_VSCROLL | WS_TABSTOP | WS_VISIBLE, 8, 42, 76, 100
CONTROL "&Fifth:", 26, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 62, 32, 10
CONTROL "5+", 301, "BUTTON", BS_AUTOCHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE, 47, 61, 21, 10
CONTROL "5-", 302, "BUTTON", BS_AUTOCHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE, 69, 61, 21, 10
CONTROL "&Ninth:", 27, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 74, 38, 10
CONTROL "9+", 401, "BUTTON", BS_AUTOCHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE, 47, 73, 21, 10
CONTROL "9-", 402, "BUTTON", BS_AUTOCHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE, 69, 73, 21, 10
CONTROL "&Inversions:", 28, "STATIC", SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 90, 63, 10
CONTROL "", 103, "COMBOBOX", CBS_DROPDOWNLIST | WS_CHILD | WS_VSCROLL | WS_TABSTOP | WS_VISIBLE, 8, 101, 76, 55
CONTROL "&1", 700, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_GROUP | WS_TABSTOP | WS_VISIBLE, 98, 149, 17, 10

```

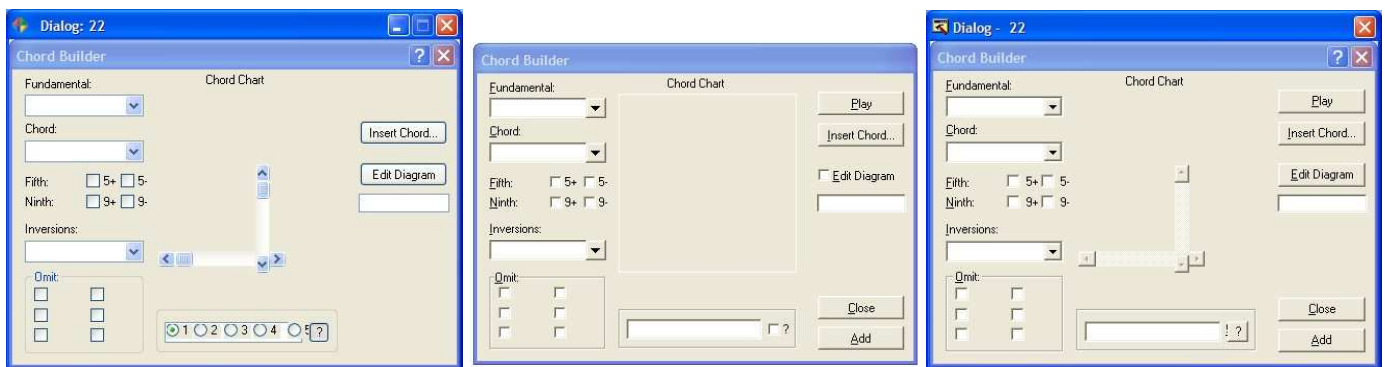
²⁴ I don’t see the reason because a drop-down combo box has also an edition field...


```

CONTROL "&2", 701, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE , 116, 149, 17, 10
CONTROL "&3", 702, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE , 135, 149, 17, 10
CONTROL "&4", 703, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE , 154, 149, 17, 10
CONTROL "&5", 704, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE , 176, 149, 17, 10
CONTROL "", 106, "BUTTON", BS_GROUPBOX | WS_CHILD | WS_GROUP | WS_VISIBLE , 92, 138, 116, 29
CONTROL "", 600, "SCROLLBAR", SBS_HORZ | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 93, 107, 81, 9
CONTROL "", 601, "SCROLLBAR", SBS_VERT | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 155, 56, 9, 64
CONTROL "&Omit:", 799, "BUTTON", BS_GROUPBOX | WS_CHILD | WS_GROUP | WS_VISIBLE , 8, 118, 76, 49
CONTROL "", 800, "BUTTON", BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 14, 128, 30, 10
CONTROL "", 801, "BUTTON", BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 50, 128, 30, 10
CONTROL "", 802, "BUTTON", BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 14, 140, 30, 10
CONTROL "", 803, "BUTTON", BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 50, 140, 30, 10
CONTROL "", 804, "BUTTON", BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 14, 152, 30, 10
CONTROL "", 805, "BUTTON", BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 50, 152, 30, 10
CONTROL "Chord Chart", 30, "STATIC", SS_CENTER | WS_CHILD | WS_VISIBLE , 108, 2, 68, 10
CONTROL "", 999, "STATIC", SS_SIMPLE | WS_CHILD , 92, 12, 116, 108
CONTROL "&Close", 2, "BUTTON", BS_OWNERDRAW | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 222, 135, 56, 14
CONTROL "&Add", 1, "BUTTON", BS_OWNERDRAW | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 222, 153, 56, 14
CONTROL "&Play", 131, "BUTTON", BS_OWNERDRAW | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 222, 11, 56, 14
CONTROL "&Insert Chord...", 105, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 222, 30, 56, 14
CONTROL "", 107, "EDIT", ES_LEFT | WS_CHILD | WS_BORDER | WS_TABSTOP | WS_VISIBLE , 97, 149, 88, 12
CONTROL "?", 109, "BUTTON", BS_AUTOCHECKBOX | BS_PUSHLIKE | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 189, 149, 13, 12
CONTROL "&Edit Diagram", 108, "BUTTON", BS_CHECKBOX | BS_PUSHLIKE | WS_CHILD | WS_TABSTOP | WS_VISIBLE , 222, 55, 56, 14
CONTROL "", 110, "LISTBOX", LBS_MULTICOLUMN | LBS_NOINTEGRALHEIGHT | WS_CHILD | WS_BORDER | WS_TABSTOP | WS_VISIBLE , 221, 73, 58, 11
END

```

It happens that a dialog box looks different in the resource editor tools, but it's especially the case in this example:



Resource Tuner

Resource Builder

Resource Hacker

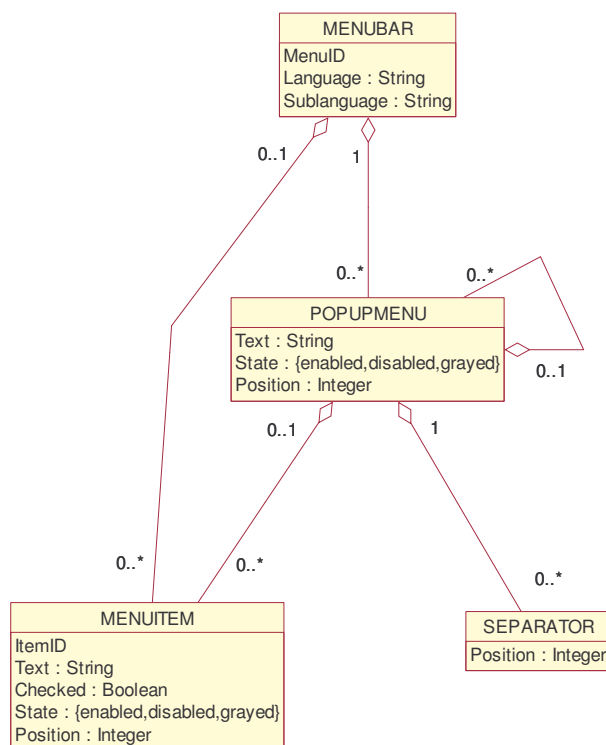
The objective of my thesis will be to attempt to faithfully reproduce in GrafiXML the graphical interface described by a given resource file, not what we really see on the screen when we use the application. A resource file gives us only a point of view, things can be managed internally in the program (a control with `WS_VISIBLE` can be not visible). Depending on the consistency of these input files, the interface specified can be different from the actual interface. The result described in UsiXML cannot naturally be better. My task is still to utilize as much as possible the informations from a resource file to restore the interface, even though a first loss of informations can occur before the reverse engineering process. The content of this files is what I will see in the next sections.

In addition, some elements not covered in UsiXML can be also lost. If it hasn't the counterpart of the property that a check box can look like a push button for example (this is what means the flag `BS_PUSHLIKE` for the check box labeled "Edit Diadram"), the control cannot have the desired appearance in the final result. In chapter 6 I'll see this other potential loss of information.

4.2 Resource files modelization

The purpose of this section is to structure informations that can be found in a resource script file. To modelize the problem in a standart way, I've used the UML (Unified Modeling Language) notation. This will also enable to have a direct model-to-model mapping. Indeed, the CUI model is represented with the UML notations, so the source and target meta-models use the same formalism. Only the resources of type menu and dialog box will reveal important. The two first sections give the related class diagrams.

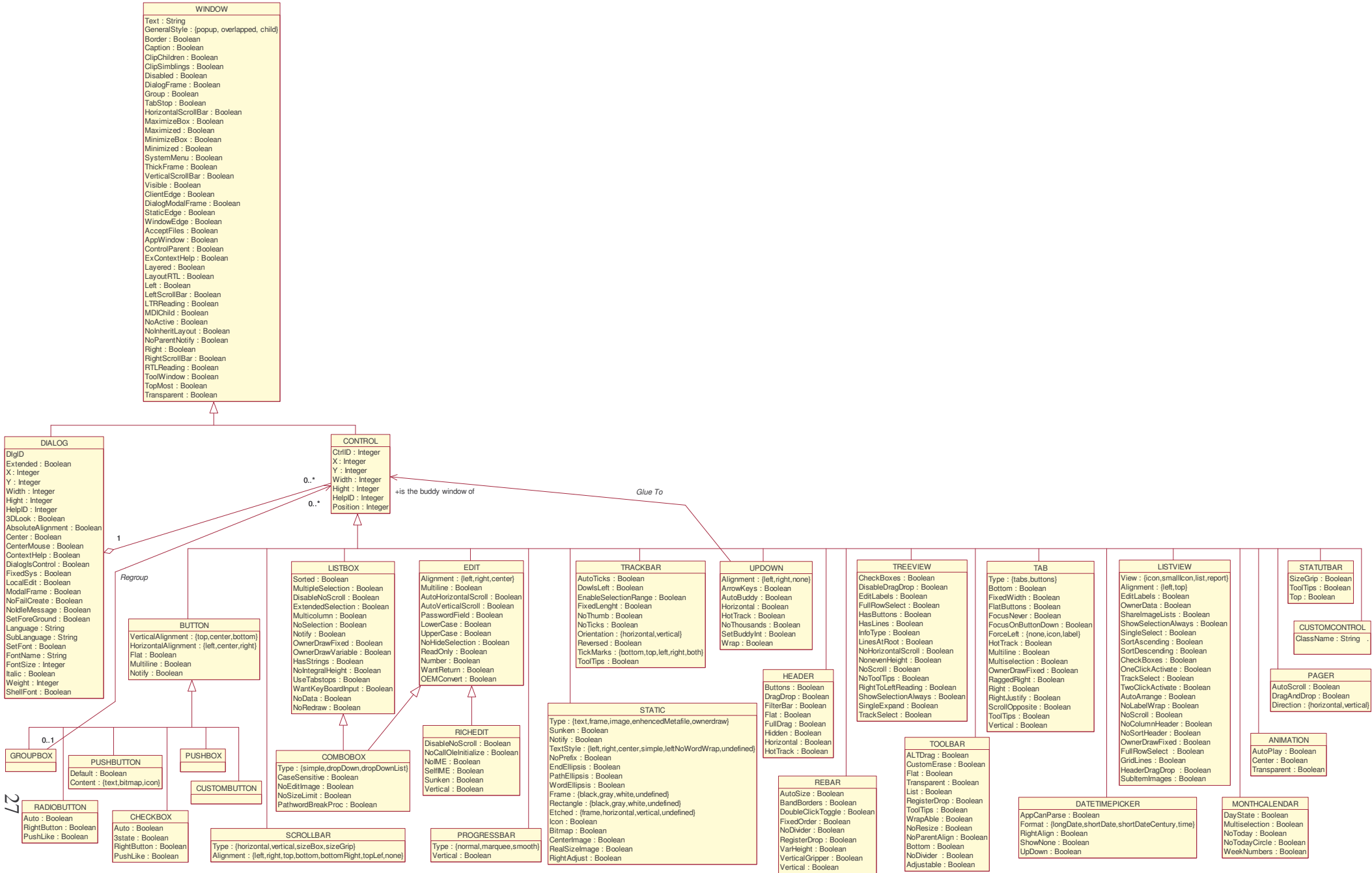
4.2.1 Menus class diagram



The class diagram modeling a resource of type menu is simple. Note that it's a modelization of a resource file, and not of the real user interface specified by the file. The link between a menu item and a dialog box cannot be represented because this relation is not specified in the file. This class diagram is then separated from the next one. It is uncommon to see menu items directly in a menu bar (represented by the aggregation relationship), but it's a valid composition.

4.2.2 Dialog boxes class diagram

Note that here I still include the relationships named "Glue To" and "Regroup" because they can be deduced from the positions and dimensions of controls specified in the file (even if these relationships are not explicitly specified).



4.2.3 Documentation

The full documentation of the diagrams is in appendix C. In this section I explain some concepts related to my examples.

There are three general styles of window (**WINDOW** class): a popup window is a temporary subsidiary window, a child window can divide a window in various regions and an overlapped window is a program's main application window.

In a graphical Windows-based application, dialog boxes (**DIALOG** class) are windows (rectangular areas of the screen) and then inherit the window's attributes. A dialog box, used to communicate with the user and to supply services that are too complicated to be in a menu, takes form of a pop-up window (**GeneralStyle** = popup) containing various child window controls through which the user interacts. Note that there is a curiosity in the fourth example : the dialog is defined as a child window. This is certainly because it is not really a dialog box, but a part of the dialog box to configure the software (a child window stay always in the same position relative to their parent when the user moves the parent). Dialog boxes can be modal or modeless, with frame often encountered.

A modal dialog box (**ModalFrame** = true) demands the user's attention before anything else can be done : when displayed, the user cannot switch between the dialog box and the window that created it (the user must explicitly end the dialog box, for instance by clicking a the Cancel button). The user can however switch to another program while displayed.²⁵ It is usually a pop-up window having a thick border, but a double border (with **DialogFrame** = true) is also common for modal dialog box. Some can also have a title bar (**Caption** = true) that identifies the dialog's purpose with the text put in it (value of **Text**), with sometimes a question mark as in the fifth and sixth example (**ContextHelp** = true).

A modeless dialog box allows the user to switch between the dialog box and the parent window, preferred when the user would find it convenient to keep the dialog box displayed for a while. It is often a pop-up window having a thin border (**Border** = true) and a caption bar (to let the user move the dialog box to another area of the display using the mouse). The dialog box can also have a system menu box (**SystemMenu** = true) to allows the user to select Move or Close from the system menu.

The controls (**CONTROL** class), contained in a dialog box to perform input and output tasks, are child windows (**GeneralStyle** = child). They can use some window's frame, as thin-line border (**Border** = true). You can see that the tool bar from the fifth uses a border with a

²⁵ Some dialogs do not allow even that the user can switch to another program while displayed: the dialog must be ended before the user does anything else in Windows (see **SysModal**).

sunken edge (**ClientEdge** = true). That is what the last check box should look like with such frame:



A control can use other window attributes. For example, it is displayed in gray rather than black when visible and disabled (**Disabled** = true) :



To illustrate the use of the TAB key and the arrow keys in a dialog box, let's go over the first example to see how they are used in a resource file (from Resource Hacker):

```
CONTROL "Fi&nd what:", -1, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 5, 7, 45, 8
CONTROL "", 222, COMBOBOX, CBS_DROPDOWN | CBS_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 50, 5, 145, 50
CONTROL "Match &whole word only", 232, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 5, 22, 120, 10
CONTROL "Match &case", 233, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 5, 34, 130, 10
CONTROL "Regular &expression", 239, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 5, 46, 120, 10
CONTROL "Wrap aroun&d", 240, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 5, 58, 120, 10
CONTROL "Transform &backslash expressions", 241, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 5, 70, 160, 10
CONTROL "Direction", -1, BUTTON, BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP, 135, 22, 60, 34
CONTROL "&Up", 234, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP, 140, 30, 45, 12
CONTROL "&Down", 235, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 140, 42, 45, 12
CONTROL "&Find Next", 1, BUTTON, BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 205, 5, 65, 14
CONTROL "&Mark All", 245, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 205, 23, 65, 14
CONTROL "Cancel", 2, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 205, 41, 65, 14
```

When the Find dialog box is first invoked, the default input focus is set to the first control specified in the resource that has the WS_TABSTOP flag (**TabStop** = true), that is the combobox control.²⁶ The other controls will usually receive their input focus when they are clicked with the mouse, but the user can also move among the controls having this flag using the TAB key (in the order they figure in the file²⁷ and with a cycle from the last control to the first). At first sight it is not the case with the 'Down' radio button, but in practice it is attainable. The style is probably added to the default checked radio when the window control is created. When the input focus is changed from a radio button to the another within a group, the system automatically assign the style to the newly checked control. This ensures that the input focus will always be on the most recently selected control when the user moves to the group using the TAB key. From this it may be deduced that flags can be added and removed for convenience in the run time by the program (see also the **LISTVIEW** class and think of a user arranging items in other views²⁸). We have here again the feeling that resource files don't always give a full description of a dialog box.



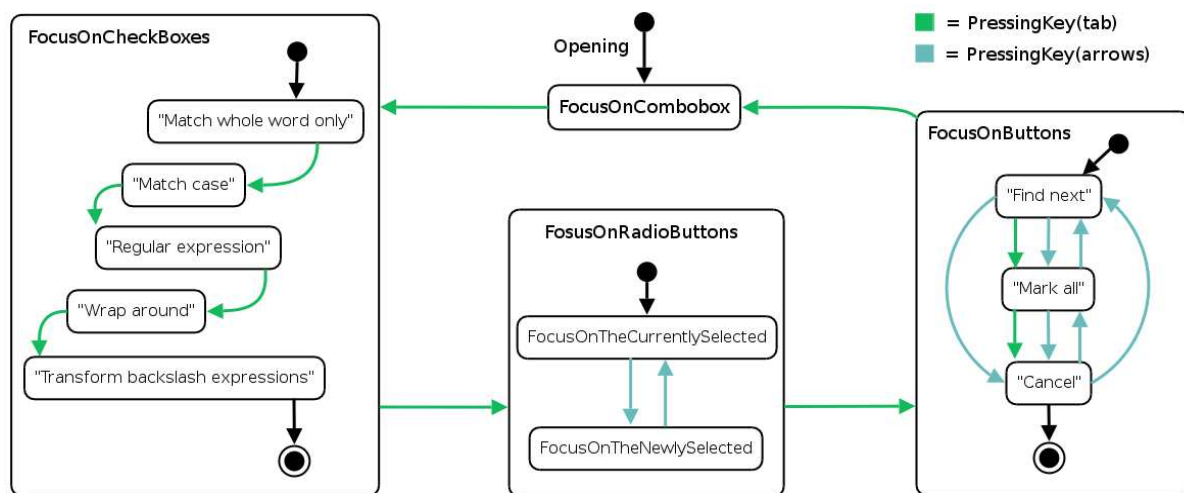
²⁶ And that is visible and not disabled. If no such control exists, the system sets the default input focus to the first control in the template.

²⁷ The order of definition in a dialog resource is represented by the **Position** attribute.

²⁸ Note that in the fifth example Windows use a list box control instead of a list-view control.

The user can use the arrow keys to shift the input focus between the two radio buttons²⁹, that is from the first radio button that has the WS_GROUP flag (**Group** = true) up to, but not including, the next control that has this flag (the default push button). The tree push buttons are then also grouped, the flag added to the static control marking the end of the group.³⁰ Nevertheless, all the check box controls get the flag to prevent the arrow keys from doing anything when these controls have the input focus (the group contains only one control, so pressing an arrow key has no effect). Should the combobox have this flag? If the combobox control and the static control are in a group, the interface should be not quite correct if the former got the focus (because it can't do anything with it). It's not important since the cursor changes in the combobox edit field and moves around when pressing the arrow keys.

There is a state chart describing the input focus in this dialog box:



There is another keyboard interface which can be specified in the value of the Text attribute: the mnemonic is the letter that follows an ampersand (&). The user can then move the input focus to any controls by pressing the ALT key and the mnemonic. However, for static control the focus moves to the first control having Tabstop set to true defined in the file after the static control definition and containing the specified mnemonic (see "Fi&nd wath" above).

In addition to those inherited, each control has specific attributes. I will define some of them.

The text (value of Text) in the rectangle of the first static control (**STATIC** class) from the first example is left-justified (**TextStyle** = left), which is not the case for example for the first one from the fourth example (**TextStyle** = center). But static controls can be used to draw frames or lines separating other controls like in the third example: three static controls are 1 dialog unit high with etched top and bottom edges (**EtchedHorizontal** = true). They can also be used to display images like the second control of the same example (**Icon** = true). In this case, the width and the height values are ignored, the control is automatically resized to fit the image.

²⁹ The selected radio button is turned on and others in the group are turned off automatically.

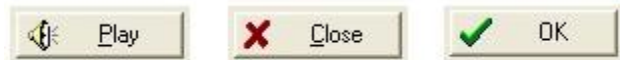
³⁰ The user can also move between these buttons with the TAB key, but only in one order (SHIFT+TAB to move to the previous control)

But the first control (**Bitmap** = true) uses a style that prevents this (**RealSizeImage** = true). This is why the 154x256 bitmap image (according to Resource Tuner bitmap resource) doesn't fit all the left area of the dialog as in reality.³¹

As mentioned in the fourth example (where two controls have **HorizontalAlignment** = right), text in a push button (**PUSHBUTTON** class) can also be positioned in different way in the rectangle. In the same example, the text of a push button will be wrapped to multiple lines if it is too long to fit on a single line in the rectangle (**Multiline** = true).³² There are other button controls that provides input to an application by notifying the parent window when the user clicks on the control with a mouse, like radio buttons (**RADIOBUTTON** class). When they have **RightButton** = true, they look like that one labelled 'Down' below. When they have **PushLike** = true, they look like that one labelled 'Up'.

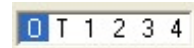


The check box (**CHECKBOX** class) 'Edit diagram' from the sixth example has also this last property. Other buttons can be owner-drawn buttons (**CUSTOMBUTTON** class). This type allows complete freedom in the button's appearance. Buttons from the sixth example have this style to have image and text together:



A scroll bar control (**SCROLLBAR** class) is not a scroll bar added at the right or the bottom of a window by setting to true the window attribute **VerticalScrollBar** or **HorizontalScrollBar**, but a child window control that can appears anywhere in the parent window, vertically (**Type** = vertical) or horizontally (**Type** = horizontal). There is such controls in the sixth example.

The list box (**LISTBOX** class) from the sixth example (as from the fifth example) has two specific properties. Items are arranged in multiple columns and the list box scrolls horizontally instead of vertically (**Multicolumn** = true).³³



The size of the control is exactly the size specified when created (**NoIntegralHeight** =true). By default, Windows resizes a list box to not display partial items.

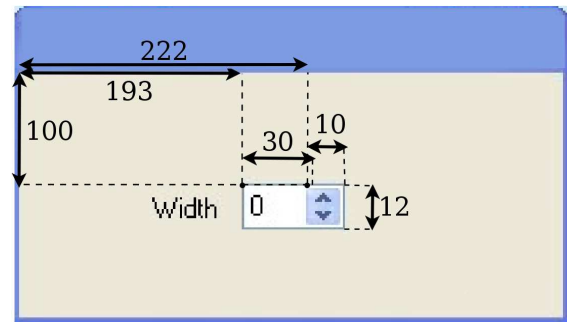
We can see in the third example two edit controls (**EDIT** class) which scrolls text to the right when the user types a digit (only allowed with **Number** = true) at the end of the line (**AutoHorizontalScroll** = true). This controls are the buddy windows of up-down controls (**UPDOWN** class) used to increment or decrement the value. To the user, they look like a single controls. It can be specified that an up-down control automatically selects the previous

³¹ In my opinion the developer have made a mistake in their resource statement resolved after in the program (it is true that when we read the flag name we could think to the opposite of what said the definition).

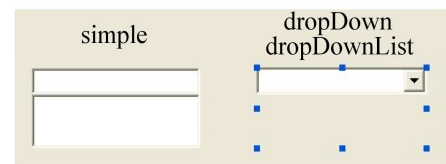
³² But it's surprising that the label doesn't in he resource file...

³³ In the sixth a horizontal scroll bar (**HorizontalScrollBar** = true) is added, but it is hidden because the list box is large enough to display all the items at once.

window in the z-order³⁴ as its buddy window. Let's look at the first up-down control (X = 222, Y = 100). **Alignment** = right positions it to next to the right edge of its buddy window (X = 193, Y = 100), the buddy window is then moved to the left and its width (30) is decreased to accommodate the width of the up-down control (10). **ArrowKeys** = true allows the user to press the arrow keys to increment and decrement the position when the buddy window has the focus.



The combo box (**COMBOBOX** class) from the first example is a drop-down one, those from the sixth example are drop-down list combo boxes. The first has a list box which is displayed when the user selects an icon next to the edit control (not all times as a simple combo box) and the current selection in the list is displayed in the edit control. The second is similar, except that the edit control is replaced by a static text item displaying the current selection. This table summarizes the three existing types:



	drop-down list	edit control
Type = dropDown	✓	✓
Type = dropDownList	✓	✗
Type = simple	✗	✓

In the dialog of the fourth example, the user can move a track bar, using either the mouse or the direction keys when the control has focus, to change the brightness of the treemap (in a range of 1 through 100). This vertical trackbars (**TRACKBAR** class and **Orientation** = vertical) have **NoTicks** = true (to not display any tick marks) and **TickMarks** = both (causes tick marks to be displayed on both sides of the control). The effect is that the thumb of the control appears as a rectangular box.

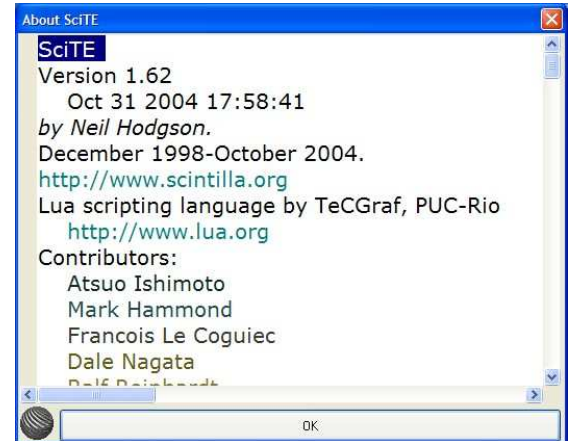
Because tool bars (**TOOLBAR** class) are especially used in an application's main window, it is unfrequent to find them in a resource files. We can still see this control in the Open dialog box illustrated in the fourth example (see the line with an asterisk). The user can then acces directly some directories (like "My Computer") when loking for a file. Tool tips (**Tooltips** = true) are descriptive text boxes that pop up when the user points to a tool bar button with the mouse cursor. As **Flat** is true the tool bar has a transparent look (both the tool bar and the buttons) that allows the client area under the tool bar to show through and enables hot tracking (a button is highlighted when the cursor moves over it). **WrapAble** set to true means by definition that the tool bar can have multiple lines of buttons. The buttons wrap to the next line when the tool bar becomes too narrow to include all buttons on the same line. Normally, the size and position of the tool bar window automatically set itself. The height is based on

³⁴ The term "z-order" refers to the order of objects along the Z-axis. In [coordinate geometry](#), X typically refers to the horizontal axis, Y to the vertical axis, and Z to the axis perpendicular to the plane (representing the depth of the stack of windows). The z-order information thus specifies the front-to-back ordering of objects on the screen.

the height of the buttons in the tool bar, the width is the same as the width of the parent window's client area and the control is positioned along the top of the parent window's client area. This behavior is turned off if **NoResize** is true (the control uses its specified width and height) and **NoParentAlign** is true (the control keeps its specified position within the parent window).

Such controls are controls that exist in Windows, corresponding to predefined window classes in Windows programming. But a custom-made child window (**CUSTOMCONTROL** class) can also be used, by defining its own window class. For example, some may want to replace a normal rectangle push button with rounded corner with an elliptical push button. Another example is the first control from the "About SciTE" dialog box shown to the right, and specified in a resource file as:

```
ABOUT DIALOG 26, 41, 350, 242
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About SciTE"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Sans Serif"
{
CONTROL "", 221, "Scintilla", 0x50000000, 1, 1, 346, 218
CONTROL "SciTE", -1, STATIC, SS_ICON | WS_CHILD | WS_VISIBLE, 1, 221, 32, 32
CONTROL "OK", 1, BUTTON, BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 26, 222, 322, 20
}
```



This control has been created to display a text using a specific font and various colors. Its only specific attribute **ClassName** set to "Scintilla" designates the name of the class that has been registered and defining the control. The name will be for example "ellipticalPushButton" if we suppose that a customized class with this name exist to use a personalized button.

4.2.4 Constraints

I talk here over some constraints that are not expressed in the class diagrams.

The inheritance from the WINDOW class is attribute-defined : controls are child window and dialog boxes are pop-up window. A dialog can inherit from extended window attributes (from ClientEdge to Transparent) only if it is also an extended dialog, that is if Extended = true. A control can inherit of an extended window attribute only if the dialog box that contain it is an extended one.³⁵ But we can see that some window attributes (extended or not) are not relevant for controls (and also for dialog boxes). Attributes that are usually used by controls are Child, Disabled, Visible, Border, TabStop, Group, VerticalScrollBar, HorizontalScrollBar. Note also that a combo box cannot use all the specific attributes of a list box and edit controls : it can use only the AutoHScroll, LowerCase, OEMConvert, UpperCase, Sort, DisableNoScroll, HasStrings, NoIntegralHeight, OwnerDrawFixed and OwnerDrawVariable attributes.

³⁵ Some attributes specific to the COMBOBOX and RICHEDIT classes can also be used in this condition.

A remark has to be made with the font in a dialog box. If one of the SetFont and ShellFont attributes is set to true, there is additional data in the resource file specifying the font and the size to use for text in the dialog box and each of its controls (and sometimes weight and italic informations). If SetFont = true, the system selects (if possible) a font according to the specified font data. If ShellFont = true, the system selects a font using the font data specified in the FontSize, Weight, and Italic attributes. The system font can vary between different versions of Windows. So having ShellFont = true with FontName = MS Shell Dlg for an extended dialog box, the application can use the system font no matter which system it is running on. The system maps this typeface such that the dialog box will use the Tahoma font on Windows XP, and the MS Sans Serif font on earlier systems. But ShellFont = true has no effect if the typeface is not MS Shell Dlg and the dialog is not extended. By default, the system draws all text in a dialog box using the default font of the system. The system always uses the system font for the dialog box title.

Most of constraints are already expressed with type declaration with enumerate values. For example, a menu item can be disabled and grayed in the same time. There still exists other constraints between attributes, like an attributes that can or not be combined with an other one. For example, a title bar cannot contain a question mark (ContextHelp = true) with a maximize box or a minimize box (MaximizeBox or MinimizeBox = true), or to make appear tabs at the left side of a tab control, multiple rows of tabs have to be able to be displayed when necessary (Vertical = true is valid only when Multiline = true). But this is the concern of graphical user interface programming. I will have an input file which is supposed to be correct, and even if it isn't, I have not to correct it (my task is to restore the dialog box in another language like it is specified). Moreover, the reason for these constraints can guarantee that a resource file respects them. The constraints that we can find in Windows documentation express in fact incompatibilities for flag identifiers. As I've yet mentioned it, the style flag is represented in memory by one bit at a specific position in a word or by n bits if it is mutually exclusive with others (where n is the minimum number of bits to represent the set S of exclusive flag, that is, $2^{n-1} < \text{cardinal number of } S \leq 2^n$). For example, the 19 different types of static controls are grouped in the five first bits (which can represent $2^5 = 36$ flags):

	4	3	2	1	0		4	3	2	1	0
SS_LEFT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	SS_USERITEM	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
SS_CENTER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	SS_SIMPLE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
SS_RIGHT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	SS_LEFTNOWORDWRAP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SS_ICON	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	SS_OWNERDRAW	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
SS_BLACKRECT	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	SS_BITMAP	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
SS_GRAYRECT	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	SS_ENHMETAFILE	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
SS_WHITERECT	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	SS_ETCHEDHORZ	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SS_BLACKFRAME	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	SS_ETCHEDVERT	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
SS_GRAYFRAME	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	SS_ETCHEDFRAME	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
SS_WHITEFRAME	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>						

A decompiler that checks the presence of bits when generating its resource text file could not specify for example that the text of a static control is both centered and right-justified in the rectangle (this is an icon).

4.3 Resource files structure

I show in this section the format of a resource script file and how the user interface specified in this language instantiates my UML class diagrams (the meta-model).

4.3.1 Resources of type dialog box

4.3.1.1 Dialog box template

We have seen in my examples that a dialog box is described in a specific language, specifying its height, width, style, and the controls it contains (with again a specific size, placement and style). This type of resource is defined in a dialog box template included in the resource script file. Here is the format:

```
<id> DIALOG[EX] <x>, <y>, <width>, <height> [, <helpId>]  
STYLE <style>*  
[EXSTYLE <ex_style>*]  
CAPTION "<text>"  
LANGUAGE <language>, <sublanguage>  
FONT <pointsize>, "<typeface>" [, <weight>, <italic>]  
{  
    <control_def>  
}
```

Note that the order of the lines after the first and before the controls block can sometime be different. With Resource Builder, <control_def> also go between BEGIN and END instead of { and }. Restorator put a comma after the keyword EXSTYLE.

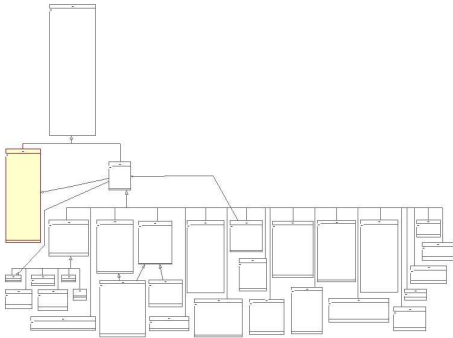
I've used the regular BNF notations as meta-language to define the syntax. In this way, it will be easier show the correspondence between a class or an attribute of the class diagram and a token of the language at a specific place in the template. The next table lists these correspondences. Before, I give the notation used to express the rules in the table. The keywords in bold appear as it stand in the resource file. Except for the first line which is always specified, these keywords are missing in the template when nothing follows (that is, the entire line is missing).

Meta-language used to describe the syntax :

- <S> : any entity of syntactical category S
- <S>* : suite of 0, 1 or more entity of category S, separated by the token |
- *text*, *typeface* : a character string (*text* may contain escape characters, e.g. quote \", new line \n, tab \t, backslash \\)
- *id* : entire number or a character string
- *x*, *y*, *width*, *height*, *helpId*, *pointsize* : entire numbers
- *style* : window flag beginning with WS_ or a dialog box flag beginning with DS_

- *ex_style* : extended window style flag (beginning with WS_EX_)
- *language* : primary language identifier flag beginning with LANG_
- *sublanguage* : sublanguage identifier flag beginning with SUBLANG_
- A ::= B : A is defined by B
- a b : a followed by b
- a & b : a and b (note that this meta-language symbol is different from the language token | used to combine several styles)
- val(A) : value of the attribute A
- upper(A) : converts the text A in upper case.
- [A] : A may occur only in a extended dialog box template (that is, a DIALOGEX resource and not a DIALOG resource)
- *control_def* : may contain any combination of control definitions (one by line)

As noticed in the examples, a numeric value can be specified in a resource script file instead of style flag identifiers. When two or more style flags are numerically represented in the same group, the values are added to form only one number. The table show also the hexadecimal number associated with each flag.

class diagram (logic name) ↔		resource script file (physic name)
		<p>dialog box template :</p> <pre> <id> DIALOG[EX] <x>, <y>, <width>, <height> [, <helpId>] STYLE <style>* CAPTION "<text>" [EXSTYLE <ex_style>*] LANGUAGE <language>, <sublanguage> FONT <pointsize>, "<typeface>" [, <weight>, <italic>] { <control_def> } </pre>
<p><u>Class</u> DIALOG</p> <p><u>Aggregation relationship</u> n ≥ 1 is the number of relationship instances in which the instance of DIALOG participates</p> <p><u>Attibutes</u> DlgID Extended = true X Y Width Height HelpID ≠ -1 3Dlook = true AbsoluteAlignment = true Center = true</p>	<p>DIALOG</p> <p>{ } and n ≥ 1 is the number of lines in <control_def></p> <pre> <id> ::= val(DlgID) DIALOGEX ³⁶ <x> ::= val(X) <y> ::= val(Y) <width> ::= val(Width) <height> ::= val(Height) ³⁷ <helpId> ::= val(HelpID) <style> ::= DS_3DLOOK <style> ::= DS_ABSALIGN <style> ::= DS_CENTER </pre>	<p><u>Flag value</u></p> <pre> 0x00000004 0x00000001 0x00000800 </pre>

³⁶ And additional information can be found between hooks ('[...]') in the dialog box template.

³⁷ The measurements in a dialog box template are specified in dialog template units, not in screen units (pixels). The system uses the average character width of the dialog box font to calculate the position and dimensions of the dialog box.

³⁸ For example, Language=french and Sublanguage=belgian gives SUBLANG_FRENCH_BELGIAN.

CenterMouse = true ContextHelp = true DialogsControl = true FixedSys = true LocalEdit = true ModalFrame = true NoFailCreate = true NoIdleMessage = true SetForeGroud = true SysModal = true val(Language) not NULL val(Sublanguage) not NULL SetFont = true FontName FontSize Italic = true Italic = false Weight ≠ -1 ShellFont = true	<style> ::= DS_CENTERMOUSE <style> ::= DS_CONTEXTHELP <style> ::= DS_CONTROL <style> ::= DS_FIXEDSYS <style> ::= DS_LOCALEDIT <style> ::= DS_MODALFRAME <style> ::= DS_NOFAILCREATE <style> ::= DS_NOIDLEMSG <style> ::= DS_SETFOREGROUND <style> ::= DS_SYSMODAL <language> ::= LANG_upper(val(Language)) ³⁸ <sublanguage> ::= SUBLANG_<language>_upper(val(Sublanguage)) <style> ::= DS_SETFONT <typeface> ::= val(FontName) <pointsize> ::= val(FontSize) <italic> ::= TRUE <italic> ::= FALSE <weight> ::= val(Weight) <style> ::= DS_SHELLFONT	0x00001000 0x00002000 0x00000400 0x00000008 0x00000020 0x00000080 0x00000010 0x00000100 0x00000200 0x00000002 0x00000040 0x00000048
Other attributes can be inherited from the WINDOW class (in addition to the attribute GeneralStyle always set to popup)		
<u>Class</u> WINDOW <u>Attributes</u> Text GeneralStyle = popup GeneralStyle = overlapped GeneralStyle = child Border = true Caption = true & Border = true ClipChildren = true ClipSimblings = true Disabled = true DialogFrame = true & Caption = false Group = true TabStop = true HorizontalScrollBar = true MaximizeBox = true Maximized = true MinimizeBox = true Minimized = true SystemMenu = true ThickFrame = true VerticalScrollBar = true Visible = true ClientEdge = true DialogModalFrame = true StaticEdge WindowEdge = true AcceptFiles = true AppWindow = true ControlParent = true ExContextHelp = true Layered = true LayoutRTL = true Left = true LeftScrollBar = true LTRReading = true MDIChild = true NoActivate = true NoInheritLayout = true NoParentNotify = true Right = true RightScrollBar = true RTLReading = true ToolWindow = true TopMost = true Transparent = true	<text> ::= val(Text) ³⁹ <style> ::= WS_POPUP <style> ::= WS_OVERLAPPED <style> ::= WS_CHILD <style> ::= WS_BORDER <style> ::= WS_CAPTION ⁴⁰ <style> ::= WS_CLIPCHILDREN <style> ::= WS_CLIPSIMBLINGS <style> ::= WS_DISABLED <style> ::= WS_DLGFRAME <style> ::= WS_GROUP <style> ::= WS_TABSTOP <style> ::= WS_HSCROLL <style> ::= WS_MAXIMIZEBOX <style> ::= WS_MAXIMIZE <style> ::= WS_MINIMIZEBOX <style> ::= WS_MINIMIZE <style> ::= WS_SYSTEMMENU <style> ::= WS_THICKFRAME <style> ::= WS_VSCROLL <style> ::= WS_VISIBLE ⁴¹ <ex_style> ::= WS_EX_CLIENTEDGE <ex_style> ::= WS_EX_DLGMODALFRAME <ex_style> ::= WS_EX_STATICEDGE <ex_style> ::= WS_EX_WINDOWEDGE <ex_style> ::= WS_EX_ACCEPTFILES <ex_style> ::= WS_EX_APPWINDOW <ex_style> ::= WS_EX_CONTROLPARENT <ex_style> ::= WS_EX_CONTEXTHELP <ex_style> ::= WS_EX_LAYERED <ex_style> ::= WS_EX_LAYOUTRTL <ex_style> ::= WS_EX_LEFT <ex_style> ::= WS_EX_LEFTSCROLLBAR <ex_style> ::= WS_EX_LTRREADING <ex_style> ::= WS_EX_MDICHILD <ex_style> ::= WS_EX_NOACTIVATE <ex_style> ::= WS_EX_NOINHERITLAYOUT <ex_style> ::= WS_EX_NOPARENTNOTIFY <ex_style> ::= WS_EX_RIGHT <ex_style> ::= WS_EX_RIGHTSCROLLBAR <ex_style> ::= WS_EX_RTLREADING <ex_style> ::= WS_EX_TOOLWINDOW <ex_style> ::= WS_EX_TOPMOST ⁴² <ex_style> ::= WS_EX_TRANSPARENT	0x80000000 0x00000000 0x40000000 0x00800000 0x00C00000 0x02000000 0x04000000 0x08000000 0x00400000 0x00020000 0x00010000 0x00100000 0x00010000 0x01000000 0x00020000 0x20000000 0x00080000 0x00040000 0x00200000 0x10000000 0x00000200 0x00000001 0x00020000 0x00000100 0x00000010 0x00040000 0x00010000 0x00000400 0x00080000 0x00400000 0x00000000 0x00004000 0x00000000 0x00000040 0x08000000 0x00100000 0x00000004 0x00001000 0x00000000 0x00002000 0x00000080 0x00000008 0x00000020

³⁹ For a dialog box, the line CAPTION "..." will place that string in the caption of the dialog only if the flag WS_CAPTION is present.

⁴⁰ The value of this flag includes also the WS_BORDER style.

Remarks:

- When a flag is not specified, the corresponding attribute of type Boolean is set to false.
- The flags by default are also denoted in the table. They correspond to a null value (in bold in the flag value column). When nothing else is specified, the presence or not of these flags makes no difference since they involve any additional value to the style (and in any case the associated attribute in the left column has to take the appropriate value).
- Some flags in the table have synonyms: WS_CHILD = WS_CHILDWINDOW, WS_OVERLAPPED = WS_TILED, WS_MINIMIZE = WS_ICONIC and WS_SIZEBOX = WS_THICKFRAME
- The definition of WS_DLGFAME was curiously enough: a window with this flag (with a thicker frame) cannot have a title bar.⁴³ The problem was that this flag is sometimes specified for a dialog box having still a title bar, as in the resource file given by Resource Tuner associated to my first example (the 'Find' dialog box) :
STYLE DS_SETFONT | DS_MODALFRAME | DS_3DLOOK | WS_POPUPWINDOW | WS_DLGFAME
As we will see in the shortcut notations (section 4.3.2), the flag WS_POPUPWINDOW includes WS_BORDER. With the numerical values, we can see that WS_CAPTION set the bits at positions 23 and 24 of to 1, whereas WS_DLGFAME and WS_BORDER set only the bit at position 23 and 24 respectively to 1 (the three flags are grouped in two bits). So, if the two last flag are specified for a dialog box, it's as if the first flag was specified. In fact, WS_DLGFAME *implies* no title bar.
- The numerical value of DS_SHELLFONT⁴⁴ is equal to DS_FIXEDSYS | DS_SETFONT. The value of DS_SHELLFONT was surely chosen so that older operating systems would accept the flag while nevertheless ignoring it. This allowed people to write a single program that got the Windows XP look when running on Windows XP and got a classic look when running on older systems. Older systems accepted the flag since it was indeed a valid flag, but they also ignored it because the DS_SETFONT flag takes precedence.
- For the font, identifiers are generally specified in the resource script file instead of particular numeric values.

<i>Value of Weight:</i>	<i>Windows identifiers:</i>
0	FW_DONTCARE
100	FW_THIN
200	FW_EXTRALIGHT or FW_ULTRALIGHT

⁴¹ This flag is not required for a modal dialog box. If not present, the system will set it even so visible. For a control, if this flag is not present, the child window will not be displayed until the function *ShowWindow* is called with SW_SHOWNORMAL as a parameter (needless if present).

⁴² The flag DS_SYSMODAL is also possible. It is obsolete but included for compatibility with 16-bits versions of Windows. When specified the system creates the dialog box with the WS_EX_TOPMOST style.

⁴³ And also the definition of WS_CAPTION (giving a title bar): this flag includes WS_BORDER (a thin-line border).

⁴⁴ Windows 2000 or later uses a different system font than Windows NT 4.0 and Windows 95/98. To have an application that use the system font no matter which system it is running on, DS_SHELLFONT can be specified. The system then maps this typeface such that the dialog box uses the Tahoma font on Windows 2000 and the MS Sans Serif font on earlier systems.

300	FW_LIGHT
400	FW_NORMAL or FW_REGULAR
500	FW_MEDIUM
600	FW_SEMIBOLD or FW_DEMIBOLD
700	FW_BOLD
800	FW_EXTRABOLD or FW_ULTRABOLD
900	FW_HEAVY or FW_BLACK

- When no style are specified for a dialog box, the default style is WS_POPUP | WS_BORDER | WS_SYSMENU.

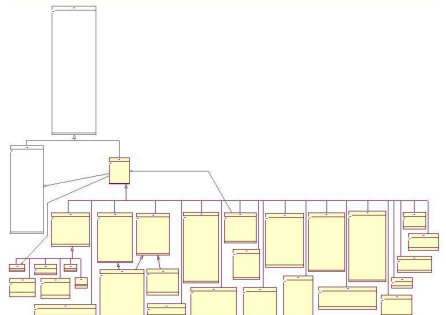
4.3.1.2 Controls definition

Inside the dialog box template, the part between { and } defines a set of controls that a dialog box contains. <control_def> is a suite of 0, 1 or more text lines defining a child window control and having this syntax:

CONTROL "<text>",<id>,<class>,<style>*,<x>,<y>,<width>,<height>[,<ex_style>*,<helpID>]

Meta-language (continuation) :

- *style* : is now a window style flag (beginning with WS_) or a child window control style flag (beginning with ES_, BS_, SS_, LBS_, SBS_ or CBS_)
- *class* : a predefined window class⁴⁵

	<p>dialog box template (continuation) :</p> <p>CONTROL "<text>", <id>, <class>, <style>*, <x>, <y>, <width>, <height> [,<ex_style>*, <helpID>]</p>	
<p><u>Class</u> CONTROL</p> <p><u>Attributes</u> CtrlID X Y Width Height HelpID ≠ -1 Position</p>	<p>CONTROL</p> <p><id> ::= val(CtrlID) <x> ::= val(X) <y> ::= val(Y) <width> ::= val(Width) <height> ::= val(Height) <helpID> ::= val(HelpID) control specified at line val(Position) of <control_body></p>	<p><u>Value</u></p>
<p><u>Class</u> BUTTON</p> <p><u>Attributes</u> VerticalAlignment = top VerticalAlignment = center VerticalAlignment = bottom HorizontalAlignment = left HorizontalAlignment = center HorizontalAlignment = right Flat = true Multiline = true</p>	<p><class> ::= BUTTON</p> <p><style> ::= BS_TOP <style> ::= BS_VCENTER <style> ::= BS_BOTTOM <style> ::= BS_LEFT <style> ::= BS_CENTER <style> ::= BS_RIGHT <style> ::= BS_FLAT <style> ::= BS_MULTILINE</p>	<p>0x0000400 0x0000C00 0x0000800 0x0000100 0x0000300 0x0000200 0x0000800 0x0000200</p>

⁴⁵ Notice from the examples that *class* can occur between quotes (like "BUTTON") in the resource script file (this is the case with Resource Builder), and also in lower case (like "button").

Notify = true	<style> ::= BS_NOTIFY	0x00004000
<u>Class</u> GROUPBOX	<style> ::= BS_GROUPBOX	0x00000007
<u>Class</u> RADIOBUTTON <u>Attributes</u> Auto = false Auto = true RightButton = true PushLike = true	<style> ::= BS_RADIOBUTTON <style> ::= BS_AUTORADIOBUTTON <style> ::= BS_RIGHTBUTTON <style> ::= BS_PUSHLIKE	0x00000004 0x00000009 0x00000020 0x00001000
<u>Class</u> PUSHBUTTON <u>Attributes</u> Default = false Default = true Content = text Content = bitmap Content = icon	<style> ::= BS_PUSHBUTTON <style> ::= BS_DEFPUSHBUTTON <style> ::= BS_TEXT ⁴⁶ <style> ::= BS_BITMAP <style> ::= BS_ICON	0x00000000 0x00000001 0x00000000 0x00000080 0x00000040
<u>Class</u> CHECKBOX <u>Attributes</u> Auto = false & 3State = false Auto = true & 3State = false Auto = false & 3State = true Auto = true & 3State = true RightButton = true PushLike = true	<style> ::= BS_CHECKBOX <style> ::= BS_AUTOCHECKBOX <style> ::= BS_3STATE <style> ::= BS_AUTO3STATE <style> ::= BS_RIGHTBUTTON <style> ::= BS_PUSHLIKE	0x00000002 0x00000003 0x00000005 0x00000006 0x00000020 0x00001000
<u>Class</u> PUSHBOX	<style> ::= BS_PUSHBOX	
<u>Class</u> CUSTOMBUTTON	<style> ::= BS_OWNERDRAWN ⁴⁷	0x0000000B
<u>Class</u> SCROLLBAR <u>Attributes</u> Type = horizontal Type = vertical Type = sizeBox Type = sizeGrip Alignment = left Alignment = right Alignment = top Alignment = bottom Alignment = bottomRight Alignment = topLeft	<class> ::= SCROLLBAR <style> ::= SBS_HORZ <style> ::= SBS_VERT <style> ::= SBS_SIZEBOX <style> ::= SBS_SIZEGRIP <style> ::= SBS_LEFTALIGN <style> ::= SBS_RIGHTALIGN <style> ::= SBS_TOPALIGN <style> ::= SBS_BOTTOMALIGN <style> ::= SBS_SIZEBOXBOTTOMRIGHTALIGN <style> ::= SBS_SIZEBOXTOPLEFTALIGN	0x00000000 0x00000001 0x00000008 0x00000010 0x00000002 0x00000004 0x00000002 0x00000004 0x00000004 0x00000004 0x00000002
<u>Class</u> LISTBOX <u>Attributes</u> Sorted = true MultipleSelection = true DisableNoScroll = true ExtendedSelection = true Multicolumn = true NoSelection = true Notify = true OwnerDrawFixed = true OwnerDrawVariable = true HasStrings = true NoIntegralHeight = true	<class> ::= LISTBOX <style> ::= LBS_SORT <style> ::= LBS_MULTIPLESEL <style> ::= LBS_DISABLENOSCROLL <style> ::= LBS_EXTENDEDSEL <style> ::= LBS_MULTICOLUMN <style> ::= LBS_NOSEL <style> ::= LBS_NOTIFY <style> ::= LBS_OWNERDRAWFIXED <style> ::= LBS_OWNERDRAWVARIABLE <style> ::= LBS_HASSTRINGS <style> ::= LBS_NOINTEGRALHEIGHT	0x00000002 0x00000008 0x00001000 0x00000800 0x00002000 0x00004000 0x00000001 0x00000010 0x00000020 0x00000040 0x00000100

⁴⁶ If fact, the BS_TEXT flag (even though it exists) is missing most of time. So Content = text when neither BS_BITMAP nor BS_ICON are present (push buttons have this style by default).

⁴⁷ BS_USERBUTTON is also possible but this flag obsolete. It is provided for compatibility with 16-bit versions of Windows. Applications should use BS_OWNERDRAWN instead. The value of this flag is 0x00000008.

<u>Attributes</u> Type = enhancedMetafile Type = ownerdraw Sunken = true Notify = true Type = text & TextStyle = left Type = text & TextStyle = right Type = text & TextStyle = center Type = text & TextStyle = simple Type = text & TextStyle = leftNoWordWrap NoPrefix = true EndEllipsis = true PathEllipsis = true WordEllipsis = true Type = frame & Frame = black Type = frame & Frame = gray Type = frame & Frame = white Type = frame & Rectangle = black Type = frame & Rectangle = gray Type = frame & Rectangle = white Type = frame & Etched = frame Type = frame & Etched = horizontal Type = frame & Etched = vertical Type = image & Icon = true Type = image & Bitmap = true CenterImage = true RealSizeImage = true RightJustify = true	<style> ::= SS_ENHMETAFILE <style> ::= SS_OWNERDRAW ⁴⁸ <style> ::= SS_SUNKEN <style> ::= SS_NOTIFY <style> ::= SS_LEFT <style> ::= SS_RIGHT <style> ::= SS_CENTER <style> ::= SS_SIMPLE <style> ::= SS_LEFTNOWORDWRAP <style> ::= SS_NOPREFIX <style> ::= SS_ENDELLIPSIS <style> ::= SS_PATHELLIPSIS <style> ::= SS_WORDELLIPSIS <style> ::= SS_BLACKFRAME <style> ::= SS_GRAYFRAME <style> ::= SS_WHITEFRAME <style> ::= SS_BLACKRECT <style> ::= SS_GRAYRECT <style> ::= SS_WHITERECT <style> ::= SS_ETCHEDHORZ <style> ::= SS_ETCHEDVERT <style> ::= SS_ETCHEDFRAME <style> ::= SS_ICON ⁴⁹ <style> ::= SS_BITMAP ⁵⁰ <style> ::= SS_CENTERIMAGE <style> ::= SS_REALSIZEIMAGE <style> ::= SS_RIGHTJUST	0x0000000F 0x0000000D 0x00001000 0x00000100 0x00000000 0x00000002 0x00000001 0x0000000B 0x0000000C 0x00000080 0x00004000 0x00008000 0x0000C000 0x00000007 0x00000008 0x00000006 0x00000004 0x00000005 0x00000009 0x00000010 0x00000011 0x00000012 0x00000003 0x0000000E 0x00000200 0x00000800 0x00000400
<u>Class</u> TRACKBAR <u>Attributes</u> AutoTicks = true DownsLeft = true EnableSelectionRange = true FixedLength = true NoThumb = true NoTicks = true Orientation = horizontal Orientation = vertical Reversed = true TickMarks = bottom TickMarks = top TickMarks = left TickMarks = right TickMarks = both Tooltips = Boolean	<class> ::= "msctls_trackbar32" <style> ::= TBS_AUTOTICKS <style> ::= TBS_DOWNISLEFT <style> ::= TBS_ENABLESELECTIONRANGE <style> ::= TBS_FIXEDLENGTH <style> ::= TBS_NOTHUMB <style> ::= TBS_NOTICKS <style> ::= TBS_HORZ <style> ::= TBS_VERT <style> ::= TBS_REVERSED <style> ::= TBS_BOTTOM <style> ::= TBS_TOP <style> ::= TBS_LEFT <style> ::= TBS_RIGHT <style> ::= TBS_BOTH <style> ::= TBS_TOOLTIPS	0x00000001 0x00000400 0x00000020 0x00000040 0x00000080 0x00000010 0x00000000 0x00000002 0x00000200 0x00000000 0x00000004 0x00000004 0x00000000 0x00000008 0x00000100
<u>Class</u> UPDOWN <u>Attributes</u> Alignment = left Alignment = right ArrowKeys = true AutoBuddy = true Horizontal = true HotTrack = true NoThousands = true SetBuddyInt = true Wrap = true	<class> ::= "msctls_updown32" <style> ::= UDS_ALIGNLEFT <style> ::= UDS_ALIGNRIGHT <style> ::= UDS_ARROWKEYS <style> ::= UDS_AUTOBUDDY <style> ::= UDS_HORZ <style> ::= UDS_HOTTRACK <style> ::= UDS_NOTHOUSANDS <style> ::= UDS_SETBUDDYINT <style> ::= UDS_WRAP	0x00000008 0x00000004 0x00000020 0x00000010 0x00000040 0x00000100 0x00000080 0x00000002 0x00000001

⁴⁸ SS_USERBUTTON (value 0x0000000A) is also possible but this flag is obsolete.

⁴⁹ For this type of control, a remark must be mentioned with <text> ::= val(Text). When defining an SS_ICON static control in the template, the icon must be defined elsewhere in the resource file. As I noted earlier the icon resource name must be specified as the text for the control. When the icon's name is a number, I notice that there are no more quotes in the template. For example, we can have the line **CONTROL 10,-1,STATIC,WS_CHILD|WS_VISIBLE|SS_ICON|WS_GROUP,5,10,15,20** and elsewhere in the resource file **10 Icon "MyIcon.ico"**.

⁵⁰ Same remark than for an icon.

<p><u>Class</u> HEADER</p> <p><u>Attributes</u> Buttons = true DragDrop = true FilterBar = true Flat = true FullDrag = true Hidden = true Horizontal = true HotTrack = true</p>	<p><class> ::= "SysHeader32"</p> <p><style> ::= HDS_BUTTONS <style> ::= HDS_DRAGDROP <style> ::= HDS_FLAT <style> ::= HDS_FILTERBAR <style> ::= HDS_FULLDRAG <style> ::= HDS_HIDDEN <style> ::= HDS_HORZ <style> ::= HDS_HOTTRACK</p>	<p>0x00000002 0x00000040</p> <p>0x00000080 0x00000080</p> <p>0x00000004</p>
<p><u>Class</u> REBAR</p> <p><u>Attributes</u> AutoSize = true BandBorders = true DoubleClickToggle = true FixedOrder = true NoDivider = true RegisterDrop = true VarHeight = true VerticalGripper = true Vertical = true</p>	<p><class> ::= "ReBarWindow32"</p> <p><style> ::= RBS_AUTOSIZE <style> ::= RBS_BANDBORDERS <style> ::= RBS_DBLCLKTOGGLE <style> ::= RBS_FIXEDORDER <style> ::= CCS_NODIVIDER <style> ::= RBS_REGISTERDROP <style> ::= RBS_VARHEIGHT <style> ::= RBS_VERTICALGRIPPER <style> ::= CCS_VERT</p>	<p>0x00002000 0x00000400 0x00008000 0x00008000 0x00000800 0x00000400 0x00001000 0x00002000 0x00004000 0x00000800</p>
<p><u>Class</u> TREEVIEW</p> <p><u>Attributes</u> CheckBoxes = true DisableDragDrop = true EditLabels = true FullRowSelect = true HasButtons = true HasLines = true InfoType = true LinesAtRoot = true NoHorizontalScroll = true NonevenHeight = true NoScroll = true NoTooltips = true RightToLeftReading = true ShowSelectionAlways = true SingleExpand = true TrackSelect = true</p>	<p><class> ::= "SysTreeView32"</p> <p><style> ::= TVS_CHECKBOXES <style> ::= TVS_DISABLEDRAHDROP <style> ::= TVS_EDITLABELS <style> ::= TVS_FULLROWSELECT <style> ::= TVS_HASBUTTONS <style> ::= TVS_HASLINES <style> ::= TVS_INFOTIP <style> ::= TVS_LINESATROOT <style> ::= TVS_NOHSCROLL <style> ::= TVS_NONEVENHEIGHT <style> ::= TVS_NOSCROLL <style> ::= TVS_NOTOOLTIPS <style> ::= TVS_RTLLREADING <style> ::= TVS_SHOWSELALWAYS <style> ::= TVS_SINGLEEXPAND <style> ::= TVS_TRACKSELECT</p>	<p>0x00000100 0x00000010 0x00000008 0x00001000 0x00000001 0x00000002</p> <p>0x00000004</p> <p>0x00004000 0x00002000 0x00000800 0x00000040 0x00000020 0x00000400 0x00000200</p>
<p><u>Class</u> TOOLBAR</p> <p><u>Attributes</u> ALTDrag = true CustomErase = true Flat = true Transparent = true List = true RegisterDrop = true ToolTips = true WrapAble = true NoResize = true NoParentAlign = true Bottom = true NoDivider = true Adjustable = true</p>	<p><class> ::= "ToolbarWindow32"</p> <p><style> ::= TBSTYLE_ALTDRAW <style> ::= TBSTYLE_CUSTOMERASE <style> ::= TBSTYLE_FLAT <style> ::= TBSTYLE_TRANSPARENT <style> ::= TBSTYLE_LIST <style> ::= TBSTYLE_REGISTERDROP <style> ::= TBSTYLE_TOOLTIPS <style> ::= TBSTYLE_WRAPABLE <style> ::= CCS_NORESIZE <style> ::= CCS_NOPARENTALIGN <style> ::= CCS_BOTTOM <style> ::= CCS_NODIVIDER <style> ::= CCS_ADJUSTABLE</p>	<p>0x00000400 0x00002000 0x00008000 0x00008000 0x00001000 0x00004000 0x00000100 0x00002000 0x00000004 0x00000008 0x00000003 0x00000040 0x00000020</p>
<p><u>Class</u> TAB</p> <p><u>Attributes</u> Type = tabs Type = buttons Bottom = true FixedWidth = true FlatButtons = true</p>	<p><class> ::= "SysTabControl32"</p> <p><style> ::= TCS_TABS <style> ::= TCS_BUTTONS <style> ::= TCS_BOTTOM <style> ::= TCS_FIXEDWIDTH <style> ::= TCS_FLATBUTTONS</p>	<p>0x00000000 0x00000100 0x00000002 0x00000400 0x00000008</p>

<p>FocusNever = true FocusOnButtonDown = true ForceLeft = icon ForceLeft = label HotTrack = true Multiline = true Multiline = false MultiSelection = true OwnerDrawFixed = true RaggedRight = true Right = true RightJustify = true ScrollOpposite = true ToolTips = true Vertical = true</p>	<pre><style> ::= TCS_FOCUSNEVER <style> ::= TCS_FOCUSONBUTTONDOWN <style> ::= TCS_FORCEICONLEFT <style> ::= TCS_FORCELABELLEFT <style> ::= TCS_HOTTRACK <style> ::= TCS_MULTILINE <style> ::= TCS_SINGLELINE <style> ::= TCS_MULTISELECT <style> ::= TCS_OWNERDRAWFIXED <style> ::= TCS_RAGGEDRIGHT <style> ::= TCS_RIGHT <style> ::= TCS_RIGHTJUSTIFY <style> ::= TCS_SCROLLOPPOSITE <style> ::= TCS_TOOLTIPS <style> ::= TCS_VERTICAL</pre>	<pre>0x00008000 0x00001000 0x00000010 0x00000020 0x00000040 0x00000200 0x00000000 0x00000004 0x00002000 0x00000800 0x00000002 0x00000000 0x00000001 0x00004000 0x00000080</pre>
<p><u>Class</u> DATETIMEPICKER <u>Attributes</u> AppCanParse = true Format = longDate Format = shortDate Format = shortDateCentury Format = time RightAlign = true ShowNone = true UpDown = true</p>	<pre><class> ::= "SysDateTimePick32" <style> ::= DTS_APPCANPARSE <style> ::= DTS_LONGDATEFORMAT <style> ::= DTS_SHORTDATEFORMAT <style> ::= DTS_SHORTDATECENTURYFORMAT <style> ::= DTS_TIMEFORMAT <style> ::= DTS_RIGHTALIGN <style> ::= DTS_SHOWNONE <style> ::= DTS_UPDOWN</pre>	<pre>0x00000010 0x00000004 0x00000000 0x0000000C 0x00000009 0x00000020 0x00000002 0x00000001</pre>
<p><u>Class</u> LISTVIEW <u>Attributes</u> View = icon View = smallIcon View = list View = report Alignment = left Alignment = top EditLabels = true OwnerData = true ShareImageLists = true ShowSelectionAlways = true SingleSelect = true SortAscending = true SortDescending = true CheckBoxes = true OneClickActivate = true TrackSelect = true TwoClickActivate = true AutoArrange = true NoLabelWrap = true NoScroll = true NoColumnHeader = true NoSortHeader = true OwnerDrawFixed = true FullRowSelect = true GridLines = true HeaderDragDrop = true SubItemImages = true</p>	<pre><class> ::= "SysListView32" <style> ::= LVS_ICON <style> ::= LVS_SMALLICON <style> ::= LVS_LIST <style> ::= LVS_REPORT <style> ::= LVS_ALIGNLEFT <style> ::= LVS_ALIGNTOP <style> ::= LVS_EDITLABELS <style> ::= LVS_OWNERDATA <style> ::= LVS_SHAREIMAGELISTS <style> ::= LVS_SHOWSELALWAYS <style> ::= LVS_SINGLESEL <style> ::= LVS_SORTASCENDING <style> ::= LVS_SORTDESCENDING <ex_style> ::= LVS_EX_CHECKBOXES <ex_style> ::= LVS_EX_ONECLICKACTIVATE <ex_style> ::= LVS_EX_TRACKSELECT <ex_style> ::= LVS_EX_TWOCCLICKACTIVATE <style> ::= LVS_AUTOARRANGE <style> ::= LVS_NOLABELWRAP <style> ::= LVS_NOSCROLL <style> ::= LVS_NOCOLUMNHEADER <style> ::= LVS NOSORTHEADER <style> ::= LVS_OWNERDRAWFIXED <ex_style> ::= LVS_EX_FULLROWSELECT <ex_style> ::= LVS_EX_GRIDLINES <ex_style> ::= LVS_EX_HEADERDRAGDROP <ex_style> ::= LVS_EX_SUBITEMIMAGES</pre>	<pre>0x00000000 0x00000002 0x00000003 0x00000001 0x00000800 0x00000000 0x00000200 0x00001000 0x00000040 0x00000008 0x00000004 0x00000010 0x00000020 0x00000004 0x00000040 0x00000800 0x00000100 0x00000080 0x00002000 0x00004000 0x00008000 0x00004000 0x00000020 0x00000001 0x00000010 0x00000002</pre>
<p><u>Class</u> MONTHCALENDAR <u>Attributes</u> DayState = true Multiselection = true NoToday = true NoTodayCircle = true WeekNumbers = true</p>	<pre><class> ::= "SysMonthCal32" <style> ::= MCS_DAYSTATE <style> ::= MCS_MULTISELECT <style> ::= MCS_NOTODAY <style> ::= MCS_NOTODAYCIRCLE <style> ::= MCS_WEEKNUMBERS</pre>	<pre>0x00000001 0x00000002 0x00000008 0x00000010 0x00000004</pre>
<p><u>Class</u> ANIMATION</p>	<pre><class> ::= "SysAnimate32"</pre>	

<u>Attributes</u> AutoPlay = true Center = true Transparent = true	<style> ::= ACS_AUTOPLAY <style> ::= ACS_CENTER <style> ::= ACS_TRANSPARENT	0x00000004 0x00000001 0x00000002
<u>Class</u> PAGER <u>Attributes</u> AutoScroll = true DragAndDrop = true Direction = horizontal Direction = vertical	<class> ::= "SysPager" <style> ::= PGS_AUTOSCROLL <style> ::= PGS_DRAGNDROP <style> ::= PGS_HORZ <style> ::= PGS_VERT	 0x00000002 0x00000004 0x00000001 0x00000000
<u>Class</u> STATUTBAR <u>Attributes</u> SizeGrip = true Tooltips = true Top = true	<class> ::= "msctls_statusbar32" <style> ::= SBARS_SIZEGRIP <style> ::= SBT_TOOLTIPS <style> ::= CCS_TOP	 0x00000100 0x00000800 0x00000001
<u>Class</u> CUSTOMCONTROL <u>Attribute</u> ClassName	<class> ::= "val(ClassName)" ⁵¹	

Note: the button style flag BS_RIGHTBUTTON is synonymous with BS_LEFTTEXT.

4.3.2 Shortcut notations in the dialog box template

Shortcut notations can be used to define child window controls in the dialog box template. As we can see in the first example, other identifiers than CONTROL are used in the dialog box template. They imply a predefined window class and some style flags that do not have to be specified. For example, LTEXT indicates that the class of the child window control is STATIC and that the style is WS_CHILD | SS_LEFT | WS_VISIBLE | WS_GROUP. Resource Tuner and Restorator use shortcuts in the resource file they generate, whereas Resource Builder and Resource Hacker use the full notation.

The following table links the control types which can be specified and their equivalent notation (window class and style flags):

<control_type>	<class>	<style>*
PUSHBUTTON	BUTTON	WS_CHILD WS_VISIBLE BS_PUSHBUTTON WS_TABSTOP
DEFPUSHBUTTON	BUTTON	WS_CHILD WS_VISIBLE BS_DEFPUSHBUTTON WS_TABSTOP
CHECKBOX	BUTTON	WS_CHILD WS_VISIBLE BS_CHECKBOX WS_TABSTOP
AUTOCHECKBOX	BUTTON	WS_CHILD WS_VISIBLE BS_AUTOCHECKBOX WS_TABSTOP
RADIOBUTTON	BUTTON	WS_CHILD WS_VISIBLE BS_RADIOBUTTON WS_TABSTOP
AUTORADIOBUTTON	BUTTON	WS_CHILD WS_VISIBLE BS_AUTORADIOBUTTON WS_TABSTOP
GROUPBOX	BUTTON	WS_CHILD WS_VISIBLE BS_GROUPBOX
LTEXT	STATIC	WS_CHILD WS_VISIBLE SS_LEFT WS_GROUP
CTEXT	STATIC	WS_CHILD WS_VISIBLE SS_CENTER WS_GROUP
RTEXT	STATIC	WS_CHILD WS_VISIBLE SS_RIGHT WS_GROUP
ICON	STATIC	WS_CHILD WS_VISIBLE SS_ICON WS_GROUP
EDITTEXT	EDIT	WS_CHILD WS_VISIBLE ES_LEFT WS_BORDER WS_TABSTOP
LISTBOX	LISTBOX	WS_CHILD WS_VISIBLE LBS_NOTIFY WS_BORDER WS_VSCROLL
COMBOBOX	COMBOBOX	WS_CHILD WS_VISIBLE CBS_SIMPLE WS_TABSTOP
SCROLLBAR	SCROLLBAR	WS_CHILD WS_VISIBLE SBS_HORZ

⁵¹ ex: `CONTROL "&Find Next",1,"ellipticalPushButton",WS_CHILD|WS_VISIBLE|WS_GROUP|WS_TABSTOP,205,5,65,14`

Notice that many of the controls that interact with the user include `WS_TABSTOP` as a default. However, radio buttons other than the first of each radio group lack `WS_TABSTOP` by default. The static controls include `WS_GROUP` by default, which conveniently mark the end of a group

For all these control types, the format to define the control is

```
<control_type> "<text>" ,<id> ,<x> ,<y> ,<width> ,<height> ,<style>* [ ,<ex_style>* ,<helpID> ]
```

except for the `EDITTEXT`, `SCROLLBAR`, `LISTBOX`, and `COMBOBOX` styles where the text field is excluded. The RC resource compiler understands the two notations.

So, the line from my first example `DEFPUSHBUTTON"&Find Next",1,205,5,65,14,WS_GROUP` is equivalent to `CONTROL"&Find Next",1,BUTTON,BS_DEFPUSHBUTTON|WS_CHILD|WS_VISIBLE|WS_GROUP|WS_TABSTOP,205,5,65,14`. These two syntaxes are encoded identically in the `.res` file and the `.exe` file. The second syntax is less convenient but more complete: with the identifiers, a push button can always be accessed with the `TAB` key for instance. The `NOT` keyword can occur in the template to precisely resolve this drawback : `DEFPUSHBUTTON"&Find Next",1,205,5,65,14,NOT WS_TABSTOP,WS_GROUP`.

The decompilers that use shortcut notations use naturally also the generic notation for the control of another class than those listed above. However, I've remarked that `WS_CHILD` and `WS_VISIBLE` are also implicit in their generic notation (these flags are never specified).

Some style flags can also be used as shortcut notations.

- `WS_EX_OVERLAPPEDWINDOW`
combines the `WS_EX_CLIENTEDGE` and `WS_EX_WINDOWEDGE` styles.
- `WS_EX_PALETTEWINDOW`
combines the `WS_EX_WINDOWEDGE` and `WS_EX_TOPMOST` styles.
- `WS_OVERLAPPEDWINDOW` or `WS_TILEDWINDOW`
combines the `WS_OVERLAPPED`, `WS_CAPTION`, `WS_SYSMENU`, `WS_THICKFRAME`, `WS_MINIMIZEBOX` and `WS_MAXIMIZEBOX` styles.
- `LBS_STANDARD`
combines the `LBS_NOTIFY`, `LBS_SORT` and `WS_BORDER` styles.
- `WS_POPUPWINDOW`
combines the `WS_BORDER`, `WS_POPUP` and `WS_SYSMENU` styles.

4.3.3 Resource of type menu

The resource script file can also contain a menu resource, which is a collection of information that defines the appearance and functions of an application menu. As you can see in the second example, the syntax is more intuitive:

```
<id> MENU  
LANGUAGE <language> , <sublanguage>  
{  
    <menu_body>  
}
```

menu_body may contain any combination of popup menu and menu item (one by line).

A popup menu has the following syntax: **POPUP** *<text>*, *<options>* { *<popup_body>* }
options is an optional list of one or more identifiers such INACTIVE, GRAYED or CHECKED (separated by commas or spaces). *popup_body* may contain any combination of menu item, separator or popup menu (one by line).

A menu item has the following syntax: **MENUITEM** *<text>*, *<id>*, *<options>*

A separator has the following syntax: **MENUITEM SEPARATOR**

It's easy to link the attributes from the class diagram with this syntax. Note that the Position attribute corresponds to the line where the item is defined in *<menu_body>* or *<popup_body>*.

4.3.4 Other types of resources

The other resources described in a resource script file are not helpful for the development of my plug-in. However, the resource editors can habitually extract the icon and bitmap from the executable and save them anywhere. Then, the user will be able to manually put a missing image in its GrafiXML project.

When accelerators are defined for menu items (key combinations), they should be found in the resource of type Accelerators. If I go back to my second example, there is this resource from Resource Hacker:

```
209 ACCELERATORS
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
{
    VK_N, 251, NOINVERT, CONTROL, VIRTKEY
    VK_F1, 610, NOINVERT, VIRTKEY
    VK_F12, 432, NOINVERT, ALT, VIRTKEY
    VK_F7, 250, NOINVERT, VIRTKEY
}
```

Only one menu item is matching: **MENUITEM** "Folders History...\tAlt+F12", 432. And there are often some curiosities. So, this type of resource will not be exploited. Anyway, this information should be in the text string of a menu item (if not, this lost of information is not really important since the user will not know the existence of the key combinations).

4.4 About the decompilation tools

I've not here to motivate the choice of a particularly tool by saying which one is the better to use or the more developed to edit resources for example. It's the produce .rc resource file which interests me.

We can see in appendix B fragments of resource files given by the different decompilers and extracted from the same executable. The tools decompiling a binary file give in general the same information. I can see that they cover all the existing style flags since a numeric value is still specified if a tool doesn't recognize a flag identifier in its implementation. It's the remaining value from a word of 32 bits stored in the binary file after extracting all the other recognized flags. The first difference in the generated files is the use of the generic or shortcut notation to define the controls. From those that use the generic notation, Resource Hacker seems more reliable than Resource Builder. Between the two other tools, I prefer Restorator than Resource Tuner. In the tests that I've made, the last gives sometimes oddities in the specification of the resources (see for example the frame that I've drawn in the specification of the sixth example).

The difference rests principally in the complete resource script file. In appendix A, we can see a file given by Resource Hacker. Note that the resources specifying the accelerators and version information are detailed in this file. Resource Builder includes in addition a text representation of binary files which are normally only referenced in the resource script file. There is how are specified an icon or a bitmap image (it's only a fragment, the icon resource for example have 511 lines of hexadecimal numbers...):

```
SCITE ICON
MOVEABLE PURE LOADONCALL DISCARDABLE
LANGUAGE LANG_ENGLISH, 1
BEGIN
'00 00 01 00 04 00 30 30 00 01 00 00 00 00 A8 0E '
'00 00 46 00 00 00 20 20 00 01 00 00 00 00 A8 08 '
'00 00 EE 0E 00 00 10 10 00 01 00 00 00 00 68 05 '
(...)
'00 01 C0 00 00 03 C0 00 00 03 E0 00 00 07 F0 00 '
'00 0F F8 00 00 1F FC 00 00 3F FE 00 00 7F FF 80 '
'01 FF FF F0 0F FF '
END

100 BITMAP
MOVEABLE PURE LOADONCALL DISCARDABLE
LANGUAGE LANG_ENGLISH, 1
BEGIN
'42 4D EE 00 00 00 00 00 00 76 00 00 00 28 00 '
'00 00 10 00 00 00 0F 00 00 00 01 00 04 00 00 00 '
'00 00 78 00 00 00 12 0B 00 00 12 0B 00 00 10 00 '
(...)
'FF FF FF FF F0 77 77 70 FF FF FF FF F0 77 77 70 '
'FF FF FF 00 00 77 77 70 FF FF FF 0F 07 77 77 70 '
'FF FF FF 00 77 77 77 70 00 00 00 07 77 77 '
END
```


These numbers should have perhaps been exploited to restore an image in my reverse engineering process. Icon and bitmap image (as cursors) are in reality a collection of bits in an array of any number of rows and columns (for a black and white icon for example, each bit corresponds to one display pixel). A deeper analysis to make the connection of an image and these numbers has not been undertaken in my thesis.

Restorator show how the resources appended to the executable are really structured. Only the text resources of type menu and dialog box are stored in the main .res file (the .rc file compiled). The other resources are defined (associated with an identifier) in this file but are stored in separated files. There is the resource file extracted from the same executable by Restorator (without the specifications of the menu and the dialog boxes):

```
// from file "C:\Program Files\wscite\SciTE.exe" last modified on 18/04/2005 15:40:52
// ----- resources of type Bitmap -----
100 Bitmap 100.bmp

// ----- resources of type Menu -----
(...)
// ----- resources of type Menu -----
(...)

// ----- resources of type Accelerator -----
ACCELS Accelerators ACCELS.txt

// ----- resources of type Icon -----
SCITE Icon SCITE.ico

// ----- resources of type Version -----
1 Version 1.res

// ----- resources of type 24 -----
1 24 1.txt
```

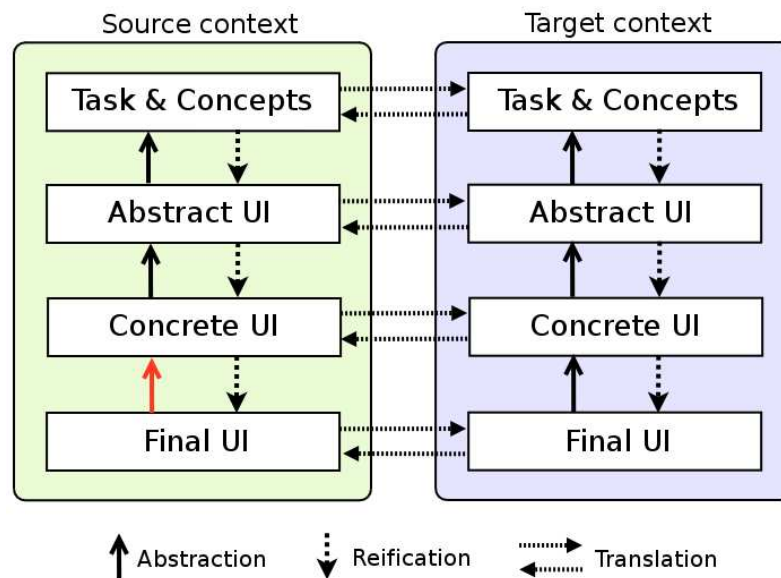
The resource 24 is a resource in the XML format. This tool adds also some comments. With Resource Tuner, each text resource is saved in a separated .rc file (one for each menu and dialog box for example). If we want save all the resources at once, the tool creates a repertory for each type of resource.

5 UsiXML

The first section of this chapter presents UsiXML (User Interface eXtensible Markup Language) and its levels of abstraction. The second section gives the UML diagram modelling the Concrete User Interface level. In the previous chapter I have seen the language used in a Windows resource file. The goal of the next chapter will be to make an abstraction from the Final User Interface level expressed in this language up to the Concrete User Interface level expressed in the UsiXML language.

5.1 Structure of UsiXML

UsiXML is then not a new language for UI implementation, but a User Interface Description Language (UIDL) based on XML. It specifies the multiple models involved in UI design which are structured according the layers of the **Cameleon**⁵² **reference framework**. To talk about the UsiXML language, this framework has first to be presented. The following illustrates the development process for two contexts of use. Four layers (development steps) compose the framework: Task and Concepts (T&C), Abstract User Interface (AUI), Concrete User Interface (CUI) and Final User Interface (FUI).

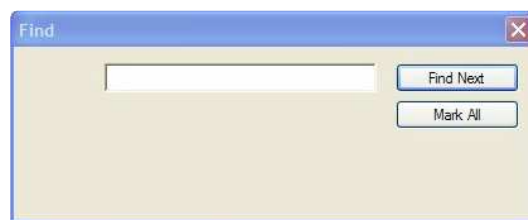


The Cameleon framework is intended to express the UI development life cycle for context-sensitive interactive applications. The context of use, related to the notion of task, determines the conception choices. It is defined by tree elements:

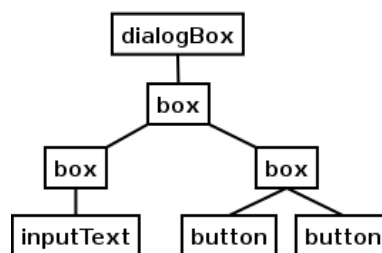
⁵² Context Aware Modelling for Enabling and Leveraging Effective interaction. For more information about this framework, see <http://giove.cnuce.cnr.it/cameleon/documents.html>.

- The interactive task will be realized by a particular *user* stereotype (or profile).⁵³
- This task will be performed on a particular *platform* (a set of devices and software which enables supporting the interactive task).⁵⁴
- The user is immersed in a physical *environment* when realizing its task on the platform.⁵⁵

The **FUI** is the UI produced at the last step of the reification process. It consists of the UI coded in any language (like Java, HTML or the language used in a resource script file) to be interpreted or compiled. It is then the operational UI running on a particular computing platform (like a PC, a Macintosh Mac OS, web terminal, a pocket PC, a mobile phone...). There is for instance the rendering of the dialog box from my first example that I have simplified:



The **CUI** level abstracts a FUI independently of any platform. The above push button control labelled “Find next” is for example abstracted into a button at the CUI level. A CUI can also be considered as a reification of an AUI concretizing the UI for a given context of use into Concrete Interaction Objects (CIOs) so as to define widgets layout and interface navigation. Widgets⁵⁶ are building blocks of the graphical UI each performing a specific function (e.g., text fields, buttons or check boxes). The interface is then composed of existing UI widgets with a set of properties (e.g., background colour, size, font) and a set of values associated with the properties (e.g., black, Tahoma, 8pt), but these widgets are independent of any particular toolkit:



The **AUI** level is assumed to abstract the CUI independently of any modality of interaction. In the example, it's a graphical interaction. But we could have also video-based interaction, vocal interaction (with eventually speech recognition), tactile interaction, haptic interaction (as vibrations in a joystick) or even soon olfactory interaction. In the other direction, an AUI

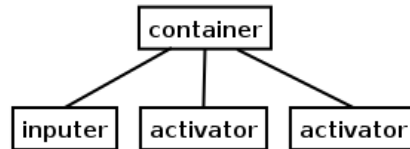
⁵³ For example, supposing a distributed application in a hospital which manages information about patients, a doctor in an ambulance using a pocket PC when detecting an urgent situation has a complex experience of analyse. The emergency ward can then be prepared to receive the patient.

⁵⁴ In the example, say an IPAQ with a large tactile screen usable with the fingers.

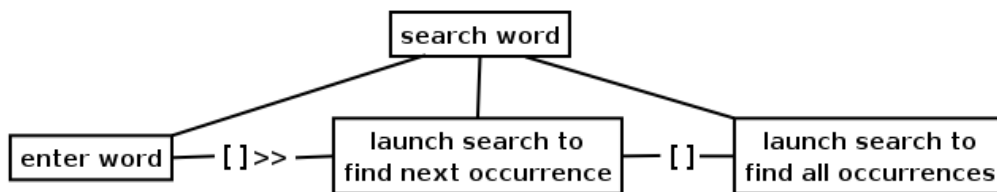
⁵⁵ In the example, the doctor is in a stressful environment.

⁵⁶ A widget evokes "window gadget".

express the rendering of the domain concepts and tasks independently of a modality. An element used in this level⁵⁷ can be then the abstraction of an existing widget, while representing a subtask defined at the upper level. There is how my simple example is in turn abstracted at the AUI level:



The **T&C** level describes the interactive tasks to be carried out by the end user (here, searching for a word in a text) and the domain concepts as they are required by these tasks to be performed. A task is typically hierarchically decomposed into sub-tasks to end up with actions (the leaves) which can be no longer decomposed and are carried out. The tasks are also ordered with temporal relationship. The same example is graphically represented below, where the `[]>>` operator indicates that sequence is needed and information is passed and the `[]` operator expresses the constraint of a choice between two tasks.



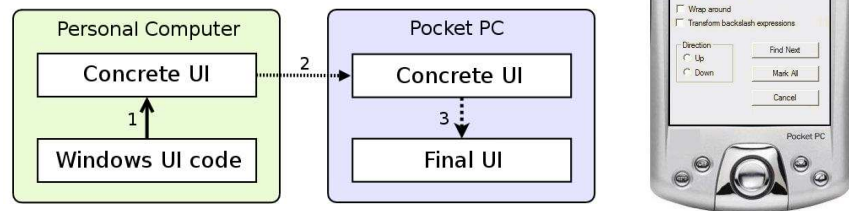
The framework exhibits three types of relationships:

- An **abstraction** transforms any specifications into specifications at a higher level of abstraction.
- **Reification** transforms any specifications into specifications at a lower level of abstraction.
- A **translation** is about transformation of the interface from one type of platform to another, or more generally, from one context to another. The term adaptation is also often used: an existing system is adapted to a new context of use. If the adaptation is performed at a high level of abstraction, the resulting process will be more flexible.

These relationships allow a multi-path UI development. The development can be started at any entry point of the framework. **Reverse engineering** follows abstractions (an existing system is recuperated). My thesis is concerned with FUI reverse engineering. Forward engineering follows reifications (a new system is produced). Lateral engineering follows translations (for example, when a UI in HTML is already designed for a desktop, we may design a corresponding UI in Java for a mobile phone).

⁵⁷ An Abstraction Interaction Objects (AIO).

Other terms exist to define specific development paths. Transcoding tools perform lateral engineering at the FUI level: a FUI for a source platform is directly transformed into another FUI for a target platform. The re-engineering combines reverse engineering and forward engineering. The retargeting is the re-engineering for another computing platform, for example through the CUI level (as illustrated below): a CUI for a source platform (say that one that I have to generate from a Windows resource file) is transformed into another CUI for a target platform⁵⁸ (as a pocket PC), that I turn leads to a new FUI for that platform:



UsiXML is based on XML, which is a compliant specification language suitable for any interface. This language describes a UI at any above mentioned levels of details and abstractions, depending of the context of use.

It specifies multiple models, structured according to the four layers of the Cameleon framework. This approach is a model-driven architecture (MDA): the interactive system to be build is described by a series of models and transformations between them all expressed in a unified notation. The models produced can be listed and maintained in a model repository, which is fed and accessed by a series of tools, to be reused in an efficient way.

The basics models are the task, domain⁵⁹, AUI, CUI and context⁶⁰ models. The language can also specify the relationship between these models, with the mapping⁶¹ and transformation⁶² models. The AUI and CUI models reflect the AUI and CUI levels. The following presents UsiXML in terms of a UML class diagram (that is, the meta-model of the language). The topmost superclass (uiModel) contains common features shared by all component models of a UI.

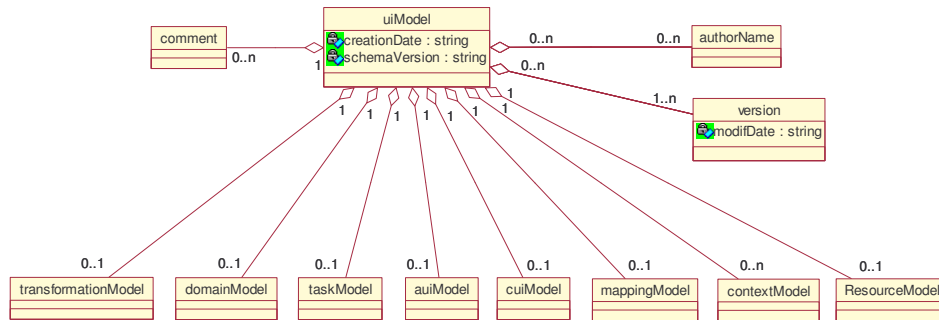
⁵⁸ Graceful degradation techniques may be applied in this case to take into account the constraints imposed by the target platform.

⁵⁹ The domain model is a description of the classes of object manipulated by a user when interacting with a system.

⁶⁰ The context model describes the tree aspects of the context of use.

⁶¹ The mapping model expresses the correspondence between elements of the models (following the reification, abstraction and translation relationships).

⁶² The transformation model formalizes explicitly the transformation between two models with the graph transformation techniques in conformity with the transformation definition (generated from a mapping specification).

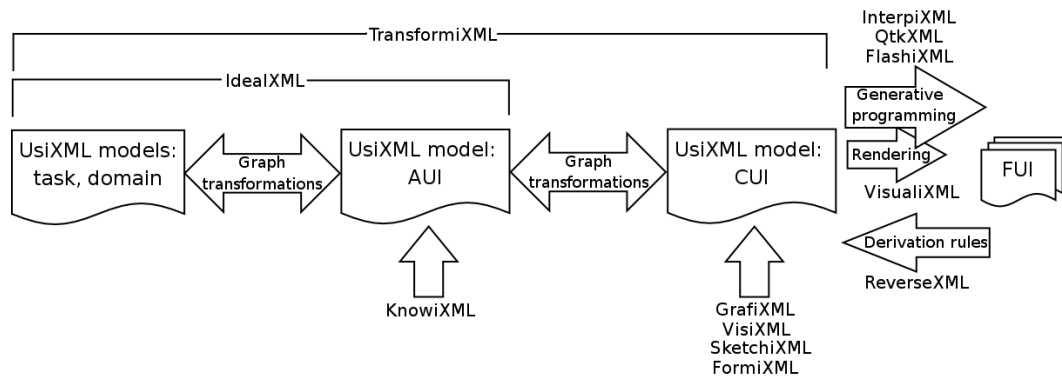


A unified language specifying these models allows designers to exchange and share fragment of specifications and enables tools to operate on these specifications.

UsiXML is currently supported by several tools⁶³ (as IdealXML which specifies the task, domain, AUI and mapping models, GrafiXML which is a CUI editor or ReversiXML which enables FUI reverse engineering of HTML files). To design a user interface, the techniques proposed can follow a forward engineering. The designer uses first IdealXML to build a task and a domain model (which is a UML class diagram, an entity-relation model or an object-oriented model) and establish different types of mappings between the two models (as, in my simple example, mapping the task “enter word” with an attribute or the task “launch search” with an operation). The abstract UI can be then obtained using several rules (for example, the structure of the AUI can be determined from the task model: if a task is decomposed in sub-tasks, then the parent task is put in a container). A set of model-to-model⁶⁴ transformation functions, transforming a UsiXML specification into another UsiXML specification, is automated in TransformiXML. A first arborescence of widgets is then elaborate for each container. The CUI is created, with graph transformations, after choosing the modality. A platform and a language are finally chosen to have a FUI by rendering (interpretation) or code generation. In this manner, the UI is derived from the task model. We reuse what it is made at the previous levels. All these tools can be combined and automatism is privileged to keep the model consistent as modifications occur (when the requirements change). Other path can be followed as well. We can use the fact that UsiXML supports a mutli-path development of UI. The development process can be initiated from any level of abstraction and proceed towards obtaining one or many FUI for various context of use:

⁶³ See <http://www.usixml.org/> for the list of these available tools.

⁶⁴ Model-to-model transformations support any change between models while model-to-code transformations are associate with code production (automated or not).



The task that my GrafXML plug-in should assume is to open a resource script file describing the UI of a Windows-based application (that is, a FUI) and generate the UI definition in UsiXML at the CUI level into GrafXML. The CUI model is presented in the next section.

5.2 Concrete User Interface model

My plug-in is devoted to generate UsiXML specifications at the Concrete User Interface (CUI) level from a resource file of Windows-based application. Therefore, the model which interests me particularly is the CUI model.

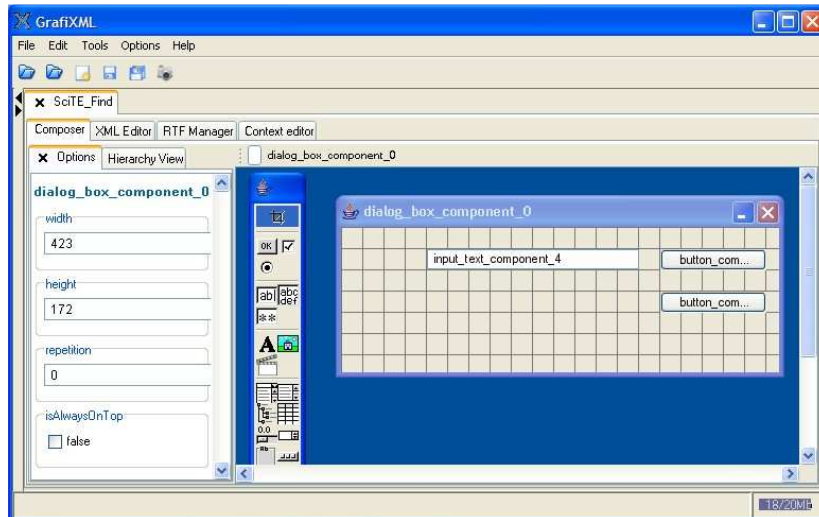
This model describes, in a certain degree of expressiveness, the appearance and behaviour of a UI. A UI represented by this model is dependent of the modality. An instance of the model addresses then a single modality at a time. In my case, the interaction is graphical.

A CUI is however independent of the platform. The elements populating a CUI realize an abstraction of common language used to develop UIs, like Windows.

The CUI model is hierarchically decomposed in Concrete Interaction Objects (CIOs). A CIO is any entity that the user can perceive and manipulate used for the acquisition or restitution of information. CIOs are grouped into two types: graphical containers (such as a window, a dialog box or a group box) and graphical individual components (such as an image, a check box or a progression bar). In my plug-in implementation, I will have to generate and manipulate such CIOs into GrafXML.

The layout of a CUI is not defined with absolute coordinates as in the resource files presented in chapter 4, but with a box embedding mechanism. Alignments between CIOs can be defined with some relationships.

There is the CUI from the first section edited in GrafXML. It's the simplified dialog box of my first example containing a text field and two push buttons and allowing the user to find a word in a text. A CUI is like a blueprint from which a FUI is created on a particular platform, giving the final look.



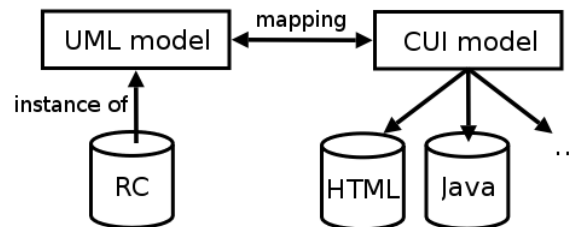
At any time, the user can see the corresponding UsiXML specifications in the XML editor. Note that UsiXML cannot naturally be executed by its own, but relies on an implementation in a third-party rendering engine.⁶⁵ The following is the description of the CUI in UsiXML.

```
<?xml version="1.0" encoding="UTF-8"?>
<cuiModel id="SciTE_Find-cui_23" name="SciTE_Find-cui">
  <dialogBox id="dialog_box_component_0" name="dialog box 400 simplified"
    defaultContent="Find" width="423" height="172">
    <box id="box_1" name="box_1" type="horizontal" relativeWidth="100" relativeHeight="100">
      <box id="box_1_1" name="box_1_1" type="vertical" relativeWidth="70" relativeHeight="100">
        <inputText id="input_text_component_4" name="edit control 222"
          isVisible="true" isEnabled="true" isEditable="true"
          textSize="8" textFont="Microsoft Sans Serif" maxLength="35"
          numberOfLines="1" numberOfColumns="35" glueHorizontal="right"/>
      </box>
      <box id="box_1_2" name="box_1_2" type="vertical" relativeWidth="30" relativeHeight="100">
        <button id="button_component_2" name="default push button 1"
          defaultContent="Find Next" isVisible="true" isEnabled="true"
          textSize="8" textFont="Microsoft Sans Serif" glueHorizontal="middle"/>
        <button id="button_component_3" name="push button 245"
          defaultContent="Mark All" isVisible="true" isEnabled="true"
          textSize="8" textFont="Microsoft Sans Serif" glueHorizontal="middle"/>
      </box>
    </dialogBox>
  </cuiModel>
```

A *cio* is characterized by various attributes such as *id*, *name* or *defaultContent*. All the *CIOs* in this CUI model are *graphicalCio*, which inherits from *cio* and has specific attributes such as *isVisible*, *isEnabled*, *textFont* or *textSize*. A *graphicalCio* can belong to one of the two possible types: *graphicalContainer* or *graphicalIndividualComponent*. For example, a *dialogBox* and a *box* are from the first type and an *inputText* and a *button* are from the second type. The properties of each final element in the hierarchy (that is, their specific and inherited attributes) are limited to describe characteristics of high common interest, independently from the future rendering.

⁶⁵ FlashiXML (which is currently being implemented) can also open a CUI UsiXML file and render it in Flash, QtXML in the Tcl/Tk environment, and JaviXML for Java.

The UML class diagram of the CUI model is given at the next page. The full documentation of the diagram can be found on <http://www.usixml.org>. Each CUI model expressed in UsiXML will be an instance of this meta-model. I can then compare the meta-model with that one given in the previous chapter, and find correspondences between them. This mapping is covered in the next chapter.



Once the resource file will be transformed in UsiXML at the CUI level, code in another language can be generated, even automatically with GrafiXML (in HTML, Java, XHTML or XUL).

6 Importing resource files in GrafiXML

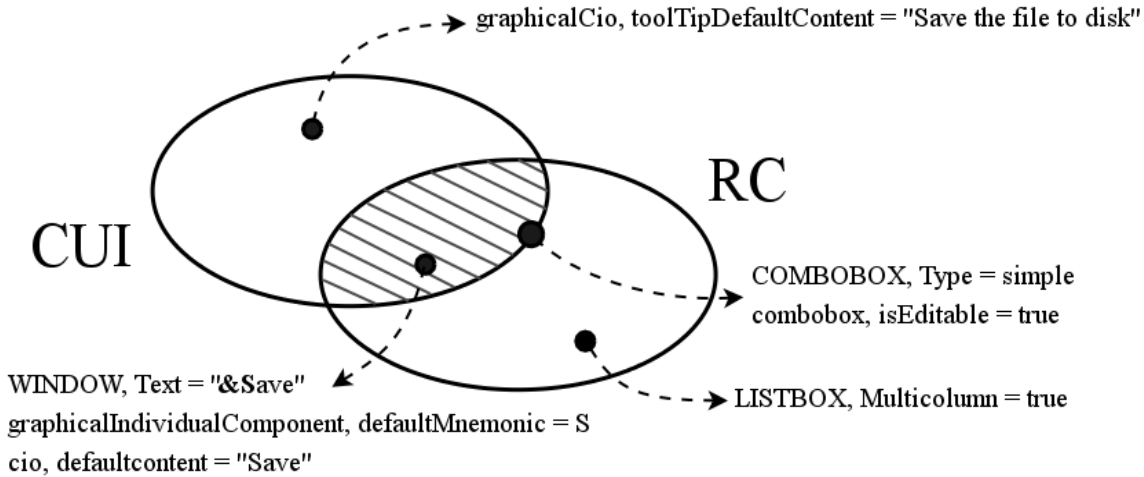
So far, I've presented the specification languages used as input and output of my reverse engineering method. This chapter covers the transformation process from a resource script file (*.rc) up to the generation of Concrete Interaction Objects (widgets provided by a toolkit) in the editor tool GrafiXML. The first section studies this transformation and proposes derivation rules, and the second section explains how it has been implemented as a plug-in of GrafiXML.

6.1 Transformation of resources into CUI

I will explore here how the content of a resource file can be transformed into the CUI model.

6.1.1 Correspondences table

I have to detect the similarities between the two meta-models: on the one hand the class diagram modeling a resource file, on the other hand the class diagram of the CUI model. The objective will be to give a value to a maximum of attributes of the second diagram with the information found in a given resource file (instance of the first diagram). The question that guides this section is to determine which objects (from the *real world*, to refer to databases vocabulary) composing of a graphical user interface and which characteristics defining the appearance and functionality of an object are covered both by the two languages. This corresponds to the crosshatched zone in the picture below. For example, a mnemonic of a menu item can be specified in a resource file as well in the CUI model. From a simple combo box, we have in part the information that the current selection is editable, but the fact that its list box is displayed at all time is not covered in the CUI model.



The following list of tables will serve as guidelines for my implementation. The Windows objects are placed in the left column and their corresponding objects of the CUI model (CIOs) in the right column. An attribute name appears in *italic* and a class name in **bold** to dissociate them. Note that there are some elements in the table that not really correspond, but that I've still decided to associate in my implementation (as such an identifier in a resource file with a name of a CIO).

6.1.1.1 Resources of type dialog box

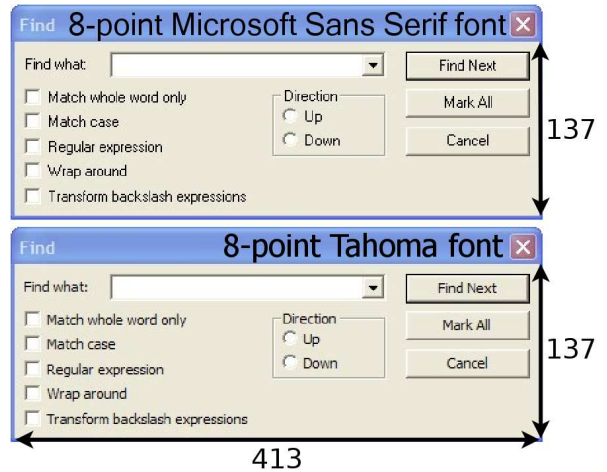
RC	→	CUI
DIALOG		dialogBox or window graphicalCio <i>isVisible</i> = true
		box <i>type</i> = vertical
<i>dlgID</i> = n		cio <i>name</i> = n
<i>Width</i> = w		graphicalContainer <i>width</i> = w*4/xChar + 2*border width (where xChar is the average width of the dialog box font character in pixel) box <i>width</i> = w*4/xChar
<i>Height</i> = h		graphicalContainer <i>height</i> = h*8/yChar + title bar height + bottom border width (where yChar is the average height of the dialog box font character in pixel) box <i>height</i> = h*8/yChar
<i>Text</i> = t and <i>Caption</i> = true and t ≠ null		cio <i>defaultContent</i> = t
<i>FontName</i> = n and <i>SetFont</i> = true		graphicalIndividualComponent <i>textFont</i> = n
<i>ShellFont</i> = true and <i>FontName</i> = "MS Shell DLG"		graphicalIndividualComponent <i>textFont</i> = "Tahoma"
<i>ShellFont</i> = true and <i>FontName</i> ≠ "MS Shell DLG"		graphicalIndividualComponent <i>textFont</i> = "Tahoma"
<i>FontSize</i> = s and (<i>SetFont</i> = true or <i>ShellFont</i> = true)		graphicalIndividualComponent <i>textSize</i> = s
<i>SetFont</i> = false and <i>ShellFont</i> = false		graphicalIndividualComponent <i>textSize</i> = 8, <i>textFont</i> = "Tahoma"
<i>Extended</i> = true and (<i>SetFont</i> = true or <i>ShellFont</i> = true) and <i>Weight</i> ≥ 550		graphicalIndividualComponent <i>isBold</i> = true
<i>Extended</i> = true and (<i>SetFont</i> = true or <i>ShellFont</i> = true) and <i>Italic</i> = true		graphicalIndividualComponent <i>isItalic</i> = true
<i>Disabled</i> = true		graphicalCio <i>isEnabled</i> = false
<i>ThirckFrame</i> = true		window <i>isResizable</i> = true
<i>TopMost</i> = true		graphicalContainer <i>isAlwaysOnTop</i> = true

I choose Tahoma as default font, which is the default system font of Windows 2000 and upper (it was MS Sans Serif for older versions). The size of the font is always 8 when not specified. It's the dimension of the dialog box client area that is specified in a resource file. In the CUI model⁶⁶, a dialog box or a window has always a caption bar and a border which are included in the dimension (a border will be 5 pixels wide and a caption bar will be 30 pixels high). Remember that the units of measure are not the same. The dimension of a graphical container is expressed in pixels, whereas a resource file expresses the measures in horizontal dialog units and vertical dialog units⁶⁷. The dimensions in Windows are then defined in term of

⁶⁶ In reality in the implementation of GraphiXML (still currently being implemented). It's principally to restore in the composer the same layout that the original dialog box that I have adapted the dimension of the container.

⁶⁷ One horizontal (vertical) unit equals 1/4 (1/8) of an average character width (height) of the font used.

characters. Three versions of a dialog box with the same specified dimensions but with different font used are illustrated in this paragraph. Although it should be possible (the system do it), I've not the necessary time and means in my thesis to calculate the average character width and height of each existing font (considering also the size of the font) in order to get the exact dimension of a dialog box in pixels.⁶⁸ To estimate the values of $xChar$ and $yChar$ in the table above, I've used an empiric method. I've choose a dialog box using the default 8-point Tahoma font (or an 8-point MS Sans Serif font which seems to have the same character width and height because the resulting dimension is the same), and I've watched its dimension in pixels. For that one from my first example, I observe that the client area is 413 pixels wide and 137 pixels high.⁶⁹ So, we have for the relations from the table:

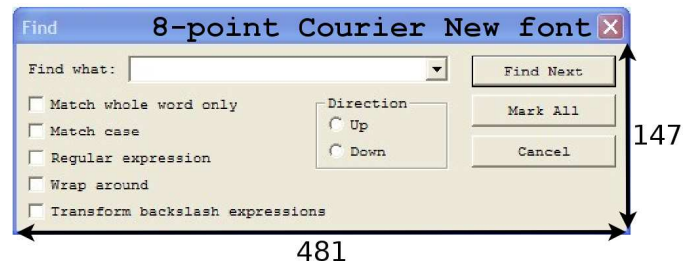


$$413 = \frac{4w}{xChar} \quad \text{and} \quad 137 = \frac{8h}{yChar}$$

Where the width w is 275 and the height h is 84 in dialog units. We have then the estimations:

$$xChar = \frac{1100}{413} \quad \text{and} \quad yChar = \frac{672}{137}$$

Seeing that another font size s can be employed, these values must still be multiplied by $s/8$. If another type of font is used (as Courier New), I decide that the dimensions in the dialog box will not be adapted and some texts may be clipped. It's not frequent to see another font used in a dialog box of a Windows application.⁷⁰ After all, when the user changes the font in a GrafiXML project, this not affects the size values firstly specified. The user has to change manually the size of the dialog box, as the size of some components if text with new font fits in the graphical component.



Note finally that the colours are not in my UML diagram. That is, they are not specified in a resource file but specific to Windows⁷¹. Such data that could be deduced seeing that Windows

⁶⁸ I don't know if we can find documentation about this pixel information. I just know that the Windows function *GetClientRect* obtains the dimensions in pixels (and the function *MapDialogRect* converts the character coordinates in the dialog box to pixel coordinates in the client area).

⁶⁹ To do this, I've edited the desired font in a decompiler, made a print screen and used MSPaint (in "C:\Program Files\Accessories") the number of pixels.

⁷⁰ In fact, I've never seen that in a resource file.

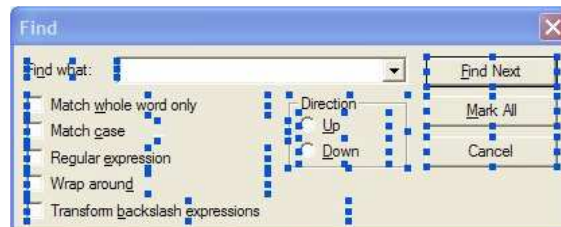
⁷¹ Here I some examples in Windows XP. The grey colour "#d8e9ec" is the surface color of a dialog box and the background color of a control to agree with the color of the client area of the dialog box in which it is designed to be displayed. By default, the colour of an active title bar is blue ("#e45403") and the white ("#ffffff") bold text has the Trebuchet MS font of size 10. The colour of text displayed in the client area of a dialog box is always black ("#000000").

interfaces are used could not be transformed into the CUI model. This concerns the FUI model.

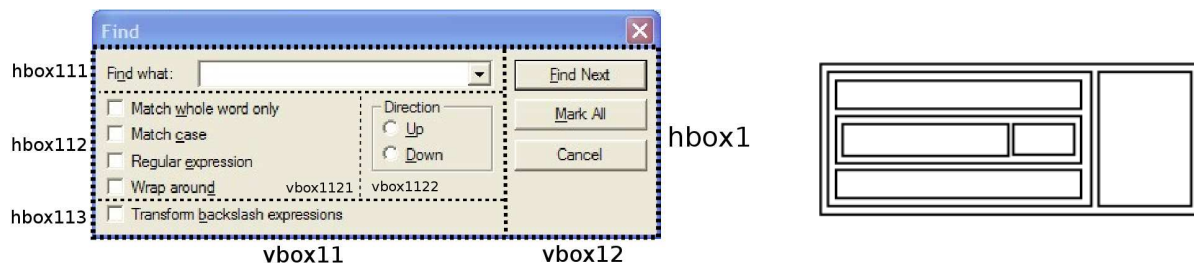
CONTROL	
<i>Position = n</i>	cio name = "control_" + n
<i>X, Y, Width, Height</i>	serve to the creation of boxes with eventually some graphicalAlignment between two components inside a box
<i>Disabled = true</i>	graphicalCio isEnabled = false
<i>Visible = false</i>	graphicalCio isVisible = false
<i>Text = t</i> and '&' is a character of t and the control is a radio button, a push button, a customized button or a check box	graphicalIndividualComponent defaultMnemonic = the character following '&' in t

In my transformation, the identifier of a resource corresponds in general to the name of a CIO.⁷² The user can then still connect the produced object with the resource in the file after the reverse engineering process. However, for child window controls, I use the line number of their definition in the template for the name. They can be then easier localised and most of all in a unique way: the controls who not receive user input (as static controls) have all -1 as identifier.

For a component, the concept of position and dimension inside a window is absent from the CUI model. To place each component, horizontal and vertical boxes will have to be defined using the position and dimension values of all controls specified in the resource file. Alignment properties can also be defined between two components inside a box. Look at the measures in the dialog box from my first example, which are not always visually identifiable for some controls:



We could divide the client area in different ways. The following illustrates one possibility to create the first boxes:



⁷² And not to the attribute *id* of a CIO. These two identifiers have not the same utility.

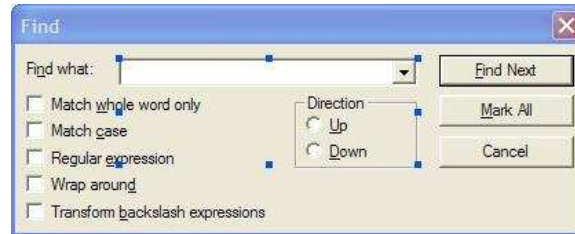
GROUPBOX	box (containing each control that participates in the <i>Group</i> relationship) graphicalCio <i>borderWidth</i> = 1, <i>borderType</i> = line, <i>borderColor</i> = "#000000"
<i>Width</i> = w, <i>Height</i> = h	box <i>width</i> = w*4/xChar, <i>height</i> = h*8/yChar
<i>Text</i> = t and size of t > 0 ⁷³	graphicalCio <i>defaultBorderTitle</i> = t
<i>HorizontalAlignment</i> = left	graphicalCio <i>borderTitleAlign</i> = left
<i>HorizontalAlignment</i> = center	graphicalCio <i>borderTitleAlign</i> = middle
<i>HorizontalAlignment</i> = right	graphicalCio <i>borderTitleAlign</i> = right
RADIOBUTTON <i>PushLike</i> = false	radioButton
<i>Text</i> = t	cio <i>defaultContent</i> = t without any '&' characters
<i>Group</i> = true	radioButton <i>groupName</i> = a given group name for each radio buttons following this radio button in the template and having <i>Group</i> = false
RADIOBUTTON <i>PushLike</i> = true	toggleButton
PUSHBUTTON	button
<i>Text</i> = t and <i>Content</i> = text	cio <i>defaultContent</i> = t without the '&' character
<i>Text</i> = t and <i>Content</i> ≠ text ⁷⁴	cio <i>defaultContent</i> = t
<i>Default</i> = true	graphicalEmphasis
CUSTOMBUTTON	button
CHECKBOX <i>PushLike</i> = false	checkBox
<i>Text</i> = t	cio <i>defaultContent</i> = t without any '&' characters
<i>PushLike</i> = true	toggleButton
<i>Group</i> = true	checkBox <i>groupName</i> = a given group name for each check boxes following this check box in the template and having <i>Group</i> = false
LISTBOX	listBox <i>isEditable</i> = false
<i>height</i> = h	listBox <i>maxLineVisible</i> = h/(11*s/8) (where / is an integer division ⁷⁵ , and s is the size of the font use in the dialog box)
<i>ExtendedSelection</i> = true	listBox <i>multipleSelection</i> = true
EDIT <i>Width</i> = w	inputText <i>isEditable</i> = true, <i>numberOfColumns</i> = w/4
<i>Alignment</i> = left	inputText <i>textHorizontalAlign</i> = left
<i>Alignment</i> = center	inputText <i>textHorizontalAlign</i> = middle
<i>Alignment</i> = right	inputText <i>textHorizontalAlign</i> = right
<i>MultiLine</i> = false	inputText <i>numberOfLines</i> = 1
<i>MultiLine</i> = false and <i>AutoHorizontalScroll</i> = false	inputText <i>maxLength</i> = w/4
<i>PasswordField</i> = true	<i>isPassword</i> = true
<i>Number</i> = true	<i>defaultFilter</i> = [0-9]
<i>Height</i> = h and <i>MultiLine</i> = true	inputText <i>numberOfLines</i> = h/(11*s/8) (where s is the size of the font)
<i>MultiLine</i> = true and <i>AutoHorizontalScroll</i> = false and <i>HorizontalScrollBar</i> = false	inputText <i>wordWrapped</i> = true, <i>forceWordWrapped</i> = true
<i>Width</i> = w and <i>MultiLine</i> = true and <i>AutoVerticalScroll</i> = false and <i>VerticalScrollBar</i> = false	inputText <i>maxLength</i> = w/4 * <i>numberOfLines</i>
COMBOBOX <i>Height</i> = h and <i>Type</i> = simple	listBox <i>isEditable</i> = true, <i>multipleSelection</i> = false, <i>maxLineVisible</i> = h/(11*s/8) (where s is the size of the font)
COMBOBOX <i>Height</i> = h and <i>Type</i> ≠ simple	comboBox <i>maxLineVisible</i> = h/(11*s/8) - 1 (where s is the size of the font)
COMBOBOX <i>Type</i> = dropDown	comboBox <i>isEditable</i> = true
COMBOBOX <i>Type</i> = dropDownList	comboBox <i>isEditable</i> = false

⁷³ In my implementation, the text in the control definition will be never null (contrary to a dialog box which can have no title specified in the caption bar) as in the syntax of the generic notation (in any case, the variable storing the text will be initialised to an empty sting before parsing the text).

⁷⁴ That is, an icon or bitmap.

⁷⁵ The fractional part of the result is truncated and discarded.

I've chosen to represent a simple combo box (that is, the list box is displayed at all times and the current selection is editable) in Windows with an editable list box in the CUI model. For drop-down and drop-down list combo boxes, care must be taken when reading the height specified in the resource file, as we can see in the same example illustrated above:



The problem is that the drop-down list is counted in the height of the control. The height must be revised when creating boxes by considering only the field of the current selection (14 horizontal dialog units when the font size is 8, then $14*s/8$ when the font is s).

Note that the dimensions of graphical individual components are not explicitly specified in the CUI model. This will pose a problem for a button. The dimension of this CIO (as the dimension of a radioButton, checkbox or toggleButton) is defined by the length of the text put in it, and then it cannot have a larger size (and be aligned with other as illustrated above). In addition, if an image is put in the button instead of text, its size is undefined since this resource (and then the size of the image) is not available. The width of a listBox depends also of the length of its items. The problem is the same: the strings contained in a list box are not specified in a resource file.

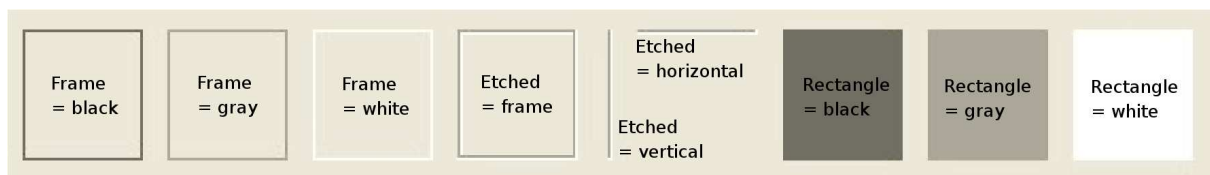
PROGRESSBAR	progressionBar
<i>Type</i> = marquee	<i>indeterminate</i> = true
<i>Type</i> ≠ marquee	<i>indeterminate</i> = false
<i>Vertical</i> = true (false)	<i>orientation</i> = vertical (horizontal)
STATIC <i>Type</i> = ownerDraw, <i>Width</i> = w, <i>Height</i> = h	outputText <i>numberOfColumns</i> = w/4, <i>numberOfLines</i> = h/8
STATIC <i>Type</i> = text, <i>Width</i> = w, <i>Height</i> = h	outputText <i>textVerticalAlign</i> = top, <i>numberOfColumns</i> = w/4, <i>numberOfLines</i> = h/8
<i>Text</i> = t, <i>NoPrefix</i> = false	cio <i>defaultContent</i> = t without any '&' characters
<i>Text</i> = t, <i>NoPrefix</i> = false and '&' is a character of t	graphicalIndividualComponent <i>defaultMnemonic</i> = the character following '&' in t Note : here the attribute is set to the cio which is the abstraction of the first control having <i>Tabstop</i> =true after this static control in the resource file
<i>Text</i> = t, <i>NoPrefix</i> = true	cio <i>defaultContent</i> = t
<i>TextStyle</i> = left or simple or leftNoWordWrap	<i>textHorizontalAlign</i> = left
<i>TextStyle</i> = center	<i>textHorizontalAlign</i> = middle
<i>TextStyle</i> = right	<i>textHorizontalAlign</i> = right
STATIC <i>Type</i> = image	imageComponent
<i>Text</i> = t	cio <i>name</i> = t
<i>RealSizeImage</i> =true, <i>Width</i> = w, <i>Height</i> = h	imageComponent <i>width</i> = w*4/xChar, <i>height</i> = h*8/yChar
<i>Border</i> = true or <i>Sunken</i> = true	imageComponent <i>imageBorder</i> = 1
STATIC <i>Type</i> = frame, <i>Width</i> = w, <i>Height</i> = h, <i>Etched</i> ≠ horizontal or vertical	box <i>width</i> = w*4/xChar, <i>height</i> = h*8/yChar (containing each control that participates in the <i>Group</i>)

	relationship) graphicalCio <i>borderWidth = 1, borderType = line</i>
<i>Etched = horizontal, Height = 1</i>	box <i>width = w*4/xChar, height = 1</i> graphicalCio <i>borderWidth = 1, borderType = line</i>
<i>Etched = vertical, Width = 1</i>	box <i>width = 1, height = h*8/yChar</i> graphicalCio <i>borderWidth = 1, borderType = line</i>
STATIC <i>Type = enhancedMetafile</i>	imageComponent <i>width = w*4/xChar, height = h*8/yC</i>

For a static control used to display an image, the given text is the name of an icon or a bitmap (not a filename) defined elsewhere in the resource file. For example, if the text of such control is "SCITE", the file (here from Restorator) has somewhere the line: `SCITE ICON SCITE.ico` But when we extract the resource and save it to the disk with a decompiler (*.ico), the name it gives to the file is not necessary the same as specified (for example, we have a file names Icon_1.ico). So, I've not exploited this information (an image file name is normally put in the *defaultContent* attribute of an ImageComponent). The user can insert the image in the GrafiXML project, knowing the name of the image resource (put in the *name* attribute for an ImageComponent overriding the name that I have defined for each control). We can't know the real size of an image resource from a resource file. The user should also put these values manually in the GrafiXML composer.

The width and height values specified in the resource file for a static control displaying an image are also ignored in Windows (except if *RealSizeImage = false*): the control auto sizes itself to accommodate the image. This will complicate the creation of boxes since these values can be not reliable information.

A static control can also represent frames corresponding to container in the CUI model. The following illustration summarizes the different types of static controls used to draw frames.



The values for the colour do not necessary mean that the colour are black, gray or white but are based on a Windows system colour: the colour used to draw window frame, to fill the screen background or to fill the window background. And the default values can be changed. So, this colour information in a resource file is in reality not present.⁷⁶ In Windows, the flags `SS_ETCHEDHORZ` and `SS_ETCHEDVERT` are always used to have an etched looking vertical or horizontal line in a rectangle one unit high or wide (as in my third example).

⁷⁶ In my implementation, I've still set these default values ("#636f71" for black, "#99a8ab" for gray and "#ffffff" for white) to the *bgColour* and *borderColor* attributes for the corresponding box to visually identify the container in the GrafiXML editor.

TRACKBAR	slider
<i>Orientation = horizontal</i>	<i>Orientation = orizontal</i>
<i>Orientation = vertical</i>	<i>Orientation = vertical</i>
UPDOWN <i>AutoBuddy = true</i> EDIT (participating to the relationship GlueTo)	spin
TREEVIEW	tree
DATETIMEPICKER <i>format ≠ time</i>	datePicker
DATETIMEPICKER <i>format = time</i>	hourPicker

If the buddy window (the previous window in the z-order) of an up-down control is an edit control, this last object should be ignored when parsing the file (a spin has already an editable field in the CUI model).

6.1.1.2 Resources of type menu

MENUBAR	menuBar <i>position = up</i>
<i>MemuId = n</i>	cio <i>name = n</i>
POPUPMENU	menuPopUp
<i>Text = t</i>	cio <i>defaultContent = t</i> without the ‘&’ character
<i>Sate = enabled</i>	graphicalCio <i>isEnabled = true</i>
<i>Sate ≠ enabled</i>	graphicalCio <i>isEnabled = false</i>
MENUITEM	menuItem
<i>ItemId = n</i>	cio <i>name = n</i>
<i>Text = t</i>	cio <i>defaultContent = t</i> without the ‘&’ character
<i>Sate = enabled</i>	graphicalCio <i>isEnabled = true</i>
<i>Sate ≠ enabled</i>	graphicalCio <i>isEnabled = false</i>
<i>Checked = true</i>	menuItem <i>type = toggle</i>
<i>Text = t</i> and ‘&’ is a character of t	graphicalIndividualComponent <i>defaultMnemonic = the character following ‘&’ in t</i>
<i>Text = t</i> and “...” is a substring of t	menuItem <i>type = command</i>
<i>Text = t</i> and ‘\t’ is a character of t	menuItem <i>defaultKeyboardShortcut = the key combination specified after ‘\t’ in t</i>
SEPARATOR	separator

In Windows’ menus, the programmer usually indicates that a menu item invokes a dialog box by adding an ellipsis (...) to the text. If I found such characters in the text of a menu item, I make the assertion that it correspond to menu item of type command in the CUI model. For a pop-up menu, the mnemonic cannot be exploited since this attribute is currently not present for a container in the CUI model.

Note that I’ll have to create a **window** graphical container before creating a menu bar object into GrafiXML to represent the overlapped window (the program’s main application window) which is not specified in a resource file.

6.1.2 Derivation rules

This section gathers some mapping rules⁷⁷ specifying the correspondence between the two models: that one modeling a Windows resource (of type menu or dialog box) and the CUI model. The list is not exhaustive, but is shown to see the nature of such rules and show that they can be elaborated with a Windows resource file as source of the mapping. Only rules concerning a dialog box will be developed. I will first explain a specific notation for reverse engineering derivation rules, and I will then define precisely in this notation a subset of the rules appearing in tables of the previous section.

A standard notation [11] can be used to express formally reverse engineering derivation rules for a UI specified in any language (or source model). The rules are applied on trees representing a UI: T_s is the source tree (an instance of the diagram modeling a Windows dialog box resource) and T_t is the target tree (an instance of the CUI model). The nodes of a tree T store hierarchically the elements composing the UI. Each connection (or arc) represents a containment relationship between the parent and the child.⁷⁸ Each node of the tree represents the different elements composing the UI. Each node can possess zero or more attributes. To construct T_t , I will use the following predefined basic update operations:

- AddNode(*class*, *id*): add a new node with the identifier *id* storing an element which is an instance of *class*.
- AddAttribute(*id*, *name*, *value*): add to the node *id* the attribute *name* initialised to *value*.
- ModifyAttribute(*id*, *name*, *newname*, *newvalue*): suppress the attribute *name* of the node *id* and add the attribute *newname* with the value *newvalue*.
- AddArc (*idSource*,*idTarget*): connect the parent node *idSource* with its child node *idTarget*.

Rules identifying containers

This first group of rules is applied first and the two new nodes are linked once created. The first rule of each group corresponds to the detection of a node in the source tree that causes the creation of a node in the target tree.

$$\begin{aligned} &\forall x \in T_s : x = \text{DIALOG} \rightarrow \text{AddNode}(\text{"dialogBox"}, \text{rootId}) \\ &\quad \wedge \text{AddAttribute}(\text{rootId}, \text{"id"}, \text{rootId}) \wedge \text{AddAttribute}(\text{rootId}, \text{"name"}, x.\text{DlgId}) \\ &\quad \wedge \text{AddAttribute}(\text{rootId}, \text{"isVisible"}, \text{"true"}) \wedge \text{AddAttribute}(\text{rootId}, \text{"isEnabled"}, \text{NOT } x.\text{Disabled}) \\ &\quad \wedge \text{AddAttribute}(\text{rootId}, \text{"width"}, 3304 \times x.\text{Weight} / 275 \times x.\text{FontSize} + 10) \\ &\quad \wedge \text{AddAttribute}(\text{rootId}, \text{"height"}, 274 \times x.\text{Height} / 21 \times x.\text{FontSize} + 35) \text{ where } \text{rootId} = \sum \text{node} \in T_t \\ &\forall x \in T_s : x = \text{DIALOG} \wedge x.\text{Caption} = \text{true} \rightarrow \text{AddAttribute}(\text{rootId}, \text{"defaultContent"}, x.\text{Text}) \\ &\forall x \in T_s : x = \text{DIALOG} \wedge x.\text{TopMost} = \text{true} \rightarrow \text{AddAttribute}(\text{rootId}, \text{"isAlwaysOnTop"}, \text{"true"}) \\ &\forall x \in T_s : x = \text{DIALOG} \rightarrow \text{ConstrBox}(\text{boxId}, \text{"vertical"}) \wedge \text{AddArc}(\text{rootId}, \text{boxId}) \text{ where } \text{boxId} = \sum \text{node} \in T_t \end{aligned}$$

⁷⁷ The mapping is direct because the two models are expressed in the same formalism (UML diagrams).

⁷⁸ Remark that the maximum height of T_s is one: the only container in a Windows dialog box is a window which can contains child windows.

The function ConstrBox(id,type) is defined as:

$$\begin{aligned} & \text{AddNode}(\text{"box"}, \text{id}) \wedge \text{AddAttribute}(\text{id}, \text{"type"}, \text{type}) \wedge \text{AddAttribute}(\text{id}, \text{"isEnabled"}, \text{"true"}) \\ & \wedge \text{AddAttribute}(\text{id}, \text{"isVisible"}, \text{"true"}) \wedge \text{AddAttribute}(\text{id}, \text{"id"}, \text{id}) \end{aligned}$$

The next group identify group boxes (not containers in Windows).

$$\begin{aligned} & \forall x \in T_s : x = \text{GROUPBOX} \rightarrow \text{ConstrBox}(\text{boxId}, \text{vertical}) \wedge \text{AddAttribute}(\text{boxId}, \text{"borderWidth"}, \text{"1"}) \\ & \wedge \text{AddAttribute}(\text{boxId}, \text{"borderType"}, \text{"line"}) \wedge \text{AddAttribute}(\text{boxId}, \text{"name"}, x.\text{CtrlId}) \\ & \text{where } \text{boxId} = \sum_{\text{node} \in T_t} \\ & \forall x \in T_s : x = \text{GROUPBOX} \wedge (\text{ParentNode}(x).\text{SetFont} = \text{true} \vee \text{ParentNode}(x).\text{ShellFont} = \text{true}) \rightarrow \\ & \text{AddAttribute}(\text{boxId}, \text{"width"}, 3304 \times x.\text{Weight} / 275 \times \text{ParentNode}(x).\text{FontSize}) \\ & \wedge \text{AddAttribute}(\text{boxId}, \text{"height"}, 274 \times x.\text{Height} / 21 \times \text{ParentNode}(x).\text{FontSize}) \\ & \forall x \in T_s : x = \text{GROUPBOX} \wedge \text{ParentNode}(x).\text{SetFont} = \text{false} \wedge \text{ParentNode}(x).\text{ShellFont} = \text{false} \rightarrow \\ & \text{AddAttribute}(\text{boxId}, \text{"width"}, 413 \times x.\text{Weight} / 275) \wedge \text{AddAttribute}(\text{boxId}, \text{"height"}, 137 \times x.\text{Height} / 84) \\ & \forall x \in T_s : x = \text{GROUPBOX} \wedge \text{Size}(x.\text{Text}) \neq 0 \wedge x.\text{HorizontalAlignment} \neq \text{center} \rightarrow \\ & \text{AddAttribute}(\text{boxId}, \text{"defaultBorderTitle"}, x.\text{Text}) \\ & \wedge \text{AddAttribute}(\text{boxId}, \text{"borderTitleAlign"}, x.\text{HorizontalAlignment}) \\ & \forall x \in T_s : x = \text{GROUPBOX} \wedge \text{Size}(x.\text{Text}) \neq 0 \wedge x.\text{HorizontalAlignment} = \text{center} \rightarrow \\ & \wedge \text{AddAttribute}(\text{boxId}, \text{"borderTitleAlign"}, \text{"middle"}) \\ & \forall x \in T_s : x = \text{GROUPBOX} \wedge x.\text{Disabled} = \text{true} \rightarrow \text{ModifyAttribute}(\text{boxId}, \text{"isEnabled"}, \text{"isEnabled"}, \text{"false"}) \\ & \forall x \in T_s : x = \text{GROUPBOX} \wedge x.\text{Visible} = \text{false} \rightarrow \text{ModifyAttribute}(\text{boxId}, \text{"isVisible"}, \text{"isVisible"}, \text{"false"}) \end{aligned}$$

Parentnode(x) returns the parent node of node x. The rule is similar for a static control drawing a frame in a dialog box.

Rules identifying components

I give as example two groups of rules relative to a combo box and radio buttons.

$$\begin{aligned} & \forall x \in T_s : x = \text{COMBOBOX} \wedge x.\text{Type} \neq \text{simple} \rightarrow \text{AddNode}(\text{"comboBox"}, \text{comboId}) \\ & \wedge \text{AddAttribute}(\text{comboId}, \text{"name"}, x.\text{CtrlId}) \wedge \text{AddAttribute}(\text{comboId}, \text{"id"}, \text{comboId}) \\ & \text{where } \text{comboId} = \sum_{\text{node} \in T_t} \\ & \forall x \in T_s : x = \text{COMBOBOX} \wedge x.\text{Type} \neq \text{simple} \\ & \wedge (\text{ParentNode}(x).\text{SetFont} = \text{true} \vee \text{ParentNode}(x).\text{ShellFont} = \text{true}) \rightarrow \\ & \text{AddAttribute}(\text{comboId}, \text{"maxLineVisible"}, \text{ParentNode}(x).\text{Height} / (11 \times \text{ParentNode}(x).\text{FontSize}) - 1) \\ & \forall x \in T_s : x = \text{COMBOBOX} \wedge x.\text{Type} \neq \text{simple} \wedge \text{ParentNode}(x).\text{SetFont} = \text{false} \\ & \wedge \text{ParentNode}(x).\text{ShellFont} = \text{false} \rightarrow \\ & \text{AddAttribute}(\text{comboId}, \text{"maxLineVisible"}, \text{ParentNode}(x).\text{Height} / 11 - 1) \\ & \forall x \in T_s : x = \text{COMBOBOX} \wedge x.\text{Type} = \text{dropDown} \rightarrow \text{AddAttribute}(\text{comboId}, \text{"isEditable"}, \text{"true"}) \\ & \forall x \in T_s : x = \text{COMBOBOX} \wedge x.\text{Type} = \text{dropDownList} \rightarrow \text{AddAttribute}(\text{comboId}, \text{"isEditable"}, \text{"false"}) \\ & \forall x \in T_s : x = \text{COMBOBOX} \wedge x.\text{Disabled} = \text{true} \rightarrow \text{ModifyAttribute}(\text{boxId}, \text{"isEnabled"}, \text{"isEnabled"}, \text{"false"}) \\ & \forall x \in T_s : x = \text{COMBOBOX} \wedge x.\text{Visible} = \text{false} \rightarrow \text{ModifyAttribute}(\text{boxId}, \text{"isVisible"}, \text{"isVisible"}, \text{"false"}) \end{aligned}$$

```


$$\forall x \in T_s : x = \text{RADIOBUTTON} \wedge x.\text{PushLike} = \text{false} \rightarrow \text{AddNode}(\text{"radioButton"}, \text{radioId})$$


$$\wedge \text{AddAttribute}(\text{radioId}, \text{"name"}, x.\text{CtrlId}) \wedge \text{AddAttribute}(\text{radioId}, \text{"id"}, \text{radioId})$$


$$\wedge \text{AddAttribute}(\text{boxId}, \text{"defaultContent"}, \text{WithoutAmper}(x.\text{Text}))$$


$$\text{where } \text{radioId} = \sum \text{node} \in T_t$$


$$\forall x \in T_s : x = \text{RADIOBUTTON} \wedge x.\text{PushLike} = \text{false} \wedge \text{"\&"} \in x.\text{Text} \rightarrow$$


$$\text{AddAttribute}(\text{boxId}, \text{"defaultMnemonic"}, \text{CharAfterAmper}(x.\text{Text}))$$


$$\forall x \in T_s : x = \text{RADIOBUTTON} \wedge x.\text{PushLike} = \text{false} \wedge x.\text{Disabled} = \text{true} \rightarrow$$


$$\text{ModifyAttribute}(\text{radioId}, \text{"isEnabled"}, \text{"isEnabled"}, \text{"false"})$$


$$\forall x \in T_s : x = \text{RADIOBUTTON} \wedge x.\text{PushLike} = \text{false} \wedge x.\text{Visible} = \text{false} \rightarrow$$


$$\text{ModifyAttribute}(\text{radioId}, \text{"isVisible"}, \text{"isVisible"}, \text{"false"})$$


```

WithoutAmp(t) gives returns the text t without any ampersand (&). CharAfterAmp(t) returns the letter that follows the first ampersand of t .

Rules for the exclusion of radio button

```


$$\forall x, z \exists y \in T_s, \exists a \in T_t : x = \text{RADIOBUTTON} \wedge x.\text{Group} = \text{false} \wedge y.\text{Position} < x.\text{Position}$$


$$\wedge y = \text{RADIOBUTTON} \wedge y.\text{Group} = \text{true} \wedge z.\text{Position} > y.\text{Position} \wedge z.\text{Position} < x.\text{Position}$$


$$\wedge z = \text{RADIOBUTTON} \wedge z.\text{Group} = \text{false} \wedge a = \text{itao}(x) \rightarrow \text{AddAttribute}(a.\text{id}, \text{"groupName"}, y.\text{id})$$


```

itao(x) (“is the abstraction of”) returns a node of T_t which is the result of an abstraction of the node x (from T_s). A relation is automatically created for each creation of nodes in the target tree. This relation can be of the type one to many or one to one.

Rules relative to the font

```


$$\forall x \in T_t, \exists y \in T_s : \text{isRoot}(y) = \text{true} \wedge y.\text{SetFont} = \text{true} \wedge (x = \text{inputText} \vee x = \text{outputText} \vee x = \text{button}$$


$$\vee x = \text{radioButton} \vee x = \text{toggleButton} \vee x = \text{checkBox} \vee x = \text{spin} \vee x = \text{comboBox} \vee x = \text{tree}$$


$$\vee x = \text{listBox} \vee (x = \text{box} \wedge \text{itro}(x) = \text{GROUPBOX})) \rightarrow \text{AddAttribute}(x.\text{id}, \text{"textFont"}, y.\text{FontName})$$


$$\wedge \text{AddAttribute}(x.\text{id}, \text{"textSize"}, y.\text{FontSize})$$


$$\forall x \in T_t, \exists y \in T_s : \text{isRoot}(y) = \text{true} \wedge y.\text{SetFont} = \text{false} \wedge (x = \text{inputText} \vee x = \text{outputText} \vee x = \text{button}$$


$$\vee x = \text{radioButton} \vee x = \text{toggleButton} \vee x = \text{checkBox} \vee x = \text{spin} \vee x = \text{comboBox} \vee x = \text{tree}$$


$$\vee x = \text{listBox} \vee (x = \text{box} \wedge \text{itro}(x) = \text{GROUPBOX})) \rightarrow \text{AddAttribute}(x.\text{id}, \text{"textFont"}, \text{"Tahoma"})$$


$$\wedge \text{AddAttribute}(x.\text{id}, \text{"textSize"}, \text{"8"})$$


```

itro(x) (“is the reification of”) returns a node of T_s which is the result of an reification of the node x (from T_t).

6.2 Plug-in development

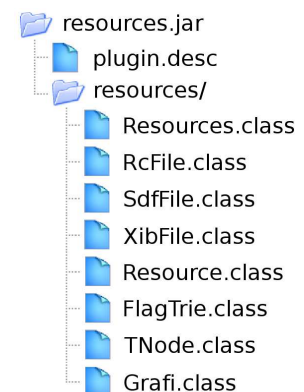
I see in this section how are implemented the set of derivation rules. The programming language used to implement the plug-in is Java (seeing that GrafiXML is developed in Java). The source code, well documented, is in appendix D.

6.2.1 How to run it?

To integrate a plug-in into GrafiXML⁷⁹, the ImportPlugin interface has to be implemented. This interface is in the `be.ac.ucl.isys.grafixml.plugins` package, which is the first import declaration of my program.

```
public interface ImportPlugin extends ImportExportPlugin {
    /* load the file chosen by the user and do something with it */
    public boolean load(File file);
    /* used to have a preview of the given file */
    public boolean loadFileContent(File file);
}
public interface ImportExportPlugin extends Plugins {
    /* return the extensions of the files that can be imported */
    public String[] getExtensions();
    /* return the type name */
    public String getExtensionName();
}
public interface Plugins {
    /* return the plugin name */
    public String getPluginName();
    /* return the author name of the plug-in */
    public String getPluginAuthor();
    /* return the versions of GrafiXML that supports the plug-in */
    public String getPluginDepend();
    /* return the plugin version */
    public String getPluginVersion();
    /* return the plugin description */
    public String getPluginDesc();
}
```

It's `Resources.java` that implements the `ImportPlugin` interface. I've also define seven other auxiliary classes to carry out my task. Once compiled, all my classes are put in a jar file with the structure shown to the right, where `plugin.desc` is a text file which contains the line "main-class=resources.Resources" which is the path to the main class of the plug-in. I put then this jar file in the Windows repertory `\Document And Settings\user_name\grafixml.plugins`.



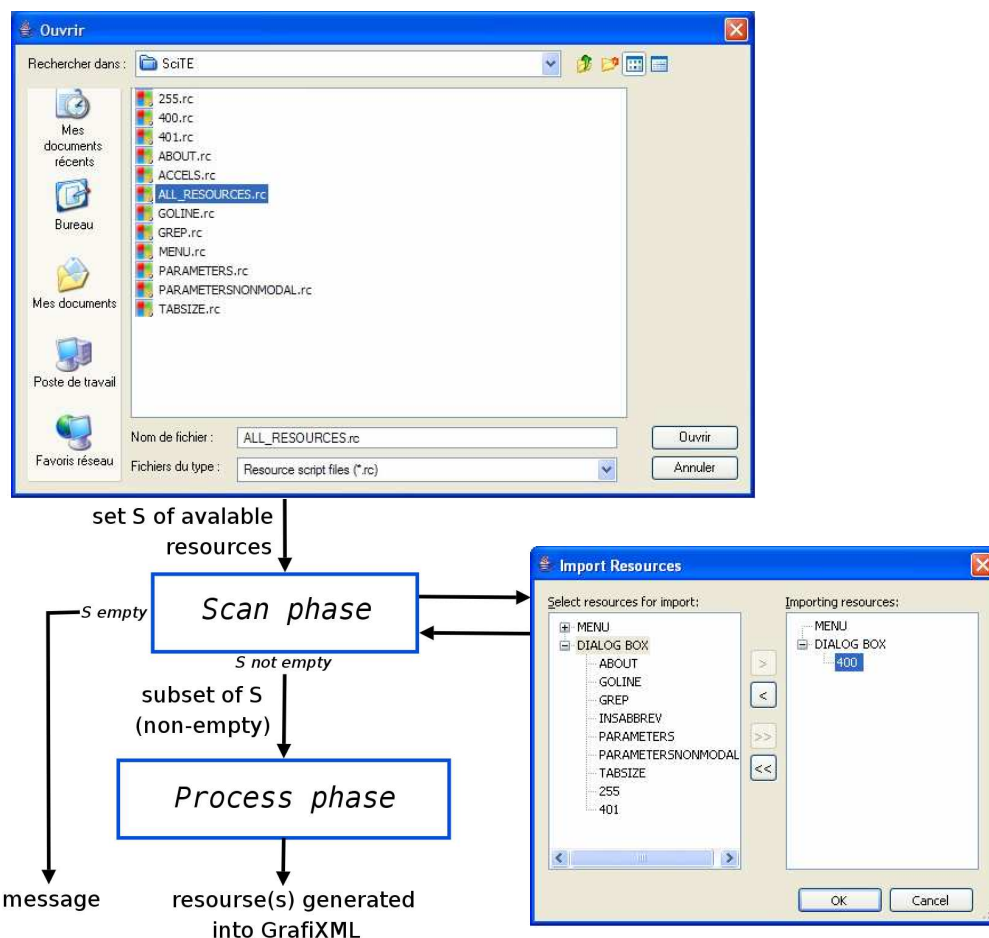
My plug-in can finally be run and tested by launching for example GrafiXML with Java Web Start.⁸⁰ A new submenu will appear when selecting the menu item 'import'.

⁷⁹ See the wiki page <http://www.usixml.org/index.php?view=wiki&title=GrafiXMLImportPlugins>.

⁸⁰ Go on <http://www.usixml.org/index.php?view=page&idpage=10> to download `grafixml.jnlp`.

6.2.2 How to use it?

I've decided to divide my implementation into two phases. When a resource file is imported, the text is first scanned to find the available resources. If the search fails (the file is empty, or the resources found are not exploitable, that is are not of the type dialog box or menu), a message is displayed to the user. Else, the set of available resources are shown in the 'Import Resources' dialog box and the user can select those to import in a project. The location in the file of the selected resources is then memorized (if there is no selection when the user presses the 'OK' button, or if the 'Cancel' button is pressed, nothing happens and the plug-in execution end). This module is the scan phase. In the second module (the process phase), the chosen resources are then each in turn analysed and transformed into GrafiXML by implementing my correspondence rules (see section 6.1). The architecture of my plug-in is pictured below.



This section will serve also as user manual and show that the interaction with the plug-in is simple. The graphical interface is composed of two dialog boxes. I've chosen the javax.Swing package⁸¹ to create in Java the graphical interfaces. The user has to choose a file to load, and the resources to import. The user can then continue its works on the editor of grafiXML. Note that before importing a file, the user must first create a new project (or open an existing one).

⁸¹ See <http://java.sun.com/j2se/1.4.2/docs/api/javaw/swing/package-summary.html>.

6.2.3 Specifications

I state here clearly the requirements of the plug-in (the *what*). Pre-specifications and post-specifications of my program are established as:

Input : An existing Windows resource script file (*.rc) f .

Output : f is closed and unchanged. The user-selected resources of type dialog box or of type menu from f are transformed into CUI in GrafiXML, or a message if f doesn't contain any exploitable resource (that is, it doesn't exist in f a dialog box or menu resource described in the expected syntax).

An error message will also be displayed if the file cannot be found (pre-specification not satisfied). Notice that only Windows resource can currently be processed in the plug-in, nothing happens if a file with another format (as an Apple resource file or a screen definition file) is imported. So, the specification should change if a future expansion of my work is undertaken. As yet mentioned, the design can be logically decomposed into two modules. The following is the tasks that each module should perform.

Module 1 : the scan phase

Input : An existing Windows resource script file f of a valid structure, that is, each different resource must be separated by at least one empty line of text. An empty line is a line containing only white space (consists of blanks, tabs and new line characters).

Output : f is closed and unchanged. An array a is returned, containing the line numbers in f of all the dialog box and menu resources or a subset of them depending of the selection of the user. a cannot be null and its length is 0 if there is no such resource in f (this includes the case of an empty file f). The first in f line starts at 1.

The expected syntax of a resource of type dialog box and menu are defined in sections 4.3.1 to 4.3.3. I suppose here that the user hasn't to modify the file given by a decompiler, and that suppressing a separator of resource (an empty line) is as well serious that suppressing the identifier or the second keyword of a resource. In the two cases, a potential resource is lost.

Module 2 : the process phase

Input : An array a of integers with $a.length \geq 1$ containing line numbers of an existing Windows resource script file f (at least one resource has then been found in f and selected by the user). The first line of f starts at 1.

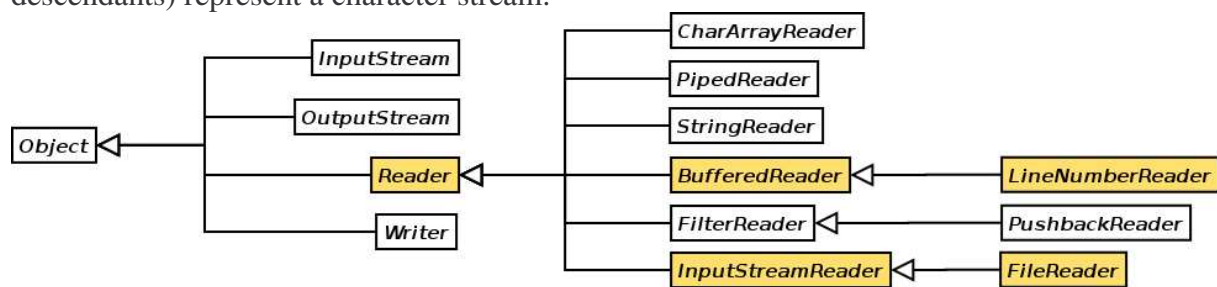
Output : f is closed and unchanged. The n resources defined at line $a[i]$ ($0 \leq i \leq n$) in f are transformed into CUI in GrafiXML.

The program is robust in the sense that it handles unexpected input in the second phase. Some scenarios are inserted in the source code comments. In the first phase, some resources will be simply ignored if the structure of the resource script file is not valid.

6.2.4 Description of my implementation

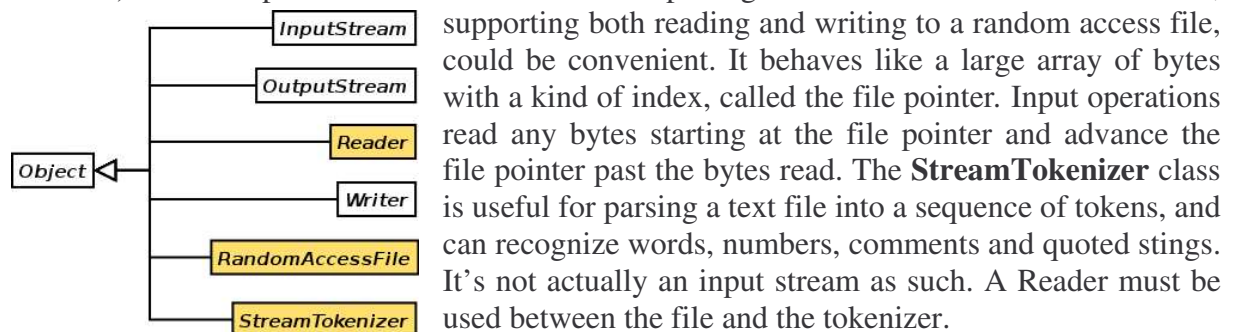
I explain here briefly how my program accomplish its requirements (the *how*) and consider some design questions. This regroups some concepts, some choices I've made during the development, the data structures that I've used and the principal algorithms.

One programming requirement was to read information from an external source (a .rc file on disk). A stream is the underlying mechanism for accomplishing any **I/O operation** in Java. The java.io package⁸² proposes many ways to create streams. We couldn't make head or tail of the classes that can be used to create and manage streams in the Java Standard library. To make a good choice and have a suitable solution to my problem, things can be clarified by subdividing all these classes in two primary ways. Firstly, a program treats a stream as either an input stream or an output stream. In my case, I'll use an **input stream** object from which we read information. Secondly, some classes deal with character data and other with byte data of binary information. I'll naturally use a **character stream**, that are designed to manage 16-bit Unicode characters (resources information are stored in a file as character data) and not 8-bit bytes of row binary data that are not interpreted. These classes are cleanly divided in the class inheritance hierarchy. The InputStream and OutputStream classes (and all their descendants) represent a byte stream, and the Reader and Writer classes (and all their descendants) represent a character stream:



There is another way to divide the classes in the java.io package. A class can acts as either a source or destination (called a data stream), or can provide the means to perform some sort of manipulation on the data in the stream (called a processing stream, or also a filtering stream). This is the same classes, but categorized in another way. The four mentioned primary class in the hierarchy can be further subdivided into those that represent data streams and those that represent processing streams. The Java I/O classes can be combined in many different ways to provide an input stream that behaves exactly as I wish.

These streams have a sequential access to the file (that is, the file is processed from the start to the end). Other specific classes exist in the package. The **RandomAccessFile** class,



supporting both reading and writing to a random access file, could be convenient. It behaves like a large array of bytes with a kind of index, called the file pointer. Input operations read any bytes starting at the file pointer and advance the file pointer past the bytes read. The **StreamTokenizer** class is useful for parsing a text file into a sequence of tokens, and can recognize words, numbers, comments and quoted strings. It's not actually an input stream as such. A Reader must be used between the file and the tokenizer.

⁸² See <http://java.sun.com/j2se/1.4.2/docs/api/java/io/package-summary.html>

Let's see some methods from the class that could be interesting for my problem.

The **Reader** class is an abstract class for reading information stored in text form (Unicode character streams). The only methods that a subclass must implement are the two first:

- `abstract void close()` : Close the stream.
- `int read()` : Read a single character in the file (-1 if eof).
- `void mark(int readLimite)` : Mark the present position in the stream. Subsequent calls to `reset()` will attempt to reposition the stream to this point. *readLimite* is a limit on the number of characters that may be read while still preserving the mark.
- `void reset()` : Reset the stream.

The **BufferedReader** class does not represent any particular data source, but filter data on a given stream by buffering it into more accessible units. In particular, it introduces a new method that enables to read a line at time and returns a String (or null when the end of file is encountered).

- `BufferedReader(Reader in)`: create a buffering character-input stream that uses a default-sized input buffer (2048).
- `String readLine()`: read a line of text.

The string returned can then be processed if I use the **StringTokenizer** class in addition. It allows to break a string into a sequence of tokens (defined by delimiters, e.g. space), but the tokenization process is much simpler than that used by the `StreamTokenizer`.

The **LineNumberReader** class is also a buffered character-input stream but that keeps track of line numbers.

- `LineNumberReader(Reader in)`: create a new line-numbering reader, using the default input-buffer size.
- `int getLineNumber()`: get the current line number.
- `void setLineNumber(int ln)`: set the current line number to ln.

The **InputStreamReader** class is an `InputStreamReader` is a bridge from byte streams to character streams: it reads bytes and decodes them into characters using a specified `char set`.

- `InputStreamReader(InputStream in)`: create an `InputStreamReader` that uses the default `char set`.

The **FileReader** class represents an input file that contains character data. It is convenient for reading character files. Its constructors set up the relationship between the program and the file, opening a stream from which data can be read.

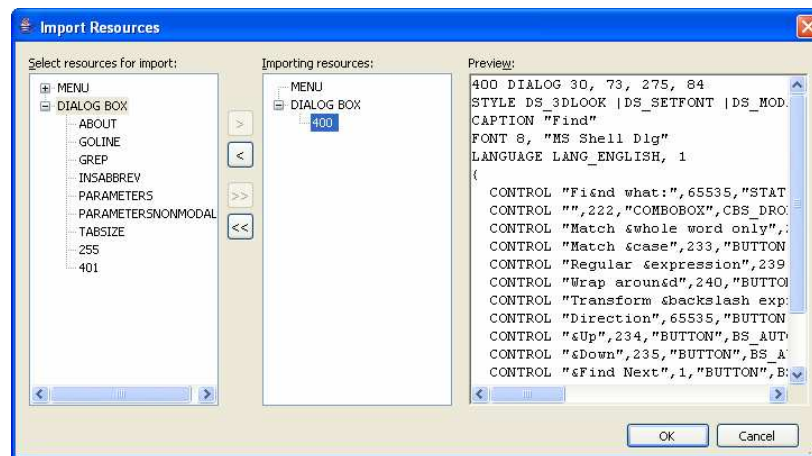
- `FileReader(File f)`: creates a new `FileReader`, given the `File f` to read from.

The **RandomAccessFile** class has a file pointer that can be read by the `getFilePointer` method and set by the `seek` method. It can also read an entire line and then be combined with a `StringTokenizer`.

By combining these classes, I can have the kind of interaction and character manipulation needed for my situation. Let's see some possibilities:

- A `RandomAccessFile`, allowing reading any byte at any location. The file pointer position associated with each resource can be recorded in the scan phase so the selected resources will be localised in the process phase.
- Several instances of `BufferedReader` and `FileReader` on the same file. A reference to a stream object is stored each time a resource location has to be memorized in the scan phase. We continue to read the file with a specific `BufferedReader` in the process phase. But when a resource is found, we have to search the next one from the start of the file.

- A FileReader and a BufferedReader combined with a StringTokenizer. All resources found are stored in a string after the first scan. We could also make an easy preview of each resource to give more information to the user in the dialog box:



But for space usage reason, it is not effective to have an array of many references to long String objects when a large file is processed (even if the garbage collector will deallocate the memory for some objects, set to null when not selected by the user).⁸³ It's better to leave the information on disk, and to extract them as and when we need required. Moreover, a preview is not very needful since the user has not to know this syntax to use this functionality, and the decompiler used by the user to obtain the imported file displays already this information (and an id is already associated with it, even better with a graphical representation of the resource which helps more the user in its choice⁸⁴). I've opted for the simplicity of information given to the user, and not the surcharge.

- A FileReader and a LineNumberReader. We scan through the entire text a first time to find resources and store line numbers, and we go through the text a second time after the user's selection to extract and process each resource at a specified line. The file is then read sequentially twice.

My choice is to combine a FileReader, a LineNumberReader and a *StringTokenizer* in the **module 1** (first scan) to read the file *f* line by line keeping track of line numbers when an exploitable resource is encountered and to parse the first line of a paragraph. A non-empty sequence $\langle L_i, L_{i+1}, \dots, L_{j-1}, L_j \rangle$ of text lines from *f* forms a paragraph if there is no empty line and L_{i-1} (if it's not the first line of *f*) as L_{j+1} (if it's not the last line of *f*) are empty lines. Paragraphs are separated by at least one empty line, and comments at the beginning are not part of a paragraph. The second word of each paragraph of *f* is checked. A non-empty sequence $\langle C_i, C_{i+1}, \dots, C_{j-1}, C_j \rangle$ of characters from *f* is a word if there is no white space character and C_{i-1} (if it's not the first character of *f*) as C_{j+1} (if it's not the last character of *f*) are white space characters. If the checked word is MENU or DIALOG, then the line number is stored and the located resource will be proposed to the user (with its id). Only the lines corresponding to resources selected by the user will be conserved.

⁸³ The full .rc file from the sixth example (TablEdit) contains 81 dialog boxes and Restorator gives 2756 lines.

⁸⁴ It's for this same reason that just the id of a resource will be displayed in my dialog box, without the caption as for example 400 ("Find"). It's not quite meaningful, the user will still have to see in the decompiler to which resource the caption correspond, or worse in the original application searching for the dialog box having this caption.

Algorithm :

```

r ← new FileReader (f);
lnr ← new LineNumberReader (r);
while line = lnr.readLine() not null do
  st ← new StringTokenizer (line, "\t"); /* tab and space are delimiters */
  token ← read the second token
  if token = "DIALOG" or "MENU" then
    store the current line number
    skip the lines of the same paragraph

```

I combine a `FileReader`, a `LineNumberReader` and a `StreamTokenizer` in the **module 2** (process phase). Note that this tokenizer can also know the number of the current line, but it should be time consuming to break a useless part of the file *f* into a sequence of tokens. Each line of *f* is read at once and skipped until reaching the next line specified. The associated resource is then parsed to store all the necessary information into variables. Each object will be then created with all its characteristics in `GrafiXML`, by applying my derivation rules.

Algorithm :

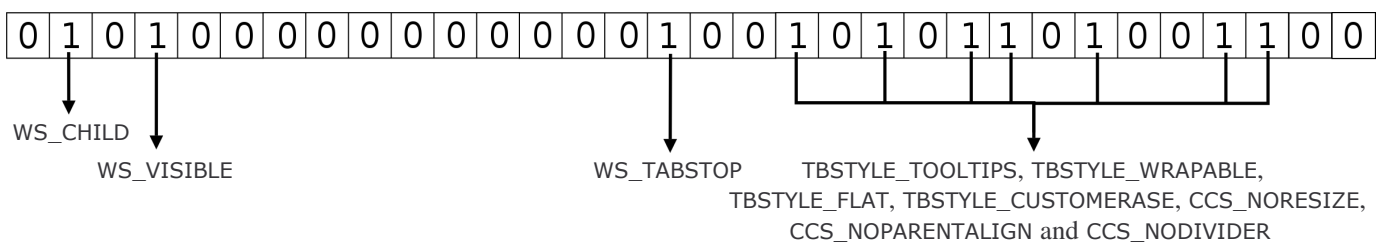
```

r ← new FileReader (f)
lnr ← new LineNumberReader (r);
st ← new StreamTokenizer (lnr);
prepare the tokenizer for resource script style tokenizing rules
lnr.setLineNumber(1) /* the default first line is 0 for lnr and 1 for st */
while there is still a stored line number do
  store the resource starting at the next line number
  generate the resource into GrafiXML

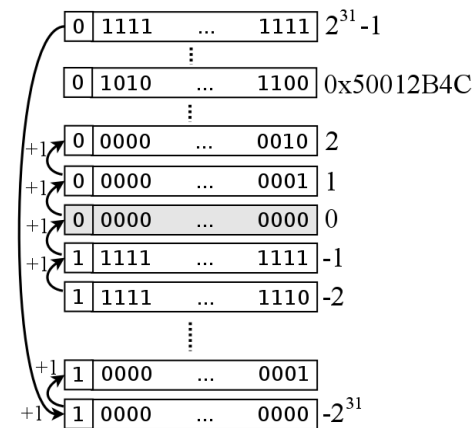
```

The two modules correspond to a physical partition into two main methods `scan (File f)` and `process(int[] a)` from the `RcFile` class.

So, for each object, in particular for each dialog box and controls defined inside a dialog box template, I first read and store the information which concerns it into variables that hold a primitive value (as a position or a dimension number) or a reference to a `String` object (as an id or a class name for a control). It's only at the end of a loop (of a nested loop for controls) that a method charged to generate the object into `GrafiXML` is called. I've not decided to make a transformation as soon as a data is read in the file. To store the style flags, each flag is first translated in its numeric representation and is then added to a single variable of type integer. In fact, as mentioned in the chapter 4, a flag corresponds to a reserved bit (or to a group of bit for some mutually exclusive flags) in a word of 32 bits. Each set of style flags and extended style flags of a dialog box or a control from a particular class is represented by a word in memory. To illustrate, let's look at the style of the tool bar from my fifth example:



The variable I use to store such number is of the signed Java data type `int`, meaning that 32 bits are used to represent the value in two's complement binary form. Note that beyond⁸⁵ a certain threshold the resulting stored number will be not the same that the original hexadecimal number read in the file: this type of primitive data can represent only 2^{31} positive integers, and not 2^{32} as desired. This is not a problem since it's the position of a bit that is significant, which is not altered considering the linear logic of the two's complement representation of numbers: if we add 1 to the highest value that can be represented (overflow), we get the lowest value (if signed numbers were represented for example with a bit of sign, we would get the second representation of the number 0, then -1 and so on).



My basic motivation to group all flags in a numeric value comes from the fact that a style in a resource file for a particular control is often specified with identifiers, less often with a hexadecimal number, but sometimes both. And in the last case, it is not naturally duplicated information. The two formats are additive: the number represents the other flags that are not textually specified whose numeric values are added to form this number. Concerning the style illustrated above, Resource Hacker gives `0x50012B4C` when Resource Tuner gives `WS_TABSTOP` (`0x00010000`) and `0x00002B4C`, plus implicitly `WS_CHILD` (`0x40000000`) and `WS_VISIBLE` (`0x10000000`) seeing that it uses of the shortcut notation.

To know if a flag used in my derivation rules is specified for a particular control or dialog box, all I have to do is to check if an individual bit is positioned to 1 at a specific position. The following is my method which read bits by using some Java bitwise operators. For example, `readBits(x, 0, 3)` returns the number formed by tree first bits of `x` ($n > 1$ is used when mutual exclusive flags are specified by a group of bits), and `readBits(x, 5, 1)` returns 1 if the bit at the fifth position is 1 or 0 if this bit is 0.

```
// return the n bits starting at the position p in x (0 ≤ p ≤ 31 and n ≤ 32 - p)
// <<  shift bits left, filling in with zeros
// >>> shift bits right, filling in with zeros
// &    bitwise AND
// ~    bitwise complement (prefix unary operator)
public int readBits(int x, short p, short n){
    return (x >>> p+1-n) & ~(~0 << n);
}
```

The expression `(x >>> p+1-n)` shifts the selected bits to the right of the word. `~0` is a word whose bits are set to 1. Once shifted to the left with `~0 << n`, its `n` right bits are set to 0. The complement of this word (`~(~0 << n)`) gives a mask whose `n` right bits are set to 1.

There are 42 different style flags for a dialog box that can appear after the `STYLE` keyword. For a dialog box containing 10 controls each having 5 flag identifiers to define its style from a set of more than possible 200 flag⁸⁶, there will be in the worst case more than $10 \times 5 \times 200 = 10000$ strings comparisons needed to store the style of all the controls in a numeric form. Even if I compare the flag string with a smaller set knowing to which class the control belong, the numerous remaining comparisons in many nested if statements will be executed. Another algorithm has to be used.

⁸⁵ That is, when the flag `WS_POPUP`, the style for a dialog box, is present (= the highest bit).

⁸⁶ I have listed in chapter four 15 style flags for a dialog box (`DS_`), 27+25 style flags for a window (`WS_` and `WS_EX`) and 258 style flags for a control.

I use a **trie** for flag identifiers reading in order to support fast pattern matching. When a flag is encountered (e.g. `WS_CAPTION`), a query is performed on a fixed memorised collection of possible flag.

A trie is a tree-based data structure that store strings. It is typically used to see if a word belongs or not to a set of words contained in a text (the name “trie” come from the word “retrieval”). One primary application for tries is to determine if a given pattern matches one of the words of a text exactly (that is, word matching), but with a simple extension prefix matching queries can be performed. My problem can be saw as a string searching (the flag just read in the file) in a text composed of the set S of all the flag identifiers listed in section 4.3. When the word matching occurs, the numeric value of the flag is returned.

I use a standard tries which is an ordered tree T with the following properties:

- Each node of T (except the root) is labelled with a character of the alphabet $\{A;Z\} \cup \{3, _ \}$.
- The ordering of the children of an internal node of T is determined using the lexicographical convention.⁸⁷
- T has s external nodes (leaf), each uniquely associates with a string of S : the concatenation of the labels of the nodes on the path from the root to an external node yields a string of S . T has then s external nodes. Note that the trie stores the common prefixes that exist among the set of the flag string.

This assumes that in S no string is a prefix of another string. This is not the case for the following flag:

`DS_CENTER` and `DS_CENTERMOUSE`
`WS_CHILD` and `WS_CHILDWINDOW` (here not important because these flags are synonyms)
`WS_MAXIMIZEBOX` and `WS_MAXIMIZE`
`WS_MINIMIZEBOX` and `WS_MINIMIZE`
`WS_OVERLAPPED` and `WS_OVERLAPPEDWINDOW`
`WS_POPUP` and `WS_POPUPWINDOW`
`WS_TILED` and `WS_TILEDWINDOW`
`WS_EX_LEFT` and `WS_EX_LEFTSCROLLBAR`
`WS_EX_RIGHT` and `WS_EX_RIGHTSCROLLBAR`
`BS_LEFT` and `BS_LEFTTEXT`
`BS_RIGHT` and `BS_RIGHTBUTTON`
`CBS_DROPDOWN` and `CBS_DROPDOWNLIST`
`MCS_NOTODAY` and `MCS_NOTODAYCIRCLE`
`SBS_SIZEBOX` and `SBS_SIZEBOXBOTTOMRIGHTALIGN`
`SBS_SIZEBOX` and `SBS_SIZEBOXTOPLEFTALIGN`
`SS_CENTER` and `SS_CENTERIMAGE`
`SS_LEFT` and `SS_LEFTNOWORDWRAP`
`SS_RIGHT` and `SS_RIGHTJUST`
`TCS_RIGHT` and `TCS_RIGHTJUSTIFY`

I can then satisfy this assumption by adding a special character that is not in the original alphabet, for example `#`⁸⁸, at the end of each flag string. An internal node can then have in theory between 1 and 29 children (the size of the alphabet), and the height of T is equals to the length of the longest string in $S + 1$.

⁸⁷ $3 < A < Z < _$

⁸⁸ $\# < 3$

- s is not in the trie, but a prefix of s corresponds to a flag already inserted in the trie. We stop at an external node v before reaching the end of s . The chain of nodes is continued after v to store the remaining characters of s , the last contains val .
- s is not already in the trie and we stop at an internal node v before reaching the end of s . A new chain of nodes descendents of v is created to store the remaining characters of s , the last contains val .

It still remains potential space inefficiency in this scheme: the tree will often terminate with long branches of linearly arranged nodes because common prefix of the style flags are usually short. Indeed, there is no need to continue to access the character stored in the nodes to compare with those of the input string when we reach a certain level of the tree, the matching flag can already be identified. When each node (except a leaf) of a subtree has only one child, only the top node (the root of this subtree) should be conserved and the existence of other nodes being connected below is a waste. But care must be taken if a node between the root of this subtree and the leaf store a flag value different from zero: all its ancestors⁹⁰ must also be conserved. If not, a flag representation in the trie is lost and an erroneous value will be returned (the value of the flag having same prefix) when searching the lost flag. To compress the trie, we could use an iterative algorithm. For each node, starting from a leaf, we suppress the node if its parent has just one child and store zero as flag value, or else we set its flag value to the one stored by the initial leaf. But this supposes to use for example a preorder traversal to access each external node. We must also add a new attribute to each node storing the reference to its parent. I use preferably the following recursive algorithm, performed on the trie (after having added all the necessary flag strings) by calling `compress(trie.root)` :

Algorithm `compress(u)` :

```

for each child  $v$  of  $u$  do
  if isInternal(v) then
     $w \leftarrow v$ 
    while isInternal(w) and  $w$  has only one child and  $w.flagValue = 0$  do
       $w \leftarrow child(w)$ 
    if isExternal(w) then
      /*  $w$  is a leaf, and the chain of nodes after  $v$  consists of redundant nodes */
       $v.flagValue \leftarrow w.flagValue()$ 
      supress(child(v))
    else
      /*  $w$  has several children or  $child(w).flagValue \neq 0$  */
      recursively compress the subtree rooted at  $w$  by calling compress(w)

```

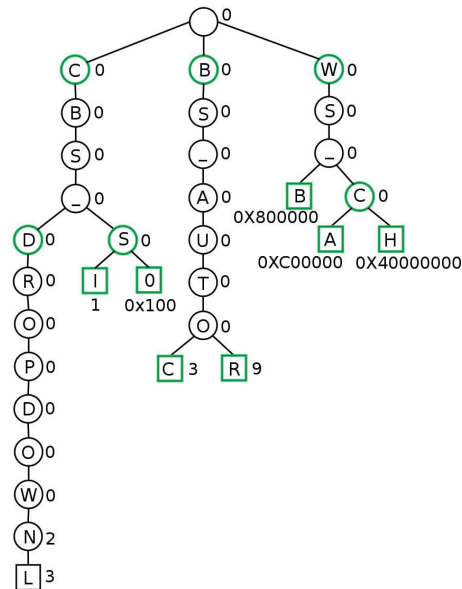
When constructing the trie, it is important to also insert flags which are not in my correspondence rules but that have as prefix a string in the compressed trie. If we forget them, some parsed flag in the resource file that should normally not be taken into consideration could give the value of another flag stored in the trie. Below is the implemented trie for the same above-mentioned set, without `SS_LEFT` (because its numeric value is 0) and enlarged with `CBS_SORT` to not get the value of `CBS_SIMPLE` when matching this flag. The nodes are augmented with indications of the stored numeric value. The matching will be made when we reach the end of the input string f or when a leaf is reached.

⁹⁰ In the terminology, an ancestor of a node is either the node itself or an ancestor of the parent node.


```

Algorithm match(f) :
  v ← trie.root
  i ← 0
  repeat
    success ← false
    for each child w of v do
      if w.label = f[i] then
        v ← w
        success ← true
        i ← i + 1
      break out of the for loop
  until i = f.length() or not success
  return v.flagValue

```



Remark: In the shortcut notation, CBS_SIMPLE (binary value 01) is a default flag for a control of type COMBOBOX. I suppose that this default can be automatically inhibited by the RC compiler if CBS_DROPDOWN (binary value 10) or CBS_DROPDOWNLIST (binary value 11) are specified (the 3 flags are mutual exclusive and grouped in two first bits), because I've remarked (in the .rc files relative to my first example) that Restorator specify only CBS_DROPDOWN (should add the binary value 10 to the default, that gives the value of CBS_DROPDOWNLIST) while Resource Tuner specify NOT CBS_SIMPLE | CBS_DROPDOWN (add 10 to the default and subtract 01, that gives 10). A corrective operation must subtract 1 to the style if NOT CBS_SIMPLE is not specified in the control definition line. For the others default flags (see section 4.3.2), there is no problem because they are not mutually exclusive with other flags and the only way to cancel their effect is to use the NOT keyword (that is, one bit is reserved to them, they are not part of group of bits). But the simpler solution is to use the OR bitwise operation instead of an addition (as probably the RC compiler does). If the bit is not already set to 1, it is positioned.

When a control is defined with the shortcut notation, the algorithm that I used to identifying the control type is much simpler that with the style flags. But if we look at it, is not very different in the sense that as soon as the current checked character is different to the characters of the other words of the same set and at the same position, we know which flag is in question :

```

token ← nextToken()
case token[0]
  is 'A' : if token[4] = 'C' then type ← "AUTOCHECKBOX" else type ← "AUTORADIOBUTTON"
  is 'C' : if token[1] = 'H' then type ← "CCHECKBOX"
           elseif token[1] = 'O' then type ← "COMBOCBOX" else type ← "CTEXT"
  is 'D' : type ← "DEFPUSHBUTTON"
  is 'E' : type ← "EDITTEXT"
  is 'G' : type ← "GROUPBOX"
  is 'I' : type ← "ICON"
  is 'L' : if token[1] = 'T' then type ← "LTEXT" else type ← "LISTBOX"
  is 'R' : if token[1] = 'A' then type ← "RADIOBUTTON" else type ← "RTEXT"
  is 'P' : type ← "PUSHBUTTON"
  is 'S' : type ← "SCROLLBAR"

```

Such conditional statement becomes feasible seeing that only 15 different class names can match with the string read from the text. Once the type identified, I know the class of the control and the flags and the style flags to add to the style variable. The is treated

Each resource of type dialog box is then parsed and stored in variables, even the style flags (converted in numeric). Each control expressed in the shortcut notation is treated as if it was defined with the generic notation (the default flags are added in addition to those specified to the value storing the styles before the transformation in GrafiXML is processed). I call then a method to apply my derivation rules and to generate into GrafiXML the corresponding objects. This method is in a separated class which uses the package `be.ac.ucl.isys.grafixml.gui.editor`⁹¹ containing the methods needed to create and manage CIOs. The entire interface with the GrafiXML's methods is confined in one specific class, which will facilitate future modifications. The resources of type menu are easier to parse and to generate.

⁹¹ See <http://www.usixml.org/javadocs/grafixml/be/ac/ucl/isys/grafixml/gui/editor/package-summary.html>.

7 Conclusion

My thesis is focused to the reverse engineering of a given Windows resource file. The scope of the analysis is then limited to Windows user interfaces. I could have iterated the method used with other types of resource file, as for example with Apple resource files⁹².

There is a possible loss of information between the original interface perceived by the user and the interface described only by a resource script file (without looking at source code of the application). The file used contains only text resources (principally dialog boxes and menus). The non text resources appended to an application are stored in separated binary files. Icons are examples of loss at the presentation level since the images themselves are not available (a binary file is just referenced in the definition of such resources) and their size often defines the place taken in the dialog box. Text information can also be missing, such as items contained in a list box for example. Concerning the dynamic aspects of the interaction, the transition between a menu and a dialog box and the navigation among dialog boxes are not specified. Moreover, the input of my method relies on the specific implementation of a decompilation tool used to generate the file.

The UsiXML specification language has been chosen to express the abstract representation of a user interface. This language can specify a set of models defining the user interface at multiple levels of abstraction. This language support also a multi-path user interface development: the development process can be initiated from any level of abstraction. The goal of my reverse engineering method was to recuperate an existing user interface and to produce directly a model specifying the user interface at a level of abstraction dependent of the modality of interaction but independent of the platform. This model which allows capturing the appearance and behaviour of a graphical interface (the Concrete User Interface) can be reused by an ulterior forward engineering step to generate new code in another specific language.

There is a second loss of information in the reverse engineering process when going from a resource file to the CUI model. We cannot come to the conclusion when analysing correspondences that the language specific to the resources is covered by UsiXML. Some properties of elements composing a Windows user interface could still be generalised. The UsiXML language is still currently in development and may then evolve in the future. In addition, the layout defined for a dialog box is different (box mechanism instead of relative coordinates) which complicates its exact restitution.

⁹² But documentation is turned out to be more difficult to find (on the web for example), and the format of such files cannot be obtained by decompilation using a PC.

This work is concretized by the implementation in Java of a reverse engineering tool integrated into GrafiXML and capable of extracting the model from a given resource file. Unfortunately, algorithm for the construction of boxes will be not implemented. This evolution can be imagined as a future work. A simple solution at the present time is to let the user draw manually the boxes in the reproduced dialog box (the relative coordinates can be still used to generate object in the composer of GrafiXML). The layout will be then automatically specified in UsiXML in the XML editor. The current implementation makes also the assumption that each resource is separated by an empty line in the resource file (as it's the case in files generated by the proposed decompilation tools). This choice has been made for efficacy reasons, but this detail can be easily modified if needed.

My tool has also been implemented in a perspective of reusability. If my transformation rules are modified or if the implementation of GrafiXML⁹³ and the UsiXML language evolve, a new Windows style flag for instance can be easily considered by creating a new tree data structure storing flags used in my transformation rules. The interface with the GrafiXML program is also encapsulated in a separated class enabling to implement a new rule (using for example the new flag given in the parameters) without having to understand my own code.⁹⁴ I think this work constitute a good basis if it is carried out. This thesis provides also a good documentation of the source language.

⁹³ This tool is still in work in progress.

⁹⁴ The plug-in and its source code will be available on <http://www.usixml.org>.

Bibliography

1. M.M. Moore, S. Rugaber, P. Seaver, Knowledge Based User Interface Migration, in Proceedings of the 1994 International Conference on Software Maintenance (Victoria, British Columbia, September 1994).
2. Merlo E., Gagné P.Y., Thiboutôt A., Inference of Graphical AUIDL Specifications for the Reverse Engineering of User Interfaces, Proc. International Conference on Software Maintenance, Victoria, BC, Canada, Septembre 19-23, 1994, pp.80-88Byrne, E., "A Conceptual Foundation for Software Re-Engineering," The International Conference on Software Maintenance 1992, pp. 226-235.
3. E. Stroulia, M. El-Ramly, P. Iglinski, P. Sorenson: User Interface Reverse Engineering in Support of Interface Migration to the Web, Automated Software Engineering Journal, 2003, Kluwer Academic PublishersBoehm B., "Software Engineering Economics", Prentice Hall, 1981.
4. A. Memon, I. Banerjee and A. Nagarajan. GUI Ripping: Reverse Engineering of Graphical User Interface for Testing. Proc. of Working Conference on Reverse Engineering, November 2003.
5. Byrne. E., "A Conceptual Foundation for Software Re-Engineering." The International Conference on Software Maintenance 1992, pp. 226-235.
6. Boehm, B. W. (1981). Software Engineering Economics. Englewood Cliffs, Prentice-Hall, Inc.
7. Boehm, B. W., E. Horowitz, R. Madachy, D. Reifer, B. K. Clark, B. Steece, A. W. Brown, S. Chulani and C. Abts (2000). Software Cost Estimation with COCOMO II. NJ, Prentice Hall.
8. Sneed, H. M. (2004). A Cost Model for Software Maintenance and Evolution. 20th IEEE International Conference on Software Maintenance (ICSM'04), Chicago, Illinois, September 11 - 14, pp 264-273.
9. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Víctor López Jaquero, UsiXML: a Language Supporting Multi-Path Development of User Interfaces, Proc. of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems, EHCI-DSVIS'2004 (Hamburg, July 11-13, 2004).
10. Vanderdonckt, J., Limbourg, Q., Michotte, B., Bouillon, L., Trevisan, D., Florins, M., UsiXML: a User Interface Description Language for Specifying Multimodal User Interfaces, in Proc. of W3C Workshop on Multimodal Interaction WMI'2004 (Sophia Antipolis, 19-20 July 2004).
11. L.Bouillon, Q.Limbourg, J.Vanderdonckt, B.Michotte, Reverse Engineering of Web Pages based on Derivations and Transformations, in Proceedings of LAWEB 2005 (Buenos Aires, 31 Oct.-2 Nov., 2005), IEEE Computer Society Press, Los Alamitos, 2005, pp. 3-13.
12. Limbourg, Q., Vanderdonckt, J., Michotte, B., Bouillon, L., Florins, M., Trevisan, D., UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces, in Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" (Gallipoli, May 25, 2004), Luyten, K., M. Abrams, Limbourg, Q., Vanderdonckt, J. (Eds.), Gallipoli, 2004, pp. 55-62.

13. Limbourg, Q., Vanderdonckt, J., UsiXML: A User Interface Description Language Supporting Multiple Levels of Independence, in Matera, M., Comai, S. (Eds.), "Engineering Advanced Web Applications", Rinton Press, Paramus, 2004
14. Vanderdonckt, J., A MDA-Compliant Environment for Developing User Interfaces of Information Systems, Proc. of 17th Conf. on Advanced Information Systems Engineering CAiSE'05 (Porto, 13-17 June 2005), O. Pastor & J. Falcão e Cunha (eds.), Lecture Notes in Computer Science, Vol. 3520, Springer-Verlag, Berlin, 2005, pp. 16-31.
15. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computers. Vol. 15, No. 3, June 2003, pp. 289-308.
16. Bouillon L., Vanderdonckt J., User Interface Reverse Engineering, Proc. of 2nd Int. Conf. on Universal Access in Human-Computer Interaction UAHCI'2003 (Crete, 22-27 June 2003), Vol. 4, Stephanidis, C. (Eds.), Lawrence Erlbaum Associates, Mahwah, 2003, pp. 1509-1513.
17. Goodrich, M.T. et R. Matassia, Data Structures and Algorithm in Java (2nd. Edition), John Wiley & Sons, 2001.
18. Petzold, C., Programming Windows (5th. Edition), Microsoft Press, 1998.
19. Georgia Tech's reverse engineering group, Georgia tech, 30 apr 2001, available at <http://www.cc.gatech.edu/reverse/>
20. Wiki page of UsiXML, available at <http://www.usixml.org/?view=wiki/>
21. Documentation of resources on the MSDN Library, available at <http://msdn.microsoft.com/library/>
22. Apple Human Interface Guidelines, available at <http://developer.apple.com/documentation/>
23. XML Markup Languages for User Interface Definition, available at <http://www.oasis-open.org/cover//userInterfaceXML.html>
24. XML tutorial, available on http://www.w3schools.com/xml/xml_what.asp
26. BCHI-ISYS research on reverse engineering, available on <http://www.isys.ucl.ac.be/bchi/research/vaquita.htm>
27. State of the art on reverse engineering and transcoding of UIs, available on <http://www.isys.ucl.ac.be/bchi/research/soare.htm>

Appendix

A

Example of complete resource file

There is an example of file (ALL_RESOURCES.rc) containing all the resources from my first example (SciTE.exe) generated by Resource Hacker. This is the ACSI resource script that can be saved by a decompiler, not the binary compiled resource file (*.res).

```
100 BITMAP "Bitmap_1.bmp"
SCITE MENU
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
{
  POPUP "&File"
  {
    MENUITEM "&New\Ctrl+N", 101
    MENUITEM "&Open... \Ctrl+O", 102
    MENUITEM "Open Selected filename \Ctrl+Shift+O", 103
    MENUITEM "&Revert\Ctrl+R", 104
    MENUITEM "&Close\Ctrl+W", 105
    MENUITEM "&Save\Ctrl+S", 106
    MENUITEM "Save &as... \Ctrl+Shift+S", 110
    MENUITEM "Save a Copy... \Ctrl+Shift+F", 116
    POPUP "Encoding"
    {
      MENUITEM "0. 8bit", 150
      MENUITEM "UCS-2 &Big Endian", 151
      MENUITEM "UCS-2 &Little Endian", 152
      MENUITEM "UTF-8", 153
      MENUITEM "UTF-8 &Unicode", 154
    }
  }
  POPUP "&Export"
  {
    MENUITEM "As &HTML", 111
    MENUITEM "As &RTF", 112
    MENUITEM "As &PDF...", 113
    MENUITEM "As &XML...", 115
    MENUITEM "As &XML...", 117
  }
  MENUITEM_SEPARATOR
  MENUITEM "Page Setup...", 130
  MENUITEM "&Print\Ctrl+P", 131
  MENUITEM_SEPARATOR
  MENUITEM "&Load Session...", 132
  MENUITEM "Save Session...", 133
  MENUITEM_SEPARATOR
  MENUITEM "E&xit", 140
}
POPUP "&Edit"
{
  MENUITEM "&Undo\Ctrl+Z", 201
  MENUITEM "&Redo\Ctrl+Y", 202
  MENUITEM_SEPARATOR
  MENUITEM "Cu&t\Ctrl+X", 203
  MENUITEM "&Copy\Ctrl+C", 204
  MENUITEM "&Paste\Ctrl+V", 205
  MENUITEM "&Delete\Del", 206
  MENUITEM "Select &All\Ctrl+A", 207
  MENUITEM "Copy as RT&F", 245
  MENUITEM_SEPARATOR
  MENUITEM "Match &Brace\Ctrl+E", 230
  MENUITEM "Select the Brace\Ctrl+Shift+E", 231
  MENUITEM "Show Callip\Ctrl+Shift+Space", 232
  MENUITEM "Complete S&ymbol\Ctrl+T", 233
  MENUITEM "Complete &Word\Ctrl+Enter", 234
  MENUITEM "&Expand Abbreviation\Ctrl+E", 242
  MENUITEM "&Insert Abbreviation\Ctrl+Shift+R", 247
  MENUITEM "Block Comment or Uncomment\Ctrl+Q", 243
  MENUITEM "Boxx Comment\Ctrl+Shift+B", 246
  MENUITEM "Stream Comment\Ctrl+Shift+Q", 244
  MENUITEM "Make &Selection Uppercase\Ctrl+Shift+U", 240
  MENUITEM "Make Selection &Lowercase\Ctrl+U", 241
  POPUP "Paragraph"
  {
    MENUITEM "&Join", 248
    MENUITEM "&Split", 249
  }
  POPUP "&Search"
  {
    MENUITEM "&Find... \Ctrl+F", 210
    MENUITEM "Find &Next\F3", 211
    MENUITEM "Find Previous\Ctrl+Shift+F3", 212
    MENUITEM "Bind in Files... \Ctrl+Shift+F", 215
    MENUITEM "Replace... \Ctrl+H", 216
    MENUITEM "Incremental Search... \Ctrl+Alt+I", 252
    MENUITEM_SEPARATOR
    MENUITEM "&Go to... \Ctrl+G", 220
    MENUITEM "Next Bookmark\F2", 221
    MENUITEM "Previous Bookmark\Ctrl+Shift+F2", 223
    MENUITEM "Toggle Bookmark\Ctrl+F2", 222
    MENUITEM "&Clear All Bookmarks", 224
  }
  POPUP "&View"
  {
    MENUITEM "Toggle &current field", 235
    MENUITEM "Toggle &all fields", 236
  }
  MENUITEM_SEPARATOR
  MENUITEM "Full Screen\F11", 961
  MENUITEM "&Tool Bar", 408
  MENUITEM "&Tab Bar", 410
  MENUITEM "&Status Bar", 411
  MENUITEM_SEPARATOR
  MENUITEM "&Win-ops\Ctrl+Shift+8", 402
  MENUITEM "&End of Line\Ctrl+Shift+9", 403
  MENUITEM "&Line Indentation Guide", 404
  MENUITEM "&Line Indent", 407
  MENUITEM "&Fold Margin", 405
  MENUITEM "&Output\F6", 409
  MENUITEM "&Parameters\Ctrl+F8", 412
}
POPUP "&Tools"
{
  MENUITEM "&Compile\Ctrl+F7", 301
  MENUITEM "&Build\F7", 302
  MENUITEM "&Go\F5", 303
  MENUITEM "&Stop Executing\Ctrl+Break", 304
  MENUITEM_SEPARATOR
  MENUITEM "&Next Message\F4", 306
  MENUITEM "&Previous Message\Ctrl+Shift+F4", 307
  MENUITEM "Clear &Output\Shift+F5", 420
  MENUITEM "&Switch Pane\Ctrl+F6", 421
}
POPUP "&Options"
{
  MENUITEM "&Always On Top", 960
  MENUITEM "Open Files &Here", 413
  MENUITEM "Vertical &Split", 401
  MENUITEM "&Wrap", 414
  MENUITEM "Wrap &Output", 415
  MENUITEM "&Read-Only", 416
  MENUITEM_SEPARATOR
  POPUP "&Line End Characters"
  {
    MENUITEM "CR & LF", 430
    MENUITEM "&CR", 431
    MENUITEM "&LF", 432
  }
  MENUITEM "&Convert Line End Characters", 433
}
}
```

```

CONTROL "Regular &expression",239,BUTTON,BS_AUTOCHECKBOX|WS_CHILD|WS_VISIBLE|WS_GROUP|WS_TABSTOP,
5,46,120,10
CONTROL "Wrap around",240,BUTTON,BS_AUTOCHECKBOX|WS_CHILD|WS_VISIBLE|WS_GROUP|WS_TABSTOP,5,58,130,10
CONTROL "Transform &backslash expressions",241,BUTTON,BS_AUTOCHECKBOX|WS_CHILD|WS_VISIBLE|WS_GROUP|
WS_TABSTOP,5,70,160,10
CONTROL "Direction",-1,BUTTON,BS_GROUPBOX|WS_CHILD|WS_VISIBLE|WS_GROUP,135,22,60,34
CONTROL "Up",234,BUTTON,BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE|WS_GROUP,140,30,45,12
CONTROL "Down",235,BUTTON,BS_AUTORADIOBUTTON|WS_CHILD|WS_VISIBLE,140,42,45,12
CONTROL "Find Next",-1,BUTTON,BS_DEFPUSHBUTTON|WS_VISIBLE|WS_GROUP|WS_TABSTOP,205,5,65,14
CONTROL "Mark All",245,BUTTON,BS_PUSHBUTTON|WS_VISIBLE|WS_TABSTOP,205,23,65,14
CONTROL "Cancel",-2,BUTTON,BS_PUSHBUTTON|WS_VISIBLE|WS_TABSTOP,205,41,65,14

GREF_DIALOG 26,41,355,61
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Find in Files"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Sans Serif"
{
CONTROL "Find what:",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,5,7,40,8
CONTROL "",222,COMBOBOX,CBS_DROPDOWN | CBS_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_TABSTOP,50,5,245,50
CONTROL "Fishes",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,5,25,40,8
CONTROL "",223,COMBOBOX,CBS_DROPDOWN | CBS_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_TABSTOP,50,23,245,50
CONTROL "SDirectory",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,5,43,40,8
CONTROL "",224,COMBOBOX,CBS_DROPDOWN | CBS_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_TABSTOP,50,41,230,50
CONTROL "SFind",-1,BUTTON,BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,301,5,50,14
CONTROL "Cancel",-2,BUTTON,BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,301,23,50,14
CONTROL "",249,BUTTON,BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,285,41,10,14
CONTROL "SBrowse...",243,BUTTON,BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,301,41,50,14
}

INSABBREV_DIALOG 26,41,181,45
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Insert Abbreviation"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Sans Serif"
{
CONTROL "SAbbreviation",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,5,8,65,8
CONTROL "",244,COMBOBOX,CBS_DROPDOWN | CBS_AUTOHSCROLL | WS_VISIBLE | WS_CHILD | WS_VISIBLE | WS_TABSTOP,70,6,106,88
CONTROL "SInsert",-1,BUTTON,BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,70,26,50,14
CONTROL "Cancel",-2,BUTTON,BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,126,26,50,14
}

PARAMETERS_DIALOG 26,41,130,100
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Parameters"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Sans Serif"
{
CONTROL "",242,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,5,5,120,8
CONTROL "S1",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,5,20,8,8
CONTROL "300,EDIT,ES_LEFT | ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,15,16,110,12
CONTROL "S2",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,5,35,8,8
CONTROL "301,EDIT,ES_LEFT | ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,15,33,110,12
CONTROL "S3",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,5,50,8,8
CONTROL "302,EDIT,ES_LEFT | ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,15,48,110,12
CONTROL "S4",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,5,65,8,8
CONTROL "303,EDIT,ES_LEFT | ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,15,63,110,12
CONTROL "SExecute",-1,BUTTON,BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,6,80,55,14
CONTROL "Cancel",-2,BUTTON,BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,70,80,55,14
}

PARAMETERSNONMODAL_DIALOG 26,41,130,85
STYLE WS_POPUP | WS_CAPTION | WS_SYSMENU

```

```

MENUITEM SEPARATOR
MENUITEM "Change Indentation Settings...{Ctrl+Shift+I}", 440
MENUITEM "Use &Monospaced Font{Ctrl+F11}", 450
MENUITEM SEPARATOR
MENUITEM "Open Local &Options File", 460
MENUITEM "Open &Next Options File", 461
MENUITEM "Open &Global Options File", 462
MENUITEM "Open &Abbreviations File", 463
MENUITEM "Open &Lua Startup Script", 464
MENUITEM SEPARATOR
POPUP "&Language"
{
MENUITEM SEPARATOR
POPUP "&Buffers"
{
MENUITEM "SPrevious{Shift+P}", 501
MENUITEM "SNext{N}", 502
MENUITEM "SClose All", 503
MENUITEM "SSave All", 504
}
POPUP "&Help"
{
MENUITEM "SHelp{F1}", 901
MENUITEM "SGetE Help", 903
MENUITEM "SAbout ScTE", 902
}
}
ABOUT_DIALOG 26,41,350,242
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "About ScTE"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Sans Serif"
{
CONTROL "",221,"Scintilla",0x50000000,1,1,346,218
CONTROL "ScTE",-1,STATIC,SS_ICON | WS_CHILD | WS_VISIBLE,1,221,32,32
CONTROL "OK",-1,BUTTON,BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,26,222,32,20
}

GOLINE_DIALOG 26,41,240,50
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Go To"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Sans Serif"
{
CONTROL "SDestination Line",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,5,8,70,8
CONTROL "220,EDIT,ES_LEFT | ES_NUMBER | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,5,8,70,8
CONTROL "SColumn",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP,115,8,35,8
CONTROL "246,EDIT,ES_LEFT | ES_NUMBER | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP,150,6,30,12
CONTROL "Current Line",-1,STATIC,SS_LEFT | WS_VISIBLE,5,21,50,8
CONTROL "1234567",225,STATIC,SS_LEFT | WS_VISIBLE,5,21,30,8
CONTROL "Column",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE,115,21,35,8
CONTROL "1234567",247,STATIC,SS_LEFT | WS_VISIBLE,150,21,30,8
CONTROL "Last Line",-1,STATIC,SS_LEFT | WS_CHILD | WS_VISIBLE,5,34,50,8
CONTROL "1234567",226,STATIC,SS_LEFT | WS_VISIBLE,5,34,50,8
CONTROL "8Go To",-1,BUTTON,BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,190,6,45,14
CONTROL "Cancel",-2,BUTTON,BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP,190,26,45,14
}

CONTROL "Find what:",-1,STATIC,SS_LEFT|WS_VISIBLE|WS_GROUP,5,7,45,8
CONTROL "",222,COMBOBOX,CBS_DROPDOWN|CBS_AUTOHSCROLL|WS_VISIBLE|WS_CHILD|WS_VISIBLE|WS_TABSTOP,50,5,145,50
CONTROL "Match whole word only",232,BUTTON,BS_AUTOCHECKBOX|WS_VISIBLE|WS_GROUP|WS_TABSTOP,
5,22,120,10
CONTROL "Match &case",233,BUTTON,BS_AUTOCHECKBOX|WS_VISIBLE|WS_GROUP|WS_TABSTOP,5,34,130,10

```



```

VK_2, 1201, ALT, VIRTKEY
VK_3, 1202, ALT, VIRTKEY
VK_4, 1203, ALT, VIRTKEY
VK_5, 1204, ALT, VIRTKEY
VK_6, 1205, ALT, VIRTKEY
VK_7, 1206, ALT, VIRTKEY
VK_8, 1207, ALT, VIRTKEY
VK_9, 1208, ALT, VIRTKEY
VK_0, 1209, ALT, VIRTKEY
VK_U, 240, CONTROL, SHIFT, VIRTKEY
VK_U, 241, CONTROL, SHIFT, VIRTKEY
VK_8, 402, CONTROL, SHIFT, VIRTKEY
VK_9, 403, CONTROL, SHIFT, VIRTKEY
VK_I, 440, CONTROL, SHIFT, VIRTKEY
VK_F1, 901, VIRTKEY
VK_F2, 221, VIRTKEY
VK_F2, 223, SHIFT, VIRTKEY
VK_F2, 222, CONTROL, VIRTKEY
VK_F2, 225, ALT, VIRTKEY
VK_F2, 226, ALT, SHIFT, VIRTKEY
VK_F3, 211, VIRTKEY
VK_F3, 212, SHIFT, VIRTKEY
VK_F3, 213, CONTROL, VIRTKEY
VK_F3, 214, CONTROL, SHIFT, VIRTKEY
VK_F4, 306, VIRTKEY
VK_F4, 307, SHIFT, VIRTKEY
VK_F4, 105, CONTROL, VIRTKEY
VK_F5, 303, VIRTKEY
VK_F5, 420, SHIFT, VIRTKEY
VK_F6, 501, SHIFT, VIRTKEY
VK_F6, 421, CONTROL, VIRTKEY
VK_F6, 502, VIRTKEY
VK_F6, 421, CONTROL, SHIFT, VIRTKEY
VK_F7, 301, CONTROL, VIRTKEY
VK_F7, 302, VIRTKEY
VK_F8, 409, VIRTKEY
VK_F8, 412, SHIFT, VIRTKEY
VK_F9, 314, SHIFT, VIRTKEY
VK_F9, 313, VIRTKEY
VK_F9, 311, CONTROL, SHIFT, VIRTKEY
VK_F9, 312, CONTROL, SHIFT, VIRTKEY
VK_F11, 961, VIRTKEY
VK_F11, 450, CONTROL, VIRTKEY
VK_TAB, 501, CONTROL, SHIFT, VIRTKEY
VK_TAB, 502, CONTROL, VIRTKEY
VK_BACK, 201, ALT, VIRTKEY
VK_BACK, 202, ALT, SHIFT, VIRTKEY
VK_CANCEL, 304, CONTROL, VIRTKEY
VK_Z, 201, CONTROL, VIRTKEY
VK_X, 203, CONTROL, VIRTKEY
VK_C, 204, CONTROL, VIRTKEY
VK_I, 252, ALT, CONTROL, VIRTKEY
}

SCITE ICON "Icon_1.ico"

1 VERSIONINFO
FILEVERSION 1,6,2,0
PRODUCTVERSION 1,6,2,0
FILEOS 0x40004
FILETYPE 0x1
}
BLOCK "StringFileInfo"
{
}

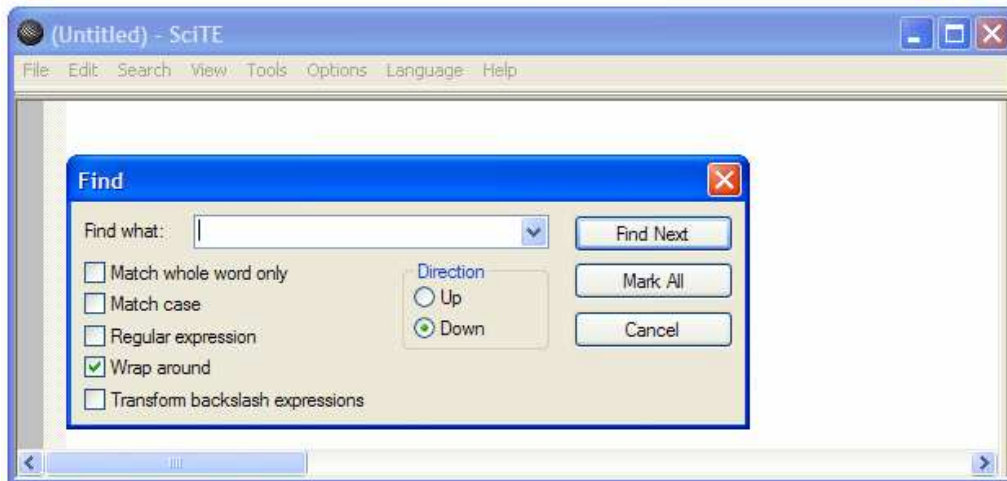
BLOCK "040904b0"
{
  VALUE "CompanyName", "Neil Hodgson neilh@scintilla.org"
  VALUE "FileDescription", "ScITE - a Scintilla based Text Editor"
  VALUE "FileVersion", "1.62"
  VALUE "InternalName", "ScITE"
  VALUE "LegalCopyright", "Copyright 1998-2004 by Neil Hodgson"
  VALUE "OriginalFilename", "ScITE.EXE"
  VALUE "ProductName", "ScITE"
  VALUE "ProductVersion", "1.62"
}
BLOCK "VarFileInfo"
{
  VALUE "Translation", 0x0409 0x04B0
}
}
1.24 "Data_1.bin"
}
}

```

Appendix B

Comparison of the resource files given by the decompilers

First example



Resource Tuner :

```

400 DIALOG 30, 73, 275, 84
STYLE DS_SETFONT | DS_MODALFRAME | DS_3DLOOK | WS_POPUPWINDOW | WS_DLGFRAME
CAPTION "Find"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Shell Dlg"
{
    LTEXT "Fi&nd what:", -1, 5, 7, 45, 8
    COMBOBOX 222, 50, 5, 145, 50, NOT CBS_SIMPLE | CBS_DROPDOWN | CBS_AUTOHSCROLL
    AUTOCHECKBOX "Match &whole word only", 232, 5, 22, 120, 10, WS_GROUP
    AUTOCHECKBOX "Match &case", 233, 5, 34, 130, 10, WS_GROUP
    AUTOCHECKBOX "Regular &expression", 239, 5, 46, 120, 10, WS_GROUP
    AUTOCHECKBOX "Wrap aroun&d", 240, 5, 58, 120, 10, WS_GROUP
    AUTOCHECKBOX "Transform &backslash expressions", 241, 5, 70, 160, 10, WS_GROUP
    GROUPBOX "Direction", -1, 135, 22, 60, 34, WS_GROUP
    AUTORADIOBUTTON "&Up", 234, 140, 30, 45, 12, NOT WS_TABSTOP | WS_GROUP
    AUTORADIOBUTTON "&Down", 235, 140, 42, 45, 12, NOT WS_TABSTOP
    DEFPUSHBUTTON "&Find Next", 1, 205, 5, 65, 14, WS_GROUP
    PUSHBUTTON "&Mark All", 245, 205, 23, 65, 14
    PUSHBUTTON "Cancel", 2, 205, 41, 65, 14
}

```

Restorator :

```

400 DIALOG 30, 73, 275, 84
STYLE DS_SETFONT | DS_MODALFRAME | DS_3DLOOK | WS_POPUPWINDOW | WS_CAPTION
CAPTION "Find"
FONT 8, "MS Shell Dlg"
{
    LTEXT "Fi&nd what:", -1, 5, 7, 45, 8
    COMBOBOX 222, 50, 5, 145, 50, CBS_DROPDOWN | CBS_AUTOHSCROLL
    AUTOCHECKBOX "Match &whole word only", 232, 5, 22, 120, 10, WS_GROUP
    AUTOCHECKBOX "Match &case", 233, 5, 34, 130, 10, WS_GROUP
    AUTOCHECKBOX "Regular &expression", 239, 5, 46, 120, 10, WS_GROUP
    AUTOCHECKBOX "Wrap aroun&d", 240, 5, 58, 120, 10, WS_GROUP
    AUTOCHECKBOX "Transform &backslash expressions", 241, 5, 70, 160, 10, WS_GROUP
    GROUPBOX "Direction", -1, 135, 22, 60, 34, WS_GROUP
    AUTORADIOBUTTON "&Up", 234, 140, 30, 45, 12, WS_GROUP | NOT WS_TABSTOP
    AUTORADIOBUTTON "&Down", 235, 140, 42, 45, 12, NOT WS_TABSTOP
    DEFPUSHBUTTON "&Find Next", 1, 205, 5, 65, 14, WS_GROUP
    PUSHBUTTON "&Mark All", 245, 205, 23, 65, 14
    PUSHBUTTON "Cancel", 2, 205, 41, 65, 14
}

```

Resource Builder :

```

400 DIALOG 30, 73, 275, 84
STYLE DS_3DLOOK | DS_SETFONT | DS_MODALFRAME | DS_CENTER | WS_POPUP | WS_SYSMENU | WS_CAPTION
CAPTION "Find"
FONT 8, "MS Shell Dlg"
LANGUAGE LANG_ENGLISH, 1
BEGIN
CONTROL "Fi&nd what:",65535,"STATIC",SS_LEFT | WS_CHILD | WS_GROUP | WS_VISIBLE ,5,7,45,8
CONTROL "",222,"COMBOBOX",CBS_DROPDOWN | CBS_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_TABSTOP | WS_VISIBLE ,50,5,145,50
CONTROL "Match &whole word only",232,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD | WS_GROUP | WS_TABSTOP | WS_VISIBLE ,5,22,120,10
CONTROL "Match &case",233,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD | WS_GROUP | WS_TABSTOP | WS_VISIBLE ,5,34,130,10
CONTROL "Regular &expression",239,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD | WS_GROUP | WS_TABSTOP | WS_VISIBLE ,5,46,120,10
CONTROL "Wrap aroun&d",240,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD | WS_GROUP | WS_TABSTOP | WS_VISIBLE ,5,58,120,10
CONTROL "Transform &backslash expressions",241,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD | WS_GROUP | WS_TABSTOP | WS_VISIBLE ,5,70,160,10
CONTROL "Direction",65535,"BUTTON",BS_GROUPBOX | WS_CHILD | WS_GROUP | WS_VISIBLE ,135,22,60,34
CONTROL "&Up",234,"BUTTON",BS_AUTORADIOBUTTON | WS_CHILD | WS_GROUP | WS_VISIBLE ,140,30,45,12
CONTROL "&Down",235,"BUTTON",BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE ,140,42,45,12
CONTROL "&Find Next",1,"BUTTON",BS_DEFPUSHBUTTON | WS_CHILD | WS_GROUP | WS_TABSTOP | WS_VISIBLE ,205,5,65,14
CONTROL "&Mark All",245,"BUTTON",BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,205,23,65,14
CONTROL "Cancel",2,"BUTTON",BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,205,41,65,14
END

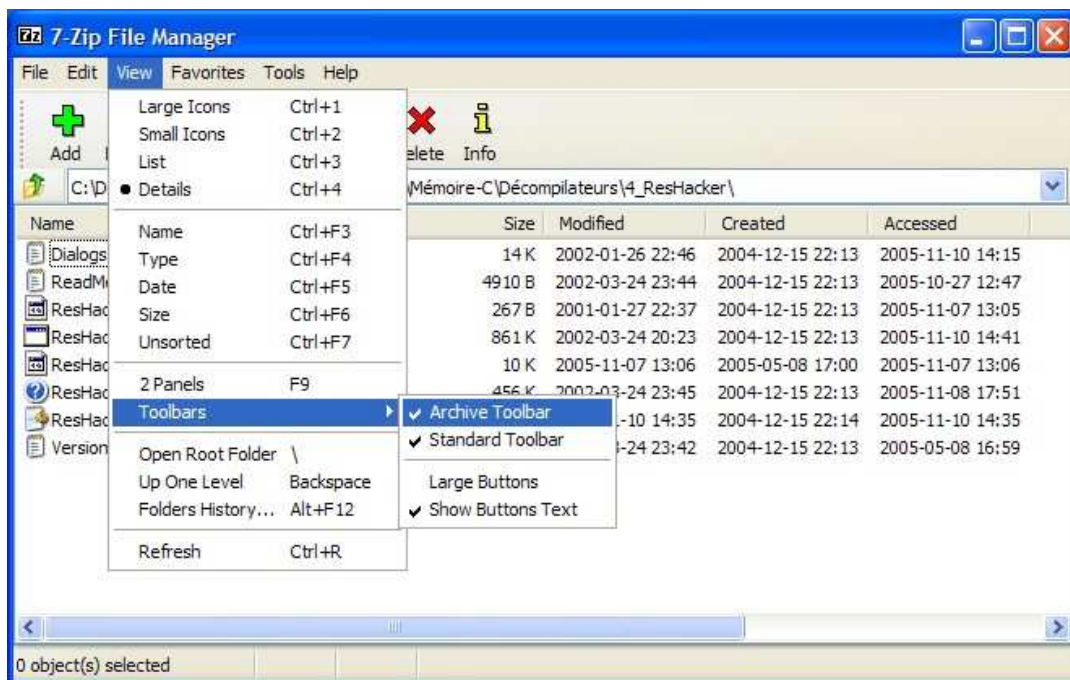
```

Resource Hacker :

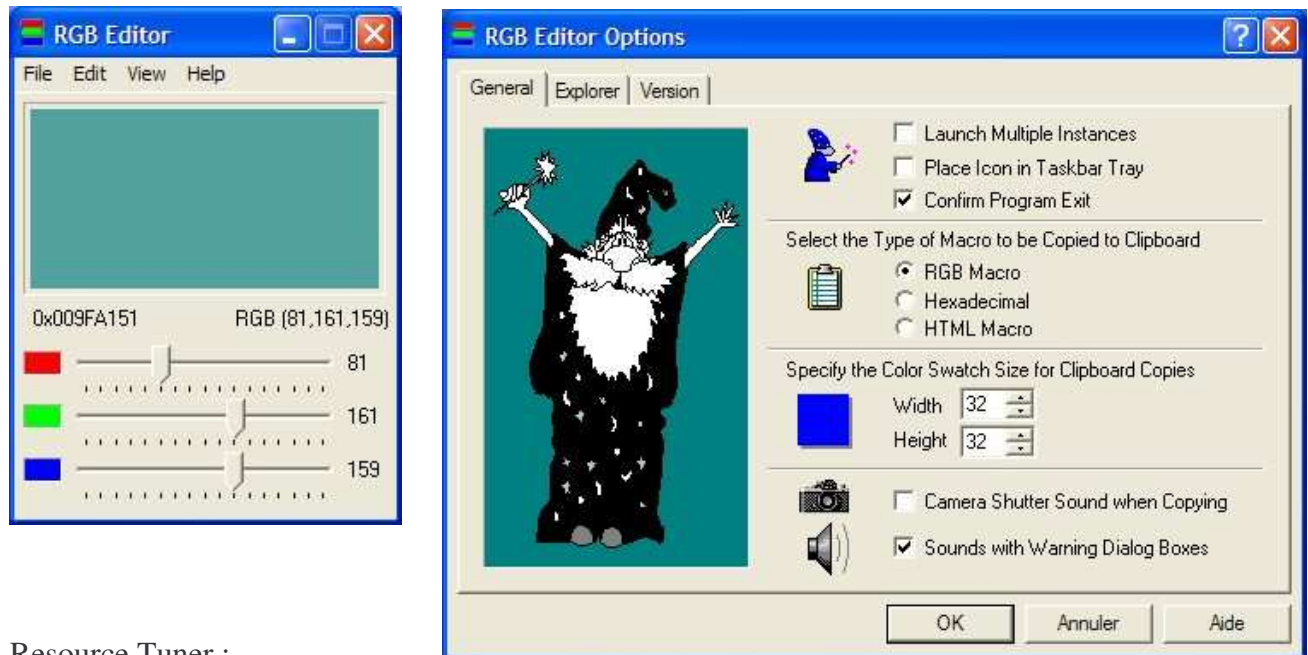
```

400 DIALOG 30, 73, 275, 84
STYLE DS_MODALFRAME | DS_CENTER | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Find"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Shell Dlg"
{
CONTROL "Fi&nd what:", -1, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 5, 7, 45, 8
CONTROL "", 222, COMBOBOX, CBS_DROPDOWN | CBS_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 50, 5, 145, 50
CONTROL "Match &whole word only", 232, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 5, 22, 120, 10
CONTROL "Match &case", 233, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 5, 34, 130, 10
CONTROL "Regular &expression", 239, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 5, 46, 120, 10
CONTROL "Wrap aroun&d", 240, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 5, 58, 120, 10
CONTROL "Transform &backslash expressions", 241, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 5, 70, 160, 10
CONTROL "Direction", -1, BUTTON, BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP, 135, 22, 60, 34
CONTROL "&Up", 234, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP, 140, 30, 45, 12
CONTROL "&Down", 235, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 140, 42, 45, 12
CONTROL "&Find Next", 1, BUTTON, BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 205, 5, 65, 14
CONTROL "&Mark All", 245, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 205, 23, 65, 14
CONTROL "Cancel", 2, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 205, 41, 65, 14
}

```

Second example

All the tools give the same information about the menu (except that Resource Builder use **BEGIN ... END** instead of **{...}**).

Third example

Resource Tuner :

```

1000 DIALOG 0, 0, 320, 172
STYLE DS_SETFONT | DS_MODALFRAME | WS_CAPTION | WS_POPUP
CAPTION "General"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Sans Serif"
{
CONTROL 1999, -1, "STATIC", SS_BITMAP | SS_REALSIZEIMAGE | SS_SUNKEN, 7, 7, 18, 20
ICON 2655, -1, 132, 7, 18, 20
AUTOCHECKBOX "&Launch Multiple Instances", 108, 167, 5, 99, 8
AUTOCHECKBOX "&Place Icon in Taskbar Tray", 111, 167, 17, 99, 8
AUTOCHECKBOX "Confirm Program &Exit", 112, 167, 29, 79, 8
LTEXT -1, 118, 40, 195, 1, NOT WS_GROUP | SS_ETCHEDHORZ
LTEXT "Select the Type of Macro to be Copied to Clipboard", -1, 125, 43, 165, 8
ICON 2535, -1, 130, 55, 18, 20
AUTORADIOBUTTON "&RGB Macro", 101, 167, 55, 52, 8, WS_GROUP
AUTORADIOBUTTON "He&xadecimal", 102, 167, 65, 56, 8
AUTORADIOBUTTON "HTML &Macro", 103, 167, 75, 56, 8
LTEXT -1, 118, 86, 195, 1, SS_ETCHEDHORZ
LTEXT "Specify the Color Swatch Size for Clipboard Copies", -1, 125, 90, 165, 8
ICON 2000, -1, 130, 103, 18, 20
LTEXT "&Width", -1, 167, 103, 22, 8
EDITTEXT 104, 193, 100, 30, 12, ES_AUTOHSCROLL | ES_NUMBER
CONTROL "Spin1", 106, "msctls_updown32", 0x000000B7, 222, 100, 10, 12
LTEXT "&Height", -1, 167, 115, 22, 8
EDITTEXT 105, 193, 114, 30, 12, ES_AUTOHSCROLL | ES_NUMBER
CONTROL "Spin2", 107, "msctls_updown32", 0x000000B7, 222, 114, 10, 12
LTEXT -1, 118, 129, 195, 1, NOT WS_GROUP | SS_ETCHEDHORZ
ICON 2815, -1, 130, 131, 18, 20
AUTOCHECKBOX "&Camera Shutter Sound when Copying", 109, 167, 137, 133, 8
ICON 2525, -1, 131, 149, 18, 20
AUTOCHECKBOX "&Sounds with Warning Dialog Boxes", 110, 167, 154, 126, 8
}

```

Restorator :

```

1000 DIALOG 0, 0, 320, 172
STYLE DS_SETFONT | DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "General"
FONT 8, "MS Sans Serif"
{
CONTROL 1999, -1, "STATIC", SS_BITMAP | SS_REALSIZEIMAGE | SS_SUNKEN, 7, 7, 18, 20
ICON 2655, -1, 132, 7, 18, 20
AUTOCHECKBOX "&Launch Multiple Instances", 108, 167, 5, 99, 8
AUTOCHECKBOX "&Place Icon in Taskbar Tray", 111, 167, 17, 99, 8
AUTOCHECKBOX "Confirm Program &Exit", 112, 167, 29, 79, 8
CONTROL "", -1, "STATIC", SS_ETCHEDHORZ, 118, 40, 195, 1
LTEXT "Select the Type of Macro to be Copied to Clipboard", -1, 125, 43, 165, 8
ICON 2535, -1, 130, 55, 18, 20
AUTORADIOBUTTON "&RGB Macro", 101, 167, 55, 52, 8, WS_GROUP
AUTORADIOBUTTON "He&xadecimal", 102, 167, 65, 56, 8
AUTORADIOBUTTON "HTML &Macro", 103, 167, 75, 56, 8
}

```

```

CONTROL "", -1, "STATIC", SS_ETCHEDHORZ | WS_GROUP, 118, 86, 195, 1
LTEXT "Specify the Color Swatch Size for Clipboard Copies", -1, 125, 90, 165, 8
ICON 2000, -1, 130, 103, 18, 20
LTEXT "&Width", -1, 167, 103, 22, 8
EDITTEXT 104, 193, 100, 30, 12, ES_AUTOHSCROLL | ES_NUMBER
CONTROL "Spin1", 106, "msctls_updown32", 0x000000B7, 222, 100, 10, 12
LTEXT "&Height", -1, 167, 115, 22, 8
EDITTEXT 105, 193, 114, 30, 12, ES_AUTOHSCROLL | ES_NUMBER
CONTROL "Spin2", 107, "msctls_updown32", 0x000000B7, 222, 114, 10, 12
CONTROL "", -1, "STATIC", SS_ETCHEDHORZ, 118, 129, 195, 1
ICON 2815, -1, 130, 131, 18, 20
AUTOCHECKBOX "&Camera Shutter Sound when Copying", 109, 167, 137, 133, 8
ICON 2525, -1, 131, 149, 18, 20
AUTOCHECKBOX "&Sounds with Warning Dialog Boxes", 110, 167, 154, 126, 8
}

```

Resource Builder :

```

1000 DIALOG 0, 0, 320, 172
STYLE DS_SETFONT | DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "General"
FONT 8, "MS Sans Serif"
LANGUAGE LANG_ENGLISH, 1
BEGIN
CONTROL 1999,65535,"STATIC",SS_SUNKEN |SS_REALSIZEIMAGE |SS_BITMAP |WS_CHILD |WS_VISIBLE ,7,7,18,20
CONTROL 2655,65535,"STATIC",SS_ICON |WS_CHILD |WS_VISIBLE ,132,7,18,20
CONTROL "&Launch Multiple Instances",108,"BUTTON",BS_AUTOCHECKBOX |WS_CHILD |WS_TABSTOP |WS_VISIBLE ,167,5,99,8
CONTROL "&Place Icon in Taskbar Tray",111,"BUTTON",BS_AUTOCHECKBOX |WS_CHILD |WS_TABSTOP |WS_VISIBLE ,167,17,99,8
CONTROL "Confirm Program &Exit",112,"BUTTON",BS_AUTOCHECKBOX |WS_CHILD |WS_TABSTOP |WS_VISIBLE ,167,29,79,8
CONTROL "",65535,"STATIC",SS_ETCHEDHORZ |WS_CHILD |WS_VISIBLE ,118,40,195,1
CONTROL "Select the Type of Macro to be Copied to Clipboard",65535,"STATIC",SS_LEFT |WS_CHILD |WS_GROUP |WS_VISIBLE ,125,43,165,8
CONTROL 2535,65535,"STATIC",SS_ICON |WS_CHILD |WS_VISIBLE ,130,55,18,20
CONTROL "&RGB Macro",101,"BUTTON",BS_AUTORADIOBUTTON |WS_CHILD |WS_GROUP |WS_TABSTOP |WS_VISIBLE ,167,55,52,8
CONTROL "He&xadecimal",102,"BUTTON",BS_AUTORADIOBUTTON |WS_CHILD |WS_TABSTOP |WS_VISIBLE ,167,65,56,8
CONTROL "HTML &Macro",103,"BUTTON",BS_AUTORADIOBUTTON |WS_CHILD |WS_TABSTOP |WS_VISIBLE ,167,75,56,8
CONTROL "",65535,"STATIC",SS_ETCHEDHORZ |WS_CHILD |WS_GROUP |WS_VISIBLE ,118,86,195,1
CONTROL "Specify the Color Swatch Size for Clipboard Copies",65535,"STATIC",SS_LEFT |WS_CHILD |WS_GROUP |WS_VISIBLE ,125,90,165,8
CONTROL 2000,65535,"STATIC",SS_ICON |WS_CHILD |WS_VISIBLE ,130,103,18,20
CONTROL "&Width",65535,"STATIC",SS_LEFT |WS_CHILD |WS_GROUP |WS_VISIBLE ,167,103,22,8
CONTROL "",104,"EDIT",ES_NUMBER |ES_AUTOHSCROLL |ES_LEFT |WS_CHILD |WS_BORDER |WS_TABSTOP |WS_VISIBLE ,193,100,30,12
CONTROL "Spin1",106,"msctls_updown32",WS_CHILD |WS_VISIBLE |0xB7,222,100,10,12
CONTROL "&Height",65535,"STATIC",SS_LEFT |WS_CHILD |WS_GROUP |WS_VISIBLE ,167,115,22,8
CONTROL "",105,"EDIT",ES_NUMBER |ES_AUTOHSCROLL |ES_LEFT |WS_CHILD |WS_BORDER |WS_TABSTOP |WS_VISIBLE ,193,114,30,12
CONTROL "Spin2",107,"msctls_updown32",WS_CHILD |WS_VISIBLE |0xB7,222,114,10,12
CONTROL "",65535,"STATIC",SS_ETCHEDHORZ |WS_CHILD |WS_VISIBLE ,118,129,195,1
CONTROL 2815,65535,"STATIC",SS_ICON |WS_CHILD |WS_VISIBLE ,130,131,18,20
CONTROL "&Camera Shutter Sound when Copying",109,"BUTTON",BS_AUTOCHECKBOX |WS_CHILD |WS_TABSTOP |WS_VISIBLE ,167,137,133,8
CONTROL 2525,65535,"STATIC",SS_ICON |WS_CHILD |WS_VISIBLE ,131,149,18,20
CONTROL "&Sounds with Warning Dialog Boxes",110,"BUTTON",BS_AUTOCHECKBOX |WS_CHILD |WS_TABSTOP |WS_VISIBLE ,167,154,126,8
END

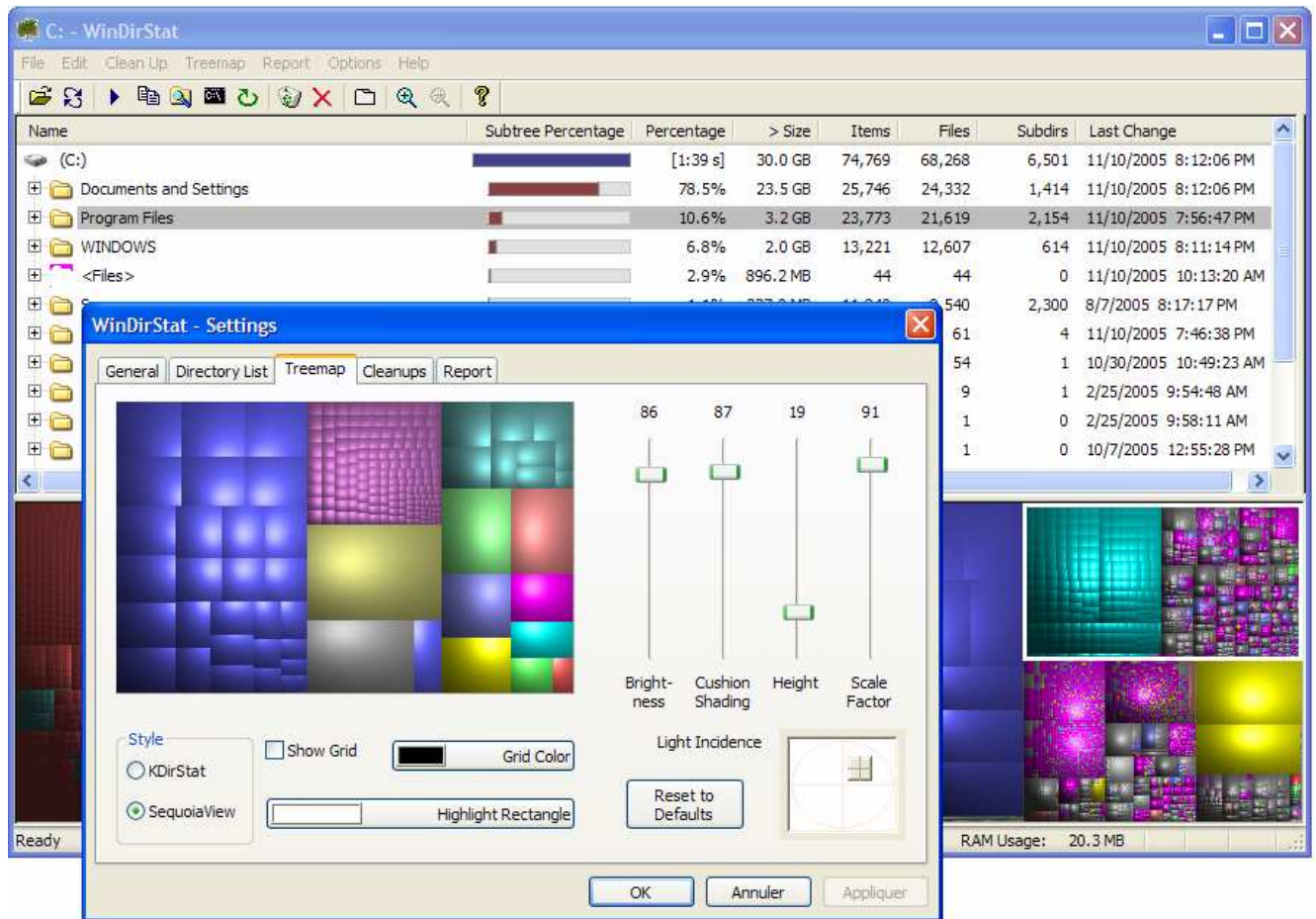
```

Resource Hacker :

```

1000 DIALOG 0, 0, 320, 172
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION
CAPTION "General"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Sans Serif"
{
CONTROL 1999, -1, STATIC, SS_BITMAP | SS_REALSIZEIMAGE | SS_SUNKEN | WS_CHILD | WS_VISIBLE, 7, 7, 18, 20
CONTROL 2655, -1, STATIC, SS_ICON | WS_CHILD | WS_VISIBLE, 132, 7, 18, 20
CONTROL "&Launch Multiple Instances", 108, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 167, 5, 99, 8
CONTROL "&Place Icon in Taskbar Tray", 111, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 167, 17, 99, 8
CONTROL "Confirm Program &Exit", 112, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 167, 29, 79, 8
CONTROL "", -1, STATIC, SS_ETCHEDHORZ | WS_CHILD | WS_VISIBLE, 118, 40, 195, 1
CONTROL "Select the Type of Macro to be Copied to Clipboard", -1, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 125, 43, 165, 8
CONTROL 2535, -1, STATIC, SS_ICON | WS_CHILD | WS_VISIBLE, 130, 55, 18, 20
CONTROL "&RGB Macro", 101, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 167, 55, 52, 8
CONTROL "He&xadecimal", 102, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 167, 65, 56, 8
CONTROL "HTML &Macro", 103, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 167, 75, 56, 8
CONTROL "", -1, STATIC, SS_ETCHEDHORZ | WS_CHILD | WS_VISIBLE | WS_GROUP, 118, 86, 195, 1
CONTROL "Specify the Color Swatch Size for Clipboard Copies", -1, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 125, 90, 165, 8
CONTROL 2000, -1, STATIC, SS_ICON | WS_CHILD | WS_VISIBLE, 130, 103, 18, 20
CONTROL "&Width", -1, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 167, 103, 22, 8
CONTROL "", 104, EDIT, ES_LEFT | ES_AUTOHSCROLL | ES_NUMBER | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 193, 100, 30, 12
CONTROL "Spin1", 106, "msctls_updown32", UDS_WRAP | UDS_SETBUDDYINT | UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS | UDS_NOTHOUSANDS | WS_CHILD | WS_VISIBLE, 222, 100, 10, 12
CONTROL "&Height", -1, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 167, 115, 22, 8
CONTROL "", 105, EDIT, ES_LEFT | ES_AUTOHSCROLL | ES_NUMBER | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 193, 114, 30, 12
CONTROL "Spin2", 107, "msctls_updown32", UDS_WRAP | UDS_SETBUDDYINT | UDS_ALIGNRIGHT | UDS_AUTOBUDDY | UDS_ARROWKEYS | UDS_NOTHOUSANDS | WS_CHILD | WS_VISIBLE, 222, 114, 10, 12
CONTROL "", -1, STATIC, SS_ETCHEDHORZ | WS_CHILD | WS_VISIBLE, 118, 129, 195, 1
CONTROL 2815, -1, STATIC, SS_ICON | WS_CHILD | WS_VISIBLE, 130, 131, 18, 20
CONTROL "&Camera Shutter Sound when Copying", 109, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 167, 137, 133, 8
CONTROL 2525, -1, STATIC, SS_ICON | WS_CHILD | WS_VISIBLE, 131, 149, 18, 20
CONTROL "&Sounds with Warning Dialog Boxes", 110, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 167, 154, 126, 8
}

```

Fourth example

Resource Tuner :

```

143 DIALOGEX 0, 0, 380, 202, 0
STYLE DS_SETFONT | DS_FIXEDSYS | WS_CAPTION | WS_CHILD | WS_DISABLED | WS_SYSMENU
CAPTION "Treemap"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Shell Dlg"
{
  CTEXT "&Bright-\r\nness", -1, 237, 122, 32, 18
  CONTROL "", 1212, "msctls_trackbar32", WS_TABSTOP | 0x0000001A, 245, 17, 16, 104
  CTEXT "&Cushion\r\nShading", -1, 271, 122, 32, 18
  CONTROL "", 1211, "msctls_trackbar32", WS_TABSTOP | 0x0000001A, 279, 17, 16, 104
  CTEXT "&Height", -1, 305, 122, 32, 18
  CONTROL "", 1210, "msctls_trackbar32", WS_TABSTOP | 0x0000001A, 313, 17, 16, 104
  CTEXT "&Scale\r\nFactor", -1, 339, 122, 32, 18
  CONTROL "", 1209, "msctls_trackbar32", WS_TABSTOP | 0x0000001A, 347, 17, 16, 104
  RTEXT "&Light Incidence", -1, 246, 147, 59, 8
  LTEXT "Static", 1220, 313, 146, 58, 48, WS_TABSTOP
  PUSHBUTTON "", 1034, 242, 167, 55, 22, BS_MULTILINE
  GROUPBOX "St&yle", -1, 7, 146, 63, 49
  AUTORADIOBUTTON "&KDirStat", 1213, 12, 159, 42, 10, WS_GROUP
  AUTORADIOBUTTON "Se&quoiaView", 1214, 12, 176, 53, 10, NOT WS_TABSTOP
  AUTOCHECKBOX "Show &Grid", 1022, 76, 150, 54, 10
  PUSHBUTTON "Grid &Color", 1030, 134, 150, 85, 14, 0, WS_EX_RIGHT
  PUSHBUTTON "H&ighlight Rectangle", 1202, 76, 175, 143, 14, 0, WS_EX_RIGHT
  CTEXT "Static", 1219, 339, 7, 32, 8
  CTEXT "Static", 1218, 305, 7, 32, 8
  CTEXT "Static", 1217, 271, 7, 32, 8
  CTEXT "Static", 1216, 237, 7, 32, 8
  LTEXT "Static", 1215, 7, 7, 211, 124
}

```

Restorator :

```

143 DIALOGEX 0, 0, 380, 202, 0
STYLE DS_SETFONT | DS_FIXEDSYS | WS_CHILD | WS_DISABLED | WS_CAPTION | WS_SYSMENU
CAPTION "Treemap"
FONT 8, "MS Shell Dlg", 400, TRUE

```

```

{
CTEXT "&Bright-\r\nness", -1, 237, 122, 32, 18
CONTROL "", 1212, "msctls_trackbar32", WS_TABSTOP | 0x0000001A, 245, 17, 16, 104
CTEXT "&Cushion\r\nShading", -1, 271, 122, 32, 18
CONTROL "", 1211, "msctls_trackbar32", WS_TABSTOP | 0x0000001A, 279, 17, 16, 104
CTEXT "&Height", -1, 305, 122, 32, 18
CONTROL "", 1210, "msctls_trackbar32", WS_TABSTOP | 0x0000001A, 313, 17, 16, 104
CTEXT "&Scale\r\nFactor", -1, 339, 122, 32, 18
CONTROL "", 1209, "msctls_trackbar32", WS_TABSTOP | 0x0000001A, 347, 17, 16, 104
RTEXT "&Light Incidence", -1, 246, 147, 59, 8
LTEXT "Static", 1220, 313, 146, 58, 48, WS_TABSTOP
PUSHBUTTON "", 1034, 242, 167, 55, 22, BS_MULTILINE
GROUPBOX "St&yle", -1, 7, 146, 63, 49
AUTORADIOBUTTON "&KDirStat", 1213, 12, 159, 42, 10, WS_GROUP
AUTORADIOBUTTON "Se&quoiaView", 1214, 12, 176, 53, 10, NOT WS_TABSTOP
AUTOCHECKBOX "Show &Grid", 1022, 76, 150, 54, 10
PUSHBUTTON "Grid &Color", 1030, 134, 150, 85, 14, 0, WS_EX_RIGHT
PUSHBUTTON "H&ighlight Rectangle", 1202, 76, 175, 143, 14, 0, WS_EX_RIGHT
CTEXT "Static", 1219, 339, 7, 32, 8
CTEXT "Static", 1218, 305, 7, 32, 8
CTEXT "Static", 1217, 271, 7, 32, 8
CTEXT "Static", 1216, 237, 7, 32, 8
LTEXT "Static", 1215, 7, 7, 211, 124
}

```

Resource Builder :

```

143 DIALOGEX 0, 0, 380, 202, 0
EXSTYLE 0
STYLE DS_FIXEDSYS | DS_SETFONT | WS_CHILD | WS_CAPTION | WS_SYSMENU | WS_DISABLED
CAPTION "Treemap"
FONT 8, "MS Shell Dlg", 400, 0, 1
LANGUAGE LANG_ENGLISH, 1
BEGIN
CONTROL "&Bright-\x0D\x0Aness", 65535, "STATIC", SS_CENTER | WS_CHILD | WS_GROUP | WS_VISIBLE, 237,122,32,18, 0x0,0
CONTROL "", 1212, "msctls_trackbar32", WS_CHILD | WS_TABSTOP | WS_VISIBLE | 0x1A, 245, 17, 16, 104, 0x0,0
CONTROL "&Cushion\x0D\x0AShading", 65535, "STATIC", SS_CENTER | WS_CHILD | WS_GROUP | WS_VISIBLE, 271,122,32,18, 0x0,0
CONTROL "", 1211, "msctls_trackbar32", WS_CHILD | WS_TABSTOP | WS_VISIBLE | 0x1A, 279,17,16,104, 0x0,0
CONTROL "&Height", -1, 305, 122, 32, 18, 0x0,0
CONTROL "", 1210, "msctls_trackbar32", WS_CHILD | WS_TABSTOP | WS_VISIBLE | 0x1A, 313,17,16,104, 0x0,0
CONTROL "&Scale\x0D\x0AFactor", 65535, "STATIC", SS_CENTER | WS_CHILD | WS_GROUP | WS_VISIBLE, 339,122,32,18, 0x0,0
CONTROL "", 1209, "msctls_trackbar32", WS_CHILD | WS_TABSTOP | WS_VISIBLE | 0x1A, 347,17,16,104, 0x0,0
CONTROL "&Light Incidence", 65535, "STATIC", SS_RIGHT | WS_CHILD | WS_GROUP | WS_VISIBLE, 246,147,59,8, 0x0,0
CONTROL "Static", 1220, "STATIC", SS_LEFT | WS_CHILD | WS_GROUP | WS_VISIBLE, 313,146,58,48, 0x0,0
CONTROL "", 1034, "BUTTON", BS_PUSHBUTTON | BS_MULTILINE | WS_CHILD | WS_TABSTOP | WS_VISIBLE, 242,167,55,22, 0x0,0
CONTROL "St&yle", 65535, "BUTTON", BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 7,146,63,49, 0x0,0
CONTROL "&KDirStat", 1213, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_GROUP | WS_VISIBLE, 12,159,42,10, 0x0,0
CONTROL "Se&quoiaView", 1214, "BUTTON", BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 12,176,53,10, 0x0,0
CONTROL "Show &Grid", 1022, "BUTTON", BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE, 76,150,54,10, 0x0,0
CONTROL "Grid &Color", 1030, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP | WS_VISIBLE, 134, 150, 85, 14, WS_EX_RIGHT, 0
CONTROL "H&ighlight Rectangle", 1202, "BUTTON", BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP | WS_VISIBLE, 76,175,143,14, WS_EX_RIGHT, 0
CONTROL "Static", 1219, "STATIC", SS_CENTER | WS_CHILD | WS_GROUP | WS_VISIBLE, 339, 7, 32, 8, 0x0, 0
CONTROL "Static", 1218, "STATIC", SS_CENTER | WS_CHILD | WS_GROUP | WS_VISIBLE, 305, 7, 32, 8, 0x0, 0
CONTROL "Static", 1217, "STATIC", SS_CENTER | WS_CHILD | WS_GROUP | WS_VISIBLE, 271, 7, 32, 8, 0x0, 0
CONTROL "Static", 1216, "STATIC", SS_CENTER | WS_CHILD | WS_GROUP | WS_VISIBLE, 237, 7, 32, 8, 0x0, 0
CONTROL "Static", 1215, "STATIC", SS_LEFT | WS_CHILD | WS_GROUP | WS_VISIBLE, 7, 7, 211, 124, 0x0, 0
END

```

Resource Hacker :

```

143 DIALOGEX 0, 0, 380, 202
STYLE DS_FIXEDSYS | WS_CHILD | WS_DISABLED | WS_CAPTION | WS_SYSMENU
CAPTION "Treemap"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Shell Dlg", FW_NORMAL, FALSE, 1
{
CONTROL "&Bright-\nness", -1, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 237, 122, 32, 18
CONTROL "", 1212, "msctls_trackbar32", TBS_VERT | TBS_BOTH | TBS_NOTICKS | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 245, 17, 16, 104
CONTROL "&Cushion\nShading", -1, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 271, 122, 32, 18
CONTROL "", 1211, "msctls_trackbar32", TBS_VERT | TBS_BOTH | TBS_NOTICKS | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 279, 17, 16, 104
CONTROL "&Height", -1, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 305, 122, 32, 18
CONTROL "", 1210, "msctls_trackbar32", TBS_VERT | TBS_BOTH | TBS_NOTICKS | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 313, 17, 16, 104
CONTROL "&Scale\nFactor", -1, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 339, 122, 32, 18
CONTROL "", 1209, "msctls_trackbar32", TBS_VERT | TBS_BOTH | TBS_NOTICKS | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 347, 17, 16, 104
CONTROL "&Light Incidence", -1, STATIC, SS_RIGHT | WS_CHILD | WS_VISIBLE | WS_GROUP, 246, 147, 59, 8
CONTROL "Static", 1220, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 313, 146, 58, 48
CONTROL "", 1034, BUTTON, BS_PUSHBUTTON | BS_MULTILINE | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 242, 167, 55, 22
CONTROL "St&yle", -1, BUTTON, BS_GROUPBOX | WS_CHILD | WS_VISIBLE, 7, 146, 63, 49
CONTROL "&KDirStat", 1213, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 12, 159, 42, 10
CONTROL "Se&quoiaView", 1214, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 12, 176, 53, 10
CONTROL "Show &Grid", 1022, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 76, 150, 54, 10
CONTROL "Grid &Color", 1030, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 134, 150, 85, 14, 0x00001000
CONTROL "H&ighlight Rectangle", 1202, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 76, 175, 143, 14, 0x00001000
CONTROL "Static", 1219, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 339, 7, 32, 8
CONTROL "Static", 1218, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 305, 7, 32, 8
CONTROL "Static", 1217, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 271, 7, 32, 8
CONTROL "Static", 1216, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE | WS_GROUP, 237, 7, 32, 8
CONTROL "Static", 1215, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 7, 7, 211, 124
}

```


Fifth example

Resource Tuner :

```

1552 DIALOGEX 0, 0, 370, 237, 0
STYLE DS_SETFONT | DS_MODALFRAME | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUPWINDOW | WS_VISIBLE | WS_CLIPCHILDREN | WS_DLDFRAME
CAPTION "Open"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Shell Dlg"
{
    LTEXT "Look &in :", 1091, 4, 7, 57, 8, SS_NOTIFY, WS_EX_RIGHT
    COMBOBOX "", 1137, 66, 4, 174, 300, CBS_DROPDOWN | CBS_OWNERDRAWFIXED | CBS_HASSTRINGS | WS_VSCROLL
    LTEXT "", 1088, 248, 4, 80, 14, NOT WS_VISIBLE | NOT WS_GROUP
    CONTROL "", 1184, "ToolbarWindow32", WS_TABSTOP | 0x00002B4C, 4, 22, 58, 208, WS_EX_CLIENTEDGE
    LISTBOX "", 1120, 66, 22, 300, 156, NOT WS_VISIBLE | LBS_SORT | LBS_NOINTEGRALHEIGHT | LBS_MULTICOLUMN | WS_HSCROLL
    LTEXT "File &name :", 1090, 67, 187, 71, 8, SS_NOTIFY
    EDITTEXT "", 1152, 144, 184, 164, 12, ES_AUTOHSCROLL
    CONTROL "", 1148, "ComboBoxEx32", WS_VSCROLL | WS_TABSTOP | 0x00000042, 144, 184, 164, 150
    LTEXT "Files of &type :", 1089, 67, 203, 71, 8, SS_NOTIFY
    COMBOBOX "", 1136, 144, 201, 164, 100, CBS_DROPDOWN | WS_VSCROLL
    AUTOCHECKBOX "Open as &read-only ", 1040, 144, 217, 160, 8
    DEFPUSHBUTTON "&Open", 1, 316, 184, 50, 14, WS_GROUP
    PUSHBUTTON "Cancel", 2, 316, 200, 50, 14, WS_GROUP
}

```

For tool bar control, “WS_TABSTOP|0x00002B4C” is the same as “0x50012B4C”, seeing that WS_CHILD (0x40000000) and WS_VISIBLE (0x10000000) are implicit with the shorthand notation.

Restorator :

```

1552 DIALOGEX 0, 0, 370, 237, 0
STYLE DS_SETFONT|DS_MODALFRAME|DS_3DLOOK|DS_CONTEXTHELP|WS_POPUPWINDOW| WS_VISIBLE|WS_CLIPCHILDREN|WS_CAPTION
CAPTION "Open"
FONT 8, "MS Shell Dlg", 0, TRUE
{
    LTEXT "Look &in :", 1091, 4, 7, 57, 8, SS_NOTIFY, WS_EX_RIGHT
    COMBOBOX 1137, 66, 4, 174, 300, CBS_DROPDOWNLIST | CBS_OWNERDRAWFIXED | CBS_HASSTRINGS | WS_VSCROLL
    LTEXT "", 1088, 248, 4, 80, 14, NOT WS_VISIBLE | NOT WS_GROUP
    CONTROL "", 1184, "ToolbarWindow32", WS_TABSTOP | 0x00002B4C, 4, 22, 58, 208, WS_EX_CLIENTEDGE
    LISTBOX 1120, 66, 22, 300, 156, LBS_SORT | LBS_NOINTEGRALHEIGHT | LBS_MULTICOLUMN | NOT WS_VISIBLE | WS_HSCROLL
    LTEXT "File &name :", 1090, 67, 187, 71, 8, SS_NOTIFY
    EDITTEXT 1152, 144, 184, 164, 12, ES_AUTOHSCROLL
    CONTROL "", 1148, "ComboBoxEx32", WS_VSCROLL | WS_TABSTOP | 0x00000042, 144, 184, 164, 150
    LTEXT "Files of &type :", 1089, 67, 203, 71, 8, SS_NOTIFY
    COMBOBOX 1136, 144, 201, 164, 100, CBS_DROPDOWNLIST | WS_VSCROLL
    AUTOCHECKBOX "Open as &read-only", 1040, 144, 217, 160, 8
    DEFPUSHBUTTON "&Open", 1, 316, 184, 50, 14, WS_GROUP
    PUSHBUTTON "Cancel", 2, 316, 200, 50, 14, WS_GROUP
}

```

Resource Builder :

```

1552 DIALOGEX 28, 85, 370, 237, 0
EXSTYLE 0
STYLE DS_3DLOOK | DS_SETFONT | DS_MODALFRAME | DS_CONTEXTHELP | WS_POPUP | WS_VISIBLE | WS_CLIPCHILDREN | WS_SYSMENU | WS_CAPTION
CAPTION "Open"
FONT 8, "MS Shell Dlg", 400, 0, 0
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
BEGIN
CONTROL "Look &in :",1091,"STATIC",SS_NOTIFY |SS_LEFT |WS_CHILD |WS_GROUP |WS_VISIBLE ,4,7,57,8,WS_EX_RIGHT ,0
CONTROL "",1137,"COMBOBOX",CBS_OWNERDRAWFIXED|CBS_HASSTRINGS|CBS_DROPDOWNLIST|WS_CHILD|WS_VSCROLL|WS_TABSTOP|WS_VISIBLE,
CONTROL "",1088,"STATIC",SS_LEFT |WS_CHILD ,248,4,80,14,0x0,0 66,4,174,300,0x0,0
CONTROL "",1184,"ToolbarWindow32",WS_CHILD |WS_TABSTOP |WS_VISIBLE |0x2B4C,4,22,58,208,0x0,0
CONTROL "",1120,"LISTBOX",LBS_MULTICOLUMN|LBS_NOINTEGRALHEIGHT|LBS_SORT|LBS_NOTIFY|WS_CHILD|WS_BORDER|WS_HSCROLL,66,22,300,156,0x0,0
CONTROL "File &name :",1090,"STATIC",SS_NOTIFY |SS_LEFT |WS_CHILD |WS_GROUP |WS_VISIBLE ,67,187,71,8,0x0,0
CONTROL "",1152,"EDIT",ES_AUTOHSCROLL |ES_LEFT |WS_CHILD |WS_BORDER |WS_TABSTOP |WS_VISIBLE ,144,184,164,12,0x0,0
CONTROL "",1148,"ComboBoxEx32",WS_CHILD |WS_VSCROLL |WS_TABSTOP |WS_VISIBLE |0x42,144,184,164,150,0x0,0
CONTROL "Files of &type :",1089,"STATIC",SS_NOTIFY |SS_LEFT |WS_CHILD |WS_GROUP |WS_VISIBLE ,67,203,71,8,0x0,0
CONTROL "",1136,"COMBOBOX",CBS_DROPDOWNLIST |WS_CHILD |WS_VSCROLL |WS_TABSTOP |WS_VISIBLE ,144,201,164,100,0x0,0
CONTROL "Open as &read-only ",1040,"BUTTON",BS_AUTOCHECKBOX |BS_LEFT |WS_CHILD |WS_TABSTOP |WS_VISIBLE ,144,217,160,8,0x0,0
CONTROL "&Open",1,"BUTTON",BS_DEFPUSHBUTTON |BS_VCENTER |BS_CENTER |WS_CHILD |WS_GROUP |WS_TABSTOP |WS_VISIBLE ,316,184,50,14,0x0,0
CONTROL "Cancel",2,"BUTTON",BS_PUSHBUTTON |BS_VCENTER |BS_CENTER |WS_CHILD |WS_GROUP |WS_TABSTOP |WS_VISIBLE ,316,200,50,14,0x0,0
END

```

Resource Hacker :

```

1552 DIALOGEX 0, 0, 370, 237

```

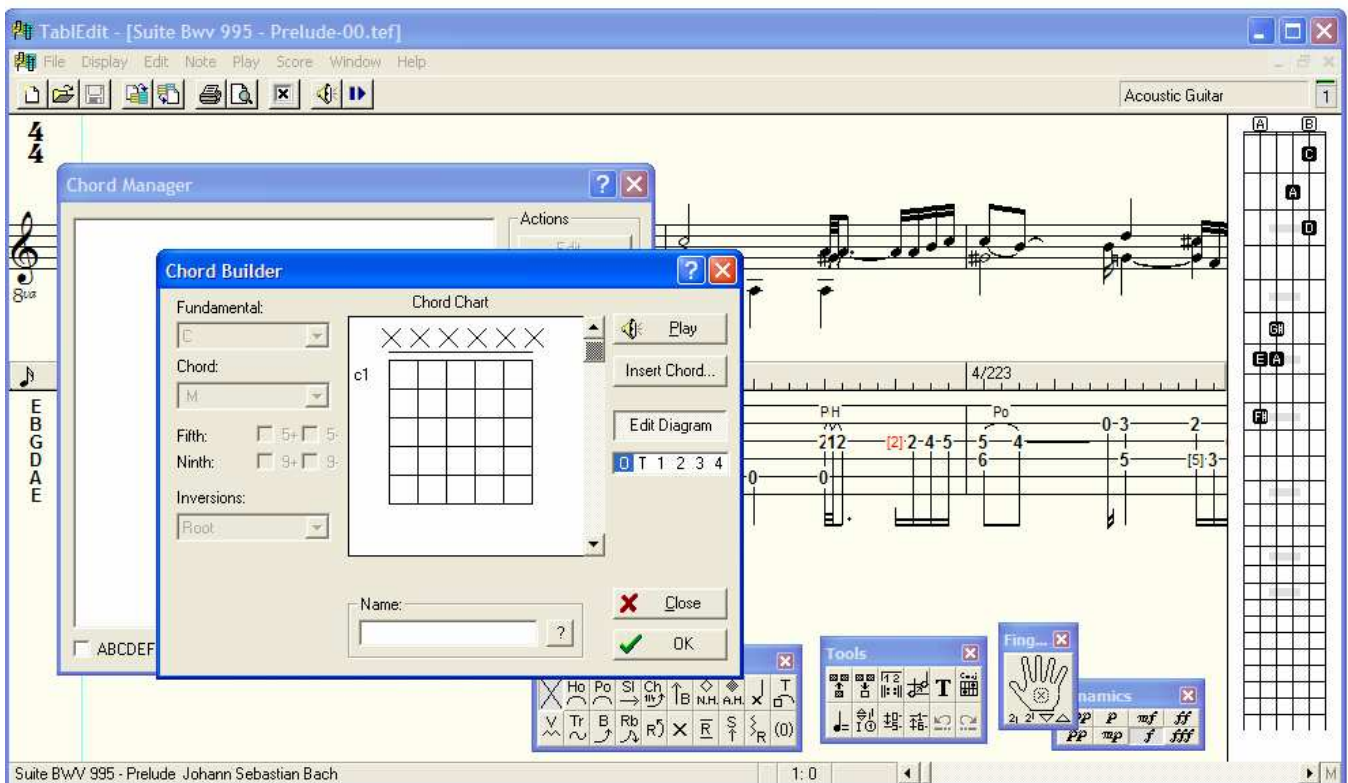
```

STYLE DS_MODALFRAME | DS_CONTEXTHELP | WS_POPUP | WS_VISIBLE | WS_CLIPCHILDREN | WS_CAPTION | WS_SYSMENU
CAPTION "Open"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Shell Dlg"
{
CONTROL "Look &in :", 1091, STATIC, SS_LEFT | SS_NOTIFY | WS_CHILD | WS_VISIBLE | WS_GROUP, 4, 7, 57, 8, 0x00001000
CONTROL "", 1137, COMBOBOX, CBS_DROPDOWNLIST|CBS_OWNERDRAWFIXED|CBS_HASSTRINGS|WS_CHILD|WS_VISIBLE|WS_VSCROLL|WS_TABSTOP,
66, 4, 174, 300
CONTROL "", 1088, STATIC, SS_LEFT | WS_CHILD, 248, 4, 80, 14
CONTROL "", 1184, "ToolBarWindow32", 0x50012B4C, 4, 22, 58, 208, 0x00000200
CONTROL "", 1120, LISTBOX, LBS_NOTIFY | LBS_NOINTEGRALHEIGHT | LBS_MULTICOLUMN | WS_CHILD | WS_BORDER | WS_HSCROLL, 66, 22, 300, 156
CONTROL "File &name :", 1090, STATIC, SS_LEFT | SS_NOTIFY | WS_CHILD | WS_VISIBLE | WS_GROUP, 67, 187, 71, 8
CONTROL "", 1152, EDIT, ES_LEFT | ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 144, 184, 164, 12
CONTROL "", 1148, "ComboBoxEx32", 0x50210042, 144, 184, 164, 150
CONTROL "Files of &type :", 1089, STATIC, SS_LEFT | SS_NOTIFY | WS_CHILD | WS_VISIBLE | WS_GROUP, 67, 203, 71, 8
CONTROL "", 1136, COMBOBOX, CBS_DROPDOWNLIST | WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 144, 201, 164, 100
CONTROL "Open as &read-only", 1040, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 144, 217, 160, 8
CONTROL "&Open", 1, BUTTON, BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 316, 184, 50, 14
CONTROL "Cancel", 2, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 316, 200, 50, 14
}

```

The first static control has also the number 1000 as extended style corresponding to `WS_EX_RIGHT`, which for static controls has the same effect as using `SS_RIGHT`. The hexadecimal number “0x50012B4C” for the tool bar control is the same as “`TBSTYLE_TOOLTIPS | TBSTYLE_WRAPABLE | TBSTYLE_FLAT | TBSTYLE_CUSTOMERASE | CCS_NORESIZE | CCS_NOPARENTALIGN | CCS_NODIVIDER | WS_TABSTOP | WS_CHILD | WS_VISIBLE`”, and the number “0x00000200” is the same as “`WS_EX_CLIENTEDGE`”. The number 50210042 for the combo box control correspond to the styles `CBS_DROPDOWN`, `CBS_AUTOHSCROLL`, `WS_TABSTOP`, `WS_VSCROLL`, `WS_VISIBLE` and `WS_CHILD`.

Sixth example



Resource Tuner :

```

22 DIALOG 76, 29, 284, 174
STYLE DS_SETFONT | DS_MODALFRAME | DS_NOIDLEMSG | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUPWINDOW | WS_DLDFRAME
EXSTYLE WS_EX_DLGMODALFRAME | WS_EX_CONTEXTHELP
CAPTION "Chord Builder"
LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
FONT 8, "MS Sans Serif"

```

```

{
LTEXT "&Fundamental:", 24, 8, 4, 60, 10, NOT WS_GROUP
COMBOBOX 101, 8, 15, 76, 115, CBS_DROPDOWN | WS_VSCROLL
LTEXT "&Chord:", 25, 8, 31, 54, 10
COMBOBOX 102, 8, 42, 76, 100, CBS_DROPDOWN | WS_VSCROLL
LTEXT "&Fifth:", 26, 8, 62, 32, 10, NOT WS_GROUP
AUTOCHECKBOX "5+", 301, 47, 61, 21, 10
AUTOCHECKBOX "5-", 302, 69, 61, 21, 10
LTEXT "&Ninth:", 27, 8, 74, 38, 10, NOT WS_GROUP
AUTOCHECKBOX "9+", 401, 47, 73, 21, 10
AUTOCHECKBOX "9-", 402, 69, 73, 21, 10
LTEXT "&Inversions:", 28, 8, 90, 63, 10, NOT WS_GROUP
COMBOBOX 103, 8, 101, 76, 55, CBS_DROPDOWN | WS_VSCROLL
AUTORADIOBUTTON "&1", 700, 98, 149, 17, 10, WS_GROUP
AUTORADIOBUTTON "&2", 701, 116, 149, 17, 10, NOT WS_TABSTOP
AUTORADIOBUTTON "&3", 702, 135, 149, 17, 10, NOT WS_TABSTOP
AUTORADIOBUTTON "&4", 703, 154, 149, 17, 10, NOT WS_TABSTOP
AUTORADIOBUTTON "&5", 704, 176, 149, 17, 10, NOT WS_TABSTOP
GROUPBOX 106, 92, 138, 116, 29, WS_GROUP
SCROLLBAR 600, 93, 107, 81, 9, WS_TABSTOP
SCROLLBAR 601, 155, 56, 9, 64, SBS_VERT | WS_TABSTOP
GROUPBOX "&Omit:", 799, 8, 118, 76, 49, WS_GROUP
CHECKBOX 800, 14, 128, 30, 10
CHECKBOX 801, 50, 128, 30, 10
CHECKBOX 802, 14, 140, 30, 10
CHECKBOX 803, 50, 140, 30, 10
CHECKBOX 804, 14, 152, 30, 10
CHECKBOX 805, 50, 152, 30, 10
CTEXT "Chord Chart", 30, 108, 2, 68, 10, NOT WS_GROUP
CONTROL "" 999, "STATIC", NOT WS_VISIBLE | NOT WS_GROUP | SS_GRAYFRAME, 92, 12, 116, 108
AUTORADIOBUTTON "&Close", 2, 222, 135, 56, 14, 0x00000002
AUTORADIOBUTTON "&Add", 1, 222, 153, 56, 14, 0x00000002
AUTORADIOBUTTON "&Play", 131, 222, 11, 56, 14, 0x00000002
PUSHBUTTON "&Insert Chord...", 105, 222, 30, 56, 14
EDITTEXT 107, 97, 149, 88, 12
AUTOCHECKBOX "?", 109, 189, 149, 13, 12, BS_PUSHLIKE
CHECKBOX "&Edit Diagram", 108, 222, 55, 56, 14, BS_PUSHLIKE
LISTBOX 110, 221, 73, 58, 11, NOT LBS_NOTIFY | LBS_NOINTEGRALHEIGHT | LBS_MULTICOLUMN | WS_TABSTOP
}

```

Restorator :

```

22 DIALOG 76, 29, 284, 174
STYLE DS_SETFONT | DS_MODALFRAME | DS_NOIDLEMSG | DS_3DLOOK | DS_CONTEXTHELP | WS_POPUPWINDOW | WS_CAPTION
EXSTYLE = WS_EX_DLGMODALFRAME | WS_EX_CONTEXTHELP
CAPTION "Chord Builder"
FONT 8, "MS Sans Serif"
{
LTEXT "&Fundamental:", 24, 8, 4, 60, 10, NOT WS_GROUP
COMBOBOX 101, 8, 15, 76, 115, CBS_DROPDOWNLIST | WS_VSCROLL
LTEXT "&Chord:", 25, 8, 31, 54, 10
COMBOBOX 102, 8, 42, 76, 100, CBS_DROPDOWNLIST | WS_VSCROLL
LTEXT "&Fifth:", 26, 8, 62, 32, 10, NOT WS_GROUP
AUTOCHECKBOX "5+", 301, 47, 61, 21, 10
AUTOCHECKBOX "5-", 302, 69, 61, 21, 10
LTEXT "&Ninth:", 27, 8, 74, 38, 10, NOT WS_GROUP
AUTOCHECKBOX "9+", 401, 47, 73, 21, 10
AUTOCHECKBOX "9-", 402, 69, 73, 21, 10
LTEXT "&Inversions:", 28, 8, 90, 63, 10, NOT WS_GROUP
COMBOBOX 103, 8, 101, 76, 55, CBS_DROPDOWNLIST | WS_VSCROLL
AUTORADIOBUTTON "&1", 700, 98, 149, 17, 10, WS_GROUP
AUTORADIOBUTTON "&2", 701, 116, 149, 17, 10, NOT WS_TABSTOP
AUTORADIOBUTTON "&3", 702, 135, 149, 17, 10, NOT WS_TABSTOP
AUTORADIOBUTTON "&4", 703, 154, 149, 17, 10, NOT WS_TABSTOP
AUTORADIOBUTTON "&5", 704, 176, 149, 17, 10, NOT WS_TABSTOP
GROUPBOX "", 106, 92, 138, 116, 29, WS_GROUP
SCROLLBAR 600, 93, 107, 81, 9, WS_TABSTOP
SCROLLBAR 601, 155, 56, 9, 64, SBS_VERT | WS_TABSTOP
GROUPBOX "&Omit:", 799, 8, 118, 76, 49, WS_GROUP
CHECKBOX "", 800, 14, 128, 30, 10
CHECKBOX "", 801, 50, 128, 30, 10
CHECKBOX "", 802, 14, 140, 30, 10
CHECKBOX "", 803, 50, 140, 30, 10
CHECKBOX "", 804, 14, 152, 30, 10
CHECKBOX "", 805, 50, 152, 30, 10
CTEXT "Chord Chart", 30, 108, 2, 68, 10, NOT WS_GROUP
CONTROL "", 999, "STATIC", SS_SIMPLE | NOT WS_VISIBLE, 92, 12, 116, 108
CONTROL "&Close", 2, "BUTTON", BS_OWNERDRAW | WS_TABSTOP, 222, 135, 56, 14
CONTROL "&Add", 1, "BUTTON", BS_OWNERDRAW | WS_TABSTOP, 222, 153, 56, 14
CONTROL "&Play", 131, "BUTTON", BS_OWNERDRAW | WS_TABSTOP, 222, 11, 56, 14
PUSHBUTTON "&Insert Chord...", 105, 222, 30, 56, 14
EDITTEXT 107, 97, 149, 88, 12
AUTOCHECKBOX "?", 109, 189, 149, 13, 12, BS_PUSHLIKE
CHECKBOX "&Edit Diagram", 108, 222, 55, 56, 14, BS_PUSHLIKE
LISTBOX 110, 221, 73, 58, 11, NOT LBS_NOTIFY | LBS_NOINTEGRALHEIGHT | LBS_MULTICOLUMN | WS_TABSTOP
}

```

Resource Builder :

22 DIALOG 76, 29, 284, 174

```

STYLE DS_3DLOOK | DS_SETFONT | DS_MODALFRAME | DS_NOIDLEMSG | DS_CONTEXTHELP | WS_POPUP | WS_SYSTEMMENU | WS_CAPTION
CAPTION "Chord Builder"
FONT 8, "MS Sans Serif"
LANGUAGE LANG_NEUTRAL, 0
BEGIN
CONTROL "&Fundamental:",24,"STATIC",SS_LEFT | WS_CHILD | WS_VISIBLE ,8,4,60,10
CONTROL "",101,"COMBOBOX",CBS_DROPDOWNLIST | WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_TABSTOP | WS_VISIBLE ,8,15,76,115
CONTROL "&Chord:",25,"STATIC",SS_LEFT | WS_CHILD | WS_GROUP | WS_VISIBLE ,8,31,54,10
CONTROL "",102,"COMBOBOX",CBS_DROPDOWNLIST | WS_CHILD | WS_VSCROLL | WS_TABSTOP | WS_VISIBLE ,8,42,76,100
CONTROL "&Fifth:",26,"STATIC",SS_LEFT | WS_CHILD | WS_VISIBLE ,8,62,32,10
CONTROL "5+",301,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,47,61,21,10
CONTROL "5-",302,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,69,61,21,10
CONTROL "&Ninth:",27,"STATIC",SS_LEFT | WS_CHILD | WS_VISIBLE ,8,74,38,10
CONTROL "9+",401,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,47,73,21,10
CONTROL "9-",402,"BUTTON",BS_AUTOCHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,69,73,21,10
CONTROL "&Inversions:",28,"STATIC",SS_LEFT | WS_CHILD | WS_VISIBLE ,8,90,63,10
CONTROL "",103,"COMBOBOX",CBS_DROPDOWNLIST | WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_TABSTOP | WS_VISIBLE ,8,101,76,55
CONTROL "&1",700,"BUTTON",BS_AUTORADIOBUTTON | WS_CHILD | WS_GROUP | WS_TABSTOP | WS_VISIBLE ,98,149,17,10
CONTROL "&2",701,"BUTTON",BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE ,116,149,17,10
CONTROL "&3",702,"BUTTON",BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE ,135,149,17,10
CONTROL "&4",703,"BUTTON",BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE ,154,149,17,10
CONTROL "&5",704,"BUTTON",BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE ,176,149,17,10
CONTROL "",106,"BUTTON",BS_GROUPBOX | WS_CHILD | WS_GROUP | WS_VISIBLE ,92,138,116,29
CONTROL "",600,"SCROLLBAR",SBS_HORZ | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,93,107,81,9
CONTROL "",601,"SCROLLBAR",SBS_VERT | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,155,56,9,64
CONTROL "&Omit:",799,"BUTTON",BS_GROUPBOX | WS_CHILD | WS_GROUP | WS_VISIBLE ,8,118,76,49
CONTROL "",800,"BUTTON",BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,14,128,30,10
CONTROL "",801,"BUTTON",BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,50,128,30,10
CONTROL "",802,"BUTTON",BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,14,140,30,10
CONTROL "",803,"BUTTON",BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,50,140,30,10
CONTROL "",804,"BUTTON",BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,14,152,30,10
CONTROL "",805,"BUTTON",BS_CHECKBOX | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,50,152,30,10
CONTROL "Chord Chart",30,"STATIC",SS_CENTER | WS_CHILD | WS_VISIBLE ,108,2,68,10
CONTROL "",999,"STATIC",SS_SIMPLE | WS_CHILD ,92,12,116,108
CONTROL "&Close",2,"BUTTON",BS_OWNERDRAW | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,222,135,56,14
CONTROL "&Add",1,"BUTTON",BS_OWNERDRAW | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,222,153,56,14
CONTROL "&Play",131,"BUTTON",BS_OWNERDRAW | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,222,11,56,14
CONTROL "&Insert Chord...",105,"BUTTON",BS_PUSHBUTTON | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,222,30,56,14
CONTROL "",107,"EDIT",ES_LEFT | WS_CHILD | WS_BORDER | WS_TABSTOP | WS_VISIBLE ,97,149,88,12
CONTROL "?",109,"BUTTON",BS_AUTOCHECKBOX | BS_PUSHLIKE | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,189,149,13,12
CONTROL "&Edit Diagram",108,"BUTTON",BS_CHECKBOX | BS_PUSHLIKE | WS_CHILD | WS_TABSTOP | WS_VISIBLE ,222,55,56,14
CONTROL "",110,"LISTBOX",LBS_MULTICOLUMN|LBS_NOINTEGRALHEIGHT|WS_CHILD|WS_BORDER|WS_TABSTOP|WS_VISIBLE,221,73,58,11
END

```

Resource Hacker :

```

22 DIALOG 76, 29, 284, 174
STYLE DS_MODALFRAME | DS_NOIDLEMSG | DS_CONTEXTHELP | WS_POPUP | WS_CAPTION | WS_SYSTEMMENU
EXSTYLE WS_EX_DLGMODALFRAME | WS_EX_CONTEXTHELP
CAPTION "Chord Builder"
LANGUAGE LANG_NEUTRAL, SUBLANG_NEUTRAL
FONT 8, "MS Sans Serif"
{
CONTROL "&Fundamental:", 24, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 4, 60, 10
CONTROL "", 101, COMBOBOX, CBS_DROPDOWNLIST | WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 8, 15, 76, 115
CONTROL "&Chord:", 25, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE | WS_GROUP, 8, 31, 54, 10
CONTROL "", 102, COMBOBOX, CBS_DROPDOWNLIST | WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 8, 42, 76, 100
CONTROL "&Fifth:", 26, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 62, 32, 10
CONTROL "5+", 301, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 47, 61, 21, 10
CONTROL "5-", 302, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 69, 61, 21, 10
CONTROL "&Ninth:", 27, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 74, 38, 10
CONTROL "9+", 401, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 47, 73, 21, 10
CONTROL "9-", 402, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 69, 73, 21, 10
CONTROL "&Inversions:", 28, STATIC, SS_LEFT | WS_CHILD | WS_VISIBLE, 8, 90, 63, 10
CONTROL "", 103, COMBOBOX, CBS_DROPDOWNLIST | WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 8, 101, 76, 55
CONTROL "&1", 700, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 98, 149, 17, 10
CONTROL "&2", 701, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 116, 149, 17, 10
CONTROL "&3", 702, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 135, 149, 17, 10
CONTROL "&4", 703, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 154, 149, 17, 10
CONTROL "&5", 704, BUTTON, BS_AUTORADIOBUTTON | WS_CHILD | WS_VISIBLE, 176, 149, 17, 10
CONTROL "", 106, BUTTON, BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP, 92, 138, 116, 29
CONTROL "", 600, SCROLLBAR, SBS_HORZ | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 93, 107, 81, 9
CONTROL "", 601, SCROLLBAR, SBS_VERT | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 155, 56, 9, 64
CONTROL "", 799, BUTTON, BS_GROUPBOX | WS_CHILD | WS_VISIBLE | WS_GROUP, 8, 118, 76, 49
CONTROL "", 800, BUTTON, BS_CHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 14, 128, 30, 10
CONTROL "", 801, BUTTON, BS_CHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 50, 128, 30, 10
CONTROL "", 802, BUTTON, BS_CHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 14, 140, 30, 10
CONTROL "", 803, BUTTON, BS_CHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 50, 140, 30, 10
CONTROL "", 804, BUTTON, BS_CHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 14, 152, 30, 10
CONTROL "", 805, BUTTON, BS_CHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 50, 152, 30, 10
CONTROL "Chord Chart", 30, STATIC, SS_CENTER | WS_CHILD | WS_VISIBLE, 108, 2, 68, 10
CONTROL "", 999, STATIC, SS_SIMPLE | WS_CHILD, 92, 12, 116, 108
CONTROL "&Close", 2, BUTTON, BS_OWNERDRAW | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 222, 135, 56, 14
CONTROL "&Add", 1, BUTTON, BS_OWNERDRAW | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 222, 153, 56, 14
CONTROL "&Play", 131, BUTTON, BS_OWNERDRAW | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 222, 11, 56, 14
CONTROL "&Insert Chord...", 105, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 222, 30, 56, 14
CONTROL "", 107, EDIT, ES_LEFT | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 97, 149, 88, 12
CONTROL "?", 109, BUTTON, BS_AUTOCHECKBOX | BS_PUSHLIKE | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 189, 149, 13, 12
CONTROL "&Edit Diagram", 108, BUTTON, BS_CHECKBOX | BS_PUSHLIKE | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 222, 55, 56, 14
CONTROL "", 110, LISTBOX, LBS_MULTICOLUMN|LBS_NOINTEGRALHEIGHT|WS_CHILD|WS_VISIBLE|WS_BORDER|WS_TABSTOP, 221, 73, 58, 11
}

```

Appendix C

Documentation of UML diagrams modeling a resource file

WINDOW

In a graphical Windows-based application, dialog boxes and controls are windows (rectangular areas of the screen where the application displays output and receives input from the user).

Text (String)

If the dialog box has a caption bar (also called title bar), it is the text put in it. Text can also be displayed with each button or static child window (for all the other control the value is an empty string). In this case, an ampersand (&) inside the value causes the letter that follows to be underlined in the interface to indicate the keyboard shortcut for that control (the mnemonic), and the ampersand is not displayed.⁹⁵ The input focus can then be moved to any of these controls by pressing ALT+mnemonic.⁹⁶



GeneralStyle ({popup, child, overlapped})

There are three general styles of window: a popup window is a temporary subsidiary window, a child window can divide a window in various regions and an overlapped window is a program's main application window. Only the two first will be relevant here (that is the inherited GeneralStyle will never take the value *overlapped*).

Border (Boolean)

Specifies the frame around the the window. If true, it has a normal thin-line window border.

Caption (Boolean)

Specifies the frame around the dialog box. If true, it has a caption bar and a normal window border (implies Border = true). A caption bar allows the user to move the dialog box to another area of the display (often not provided with modal dialog box for example, because the user can't do anything in the underlying window anyway). Caption and DialogFrame cannot both be true (see DialogModalFrame beyond).

ClipChildren (Boolean)

If true, excludes the area occupied by child windows when drawing occurs within the parent window (style used when creating the parent window).

⁹⁵ Unless otherwise specified by the NoPrefix attribute from the STATIC class.

⁹⁶ For a static control, the focus move to the first control having Tabstop=true after the static control containing the specified mnemonic.

ClipSimblings (*Boolean*)

If true, clips child windows relative to each other. That is, when a particular child window receives a paint message⁹⁷, all other overlapping child windows are clipped out of the region of the child window to be updated. If false and child windows overlap, it is possible, when drawing within the client area of a child window, to draw within the client area of a neighbouring child window.

Disabled (*Boolean*)

Specifies the initial state of the window (dialog box or control). If true, it is initially disabled. To receive mouse and keyboard input, a control must be both visible and enabled (by default).

DialogFrame (*Boolean*)

Specifies the frame around the dialog box. If true, it has a thicker frame (double border) typically used with dialog boxes. A window with this style cannot have a title bar.

Group (*Boolean*)

If true, it's the first control of a group of controls in which the user can move from one control to the next with the arrow keys. In a resource file, all controls without this style (Group = false in the class diagram) after such control belong to the same group. The next control with this style starts the next group (that is, one group ends where the next begins).

TabStop (*Boolean*)

If true, the user can move by using the TAB key through the controls. The TAB key moves the user to the next control having this style. The first control in each group usually has this attribute set to true so that the user can move from group to group. These key interfaces that Windows adds to a dialog box are important with radio buttons. After using the TAB key to move to the currently checked radio button within the group, the arrow keys can be used to change the input focus from that radio button to another within this group. The system automatically assigns the style to the newly checked control when the user moves between controls in the group. This ensures that the input focus will always be on the most recently selected control when the user moves to the group using the TAB key.

HorizontalScrollBar (*Boolean*)

If true, a horizontal scroll bar is added at the bottom of the window. Also valid for child windows: when the multiline edit controls have the automatic scrolling style, it's sometimes useful to add window scroll bars to the edit control.

MaximizeBox (*Boolean*)

If true (although unusual), an icon is displayed in the upper left corner that allow the user to maximize the dialog box (and the maximize option in the system menu is enabled).

Maximized (*Boolean*)

Specifies the initial state of the dialog box. If true, it is initially maximized. Without specific initial state, it appears as specified by its coordinate and dimension.

MinimizeBox (*Boolean*)

If true, an icon is displayed in the upper left corner that allow the user to minimize the dialog box (and the minimize option in the system menu is enabled).

⁹⁷ The WM_PAINT message.

Minimized (Boolean)

Specifies the initial state of the dialog box. If true, it is initially minimized.

SystemMenu (Boolean)

If true, the window has a system menu box in its caption bar (implies `Caption = true`). You can then close or minimize the dialog, move it around the display using the keyboard, etc. A popup window has often just the Close option.

ThickFrame (Boolean)

If true, a thick frame is added so the user can resize the dialog box (and the Size option in the system menu is enabled).

VerticalScrollBar (Boolean)

If true, a vertical scroll bar is added at the right of the window. Also valid for child windows: a scroll bar can be added to a multiline edit control with the automatic scrolling style and to a list box.

Visible (Boolean)

Specifies the initial state of the window. If true, this ensures for example that a control is visible when the dialog box is displayed.

The remaining attributes adds extended styles to the window.

ClientEdge (Boolean)

If true, the client area of the window has a 3D look — that is, a border with a sunken edge.

DialogModalFrame (Boolean)

If true, the window has a double border that may (optionally) be created with a title bar (if `Caption = true`).

StaticEdge (Boolean)

If true, the window has a three-dimensional border style intended to be used for items that do not accept user input.

WindowEdge (Boolean)

If true, the window has a border with a raised edge.

AcceptFiles (Boolean)

If true, the window accepts drag-and-drop files.

AppWindow (Boolean)

If true, forces a top-level window onto the taskbar when the window is visible.

ControlParent (Boolean)

If true, the dialog manager recurses into children of this window when performing navigation operations such as handling the TAB key, an arrow key, or a keyboard mnemonic.

ExContextHelp (Boolean)

If true, a question mark is included in the title bar of the window. When the user clicks it, the cursor changes to a question mark with a pointer. If the user then clicks a child window, the

child receives a help message⁹⁸. The control should pass the message to the dialog box procedure. The help application displays a pop-up window that typically contains help for the control. `ExContextHelp` and `MaximizeBox` or `MinimizeBox` cannot be true together.

Layered (Boolean)

If true, it's a layered window. Using a layered window can significantly improve performance and visual effects for a window that has a complex shape, animates its shape, or wishes to use alpha blending effects (that is, to be partially translucent).

LayoutRTL (Boolean)

If true and for Arabic and Hebrew versions of Windows, creates a window whose horizontal origin is on the right edge (increasing horizontal values advance to the left).

Left (Boolean)

If true, the window has generic left-aligned properties. This is the default.

LeftScrollBar (Boolean)

If true and if the shell language is Hebrew, Arabic, or another language that supports reading order alignment, the vertical scroll bar (if present) is to the left of the client area. For other languages, the style is ignored.

LTRReading (Boolean)

If true, the window text is displayed using left-to-right reading-order properties. This is the default.

MDIChild (Boolean)

If true, it's a MDI child window. The multiple-document interface (MDI) technique uses a single primary window, called a parent window, to visually contain a set of related document or child windows, as shown in the figure below. Each child window is essentially a primary window, but it is constrained to appear only within the parent window instead of on the desktop. These applications allow to display multiple documents at the same time, with each document displayed in its own window.

NoActivate (Boolean)

If true, the top-level window does not become the foreground window and activated when the user clicks it. The system does not bring this window to the foreground when the user minimizes or closes the foreground window. The window does not appear on the taskbar by default. To force the window to appear on the taskbar, `AppWindow` has to be true.

NoInheritLayout (Boolean)

If true, the window does not pass its window layout to its child windows.

NoParentNotify (Boolean)

If true, a child window does not send a message⁹⁹ to its parent window when it is created or destroyed. All child windows, except those that have this attribute to true, send this message to their parent windows. By default, child windows in a dialog box have this attribute to true.

⁹⁸ The `WM_HELP` message.

⁹⁹ The `WM_PARENTNOTIFY` message.

Right (Boolean)

If true and if the shell language is Hebrew, Arabic, or another language that supports reading-order alignment, the window has generic right-aligned properties. Otherwise, the style is ignored. Using this attribute for static or edit controls has the same effect as using `TextStyle = right` or `Alignment = right` respectively. Using this attribute with button controls has the same effect as using `HorizontalAlignment = right` and `RightButton = true` (for check boxes and radio buttons).

RightScrollBar (Boolean)

If true, the vertical scroll bar (if present) is to the right of the client area. This is the default.

RTLReading (Boolean)

If true and if the shell language is Hebrew, Arabic, or another language that supports reading-order alignment, the window text is displayed using right-to-left reading-order properties. Otherwise, the style is ignored.

ToolWindow (Boolean)

If true, it's a tool window, i.e. a window intended to be used as a floating toolbar. A tool window has a title bar that is shorter than a normal title bar, and the window title is drawn using a smaller font. A tool window does not appear in the task bar or in the window that appears when the user presses ALT+TAB.

TopMost (Boolean)

If true, the window is placed above all no topmost windows and stay above them even when the window is deactivated. The dialog is then always displayed in the application. This style does not prevent the user from accessing other windows on the desktop. It is often used to signal there is a problem from the system, like lack of resources, no memory.

Transparent (Boolean)

If true, the window is transparent. That is, any windows that are beneath the window are not obscured by the window.

DIALOG

This class defines a dialog box, which is used to communicate with the user and to supply services that are too complicated to be in a menu. It's a temporary subsidiary window, and takes form of a pop-up window containing various child window controls through which the user interacts.

DlgID (either a unique name or a unique 16-bit unsigned integer value in the range 1 to 65,535)

This unique identifier (often a number) is the reference of the program to identify the dialog box.

Extended (Boolean)

If true, it's an extended dialog (i.e. with extended window styles).

X (Integer)

Specifies the X-coordinate (in horizontal dialog units¹⁰⁰) of the upper left corner of the dialog box, relative to the client area of its parent when the dialog box is invoked by the program.

Y (Integer)

Specifies the Y-coordinate (in vertical dialog units¹⁰¹) of the upper left corner of the dialog box, relative to the client area of its parent when the dialog box is invoked by the program.

Width (Integer)

Specifies the width of the dialog box (in dialog box units).

Height (Integer)

Specifies the height of the dialog box (in dialog box units).

HelpID (Integer)

Specifies a numeric expression that indicates the identifier of the dialog box during help messages processing. The value is -1 if

3DLook (Boolean)

If true, gives three-dimensional borders to child controls in the dialog box and draws the entire dialog box using the three-dimensional colour scheme

AbsoluteAlignment (Boolean)

If true, indicates that the coordinates of the dialog box are screen coordinates. If not, the system assumes they are client coordinates.

Center (Boolean)

If true, centers the dialog box in the working area.

CenterMouse (Boolean)

If true, places the dialog box so that the mouse cursor is centered in the dialog box.

ContextHelp (Boolean)

If true, includes a question mark in the title bar of the dialog box. ContextHelp designates only a placeholder. When the dialog box is created, the system checks for this style and, if it is there, adds the extended style (see ExContextHelp in the WINDOW class) to the dialog box.

DialogIsControl (Boolean)

If true, the dialog appears like a control (not child), with no frame.

FixedSys (Boolean)

By default, the system draws all text in a dialog box using the SYSTEM_FONT¹⁰² font. If true, causes the dialog box to use the SYSTEM_FIXED_FONT¹⁰³ instead of the default SYSTEM_FONT.

¹⁰⁰ One horizontal dialog unit is equal to 1/4 of the average character width.

¹⁰¹ One vertical dialog unit is equal to 1/8 of the character height.

¹⁰² This is a proportional font based on the Windows character set, and is used by the operating system to display window titles, menu names, and text in dialog boxes. The System font is always available. Other fonts are available only if they have been installed.

¹⁰³ This is a monospace font compatible with the System font in 16-bit versions of Windows.

LocalEdit (Boolean)

If true, directs edit controls in the dialog box to allocate memory from the application's data segment. Otherwise, edit controls allocate storage from a global memory object.¹⁰⁴

ModalFrame (Boolean)

Dialog boxes are either modal or modeless. If true, it's a modal dialog box (with a thick border). It demands the user's attention before anything else can be done: when displayed, the user cannot switch between the dialog box and the window that created it (the user must explicitly end the dialog box, for instance by clicking the Cancel button). The user can however switch to another program while displayed. If false, it's a modeless dialog box, allowing the user to still work on the parent window. It is similar to normal window, usually with a caption bar and a system menu, and is preferred when the user would find convenient to keep the dialog box displayed for a while.

NoFailCreate (Boolean)

If true, creates the dialog box even if errors occur (for example, if a child window cannot be created or if the system cannot create a special data segment for an edit control).

NoIdleMessage (Boolean)

If true, suppresses messages¹⁰⁵ that the system would otherwise send to the owner of a modal dialog box that is entering an idle state. A modal dialog box enters an idle state when no messages are waiting in its queue after it has processed one or more previous messages.

SetForeground (Boolean)

If true, brings the dialog box to the foreground.¹⁰⁶ This style is useful for modal dialog boxes that require immediate attention from the user regardless of whether the owner window is the foreground window.

Language (String)

A language identifier specifying the language of the dialog box.

Sublanguage (String)

A sublanguage identifier.

SetFont (Boolean)

If true, indicates that FontName (see beyond) contains the font to use for text in the client area and controls of the dialog box. If possible, the system selects a font according to the specified font data.

FontName (String)

Specifies the name of the typeface for the font that the system uses to draw text in the dialog box (this attribute enables then to set something other than the system font for use with dialog box text).

¹⁰⁴ Applies to 16-bit applications only.

¹⁰⁵ The WM_ENTERIDLE messages.

¹⁰⁶ Causes the system to use the SetForegroundWindow function. This function puts the thread that created the window into the foreground and activates the window. Keyboard input is directed to the window, and various visual cues are changed for the user. The system assigns a slightly higher priority to the thread that created the foreground window than it does to other threads.

FontSize (Integer)

Specifies the size (in points) of the font used for the text in the dialog box and its controls. A point is .013837 of an inch (roughly 1/72 inch) or 2.54 of a centimetre. The value is generally determined by measuring the distance from the bottom of a lowercase g to the top of an adjacent uppercase M, as shown in the illustration.

Italic (-1, false, true)

If true, the font is italic. The value is -1 if there is no information about this property (default value).

Weight (Integer)

Specifies the weight of the font in the range 0 through 1000. For example, 400 is normal and 700 is bold. If the value is zero, a default weight is used. The value -1 is the default value (nothing specified).

ShellFont (Boolean)

If true, indicates that the dialog box should use the system font. FontName must be set to MS Shell Dlg. Otherwise, this style has no effect. The system selects a font using the font data specified in the Fontsize, Weight, and Italic attributes.

CONTROL

This class defines controls. Every control is a child window that an application uses in conjunction with another window to perform simple input and output (I/O) tasks (type text, choose options, and direct a dialog box to complete its action). Common controls in a dialog box are push buttons, check boxes, radio buttons, edit boxes, list boxes, combo boxes, text strings and scroll bars. Such controls are predefined in Windows. When an overlapped window (program's main application window) is moved, a popup window stay up. This is not true for child windows, witch follow their parent around the display (and are never displayed outside the client area of their parent).

CtrlID (either a unique name or a unique 16-bit unsigned integer value in the range 1 to 65,535)

Is the value that the child uses to identify itself when sending messages to its parent (when you click a button with the mouse for instance, the child window control sends a message¹⁰⁷, with notification code `CLICKED`, to its parent window). The value is set to -1 when the control does not send messages back to its parent (like text and icon control witch do not accept mouse or keyboard input).

X (Integer)

Specifies the X-coordinate (in dialog box units) of the upper left corner of the child window relative to the upper left corner of the parent window's client area.

Y (Integer)

Specifies the Y-coordinate (in dialog box units) of the upper left corner of the child window relative to the upper left corner of the parent window's client area.

¹⁰⁷ The `WM_COMMAND` message.

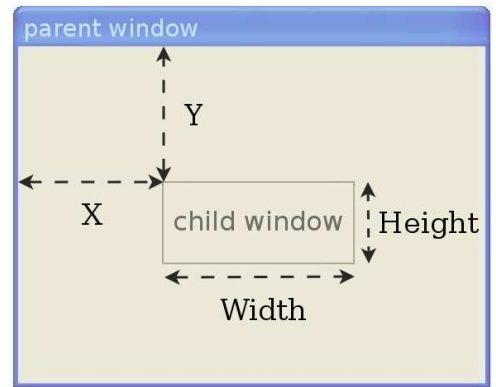
Width (Integer)
Specifies the width of the child window (in dialog box units).

Height (Integer)
Specifies the height of the child window (in dialog box units).

HelpID (Integer)
Specifies a numeric expression that indicating the identifier of the control during help messages processing.

Position (Integer)
Is a positive number that defines the order the controls are specified in the resource script file. This value is used in conjunction with the Group and TabStop attributes from the WINDOW class.

The child classes correspond to predefined window classes in Windows used to create child window controls. Additional window styles are used to define more precisely the appearance or functionality of these controls.



BUTTON

This class defines a button child window. A button control typically provides input to an application by notifying the parent window when the user clicks on the control with a mouse. It can be used either alone or in groups, and can appear either with or without a label (the text is specified in the window class).

VerticalAlignment (*{top, center, bottom, none}*)
Places text at the top, centers text vertically or places text at the bottom in the button rectangle.

HorizontalAlignment (*{left, center, right, none}*)
Left aligns the text, centers text horizontally or right aligns the text in the button rectangle (or in the remainder rectangle after removing the check box or radio button).

Flat (Boolean)
If true, the button is two-dimensional. It does not use the default shading to create a 3-D image.

Multiline (Boolean)
If true, wraps the button text to multiple lines if the text string is too long to fit on a single line in the button rectangle.

Notify (Boolean)
If true, enables a button to send notification messages¹⁰⁸ to its parent window which tells that the button has been double clicked, lost the keyboard focus or gained the keyboard focus. A

¹⁰⁸ Notification messages are submessage codes that the child use to tell the parent in more detail what the message means (like the control has been clicked, double clicked etc.). Here, the control can use the BN_DBLCLK, BN_KILLFOCUS and BN_SETFOCUS codes.

button sends always a notification message¹⁰⁹ which tells the parent that the button has been clicked, regardless of whether it has this style.

The next six classes define various types of button.

PUSHBUTTON

A push button is a rectangle with rounded corners enclosing text that the user can click with a mouse. The rounded rectangle takes up the full height and width of the dimensions given. The text is centered within the rectangle. This control is used to trigger an immediate action.¹¹⁰



Default (Boolean)

If true, it is the default push button and has a heavier outline. Pressing the Enter key has the same effect that clicking on it with the mouse. That enables the user to quickly select the most likely option (the default option).

Content ({text, bitmap, icon})

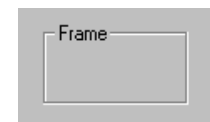
The push button can displays text, which for normal window is the text that appears in the caption bar (this is the default). It can be also labelled with an icon or a bitmap instead of text.

PUSHBOX

A push box is identical to a push button, except that it does not display a button face or frame. Only the text appears.

GROUPBOX

A group box is an oddity in the button types because it can not be selected and neither process input nor sends a message to its parent. This button is just a rectangular outline with its window text at the upper-left corner. It is used to enclose other controls.



CHECKBOX

A check box is a square box usually aligned with the left edge and centered within the top and bottom dimensions of the child window's rectangle, with text that appears to the right.¹¹¹ Clicking the box once causes an X appear, clicking again toggle the X off. There are incorporated in an application to allow a user to select options.



Auto (Boolean)

If true, the check state automatically toggles between checked and cleared each time the user selects the check box.¹¹²

¹⁰⁹ A message with the `BN_CLICKED` notification.

¹¹⁰ The button posts a `WM_COMMAND` message to the owner window when the user selects the button.

¹¹¹ Unless `LeftText=true`.

¹¹² That is the control itself toggles the check mark on and off, and `WM_COMMAND` messages are ignored by its parent. Otherwise, the program must control the checking and un-checking of the box (after receiving `WM_COMMAND`, the parent window send `BM_SETCHECK` message to the child control).

3state (Boolean)

If true, the box can display a third state: it can be dimmed as well as checked. A grey colour within the check box indicates to the user that the box cannot be checked - i.e., that it has been disabled.

RightButton (Boolean)

If true, positions the check box's square on the right side of the button rectangle (the text appears to the left).

PushLike (Boolean)

If true, makes a check box look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked.

RADIOBUTTON

A radio button looks like a check box except that it uses a circle rather than a box. A dot within the circle indicates that it has been checked. Text is usually displayed to its right.¹¹³ Only one check may be made if there are multiple buttons: groups of radio buttons are conventionally used to indicate mutually exclusive options. Unlike check boxes, they do not work as toggle (that is, when you click it a second time, its state remains unchanged).

Auto (Boolean)

If true, when the user selects it, the system automatically sets the button's check state to checked and automatically sets the check state for all other buttons in the same group to cleared.

RightButton (Boolean)

If true, positions the radio button's circle on the right side of the button rectangle (the text appears to the left).

PushLike (Boolean)

Makes a radio button look and act like a push button. The button looks raised when it isn't pushed or checked, and sunken when it is pushed or checked.

CUSTOMBUTTON

It's an owner-drawn button, that is, a button that the program is responsible for drawing.¹¹⁴

LISTBOX

This class defines child window controls called list boxes. A list box contains a collection of items¹¹⁵ displayed as a scrollable columnar list within a rectangle from which a user can chose (such filenames in the current directory when the user select Open from the File menu). The user can view and select an item by highlighting it with a left mouse click.¹¹⁶ If the list box is not large enough to display all the list box items at once, the list box can also provide a scroll

¹¹³ Unless `RightButton=true`.

¹¹⁴ The owner window receives the `WM_DRAWITEM` message.

¹¹⁵ Items can be represented by text strings, bitmaps, or both.

¹¹⁶ Selecting a list box item changes its visual appearance, usually by changing the text and background colors.

bar (if `VerticalScrollBar` from the `WINDOW` class is set to `true`¹¹⁷). By default, the list box displays only the list of items without any border around it. A border can be added with `Border` set to `true` (from the `WINDOW` class).



Sorted (Boolean)

If true, strings in the list box are sorted alphabetically.

MultipleSelection (Boolean)

If true, turns string selection on or off each time the user clicks a string in the list box. Any number of strings can be selected at a time. A list box is by default simple selection.

DisableNoScroll (Boolean)

If true, the list box shows a disabled scroll bar when the list box does not contain enough items to scroll. Otherwise, the scroll bar is hidden when the list box does not contain enough items.¹¹⁸

ExtendedSelection (Boolean)

If true, the user can select multiple items using the `SHIFT` key and the mouse or special key combinations.

Multicolumn (Boolean)

If true, it's a multi column list box that is scrolled horizontally. A multi column list box places items into as many columns as are needed to make vertical scrolling unnecessary (a vertical scroll bar is never displayed). The user can use the keyboard to navigate to columns that are not currently visible. To have a horizontal scroll bar displayed that allows the user to scroll to columns that are not currently shown in the visible region of the list box requires the addition of `HorizontalScrollBar = true` from the `WINDOW` class.

NoSelection (Boolean)

If true, the list box contains items that can be viewed but not selected.

Notify (Boolean)

If true, the list box notifies its parent whenever the user clicks a string (list boxes detect also double-clicks¹¹⁹). List boxes almost always notify their parent window about action done by the user.

OwnerDrawFixed (Boolean)

If true, the owner of the list box is responsible for drawing, sorting, and storing its contents and the items in the list box are all the same height. The owner window receives a message¹²⁰ when a visual aspect of the list box has changed.

OwnerDrawVariable (Boolean)

If true, the owner of the list box is responsible for drawing, sorting, and storing its contents and the items in the list box are variable in height. The owner window receives a message when a visual aspect of the list box has changed.

¹¹⁷ Note that a horizontal scrollbar can only be displayed with a multi column list box.

¹¹⁸ And the vertical scrollbar is added when there are more items than will fit in the visible area of the listbox.

¹¹⁹ They can sent `LBN_DBLCLK` notification code.

¹²⁰ The `WM_DRAWITEM` message.

HasStrings (Boolean)

If true, the list box contains items consisting of strings. The list box maintains the memory and pointers for the strings so the application can use a message¹²¹ to retrieve the text for a particular item. By default, all list boxes except owner-drawn list boxes have this style.

NoIntegralHeight (Boolean)

If true, the size of the list box is exactly the size specified by the application when it created the list box. By default, Windows sizes a list box so that the list box does not display partial items.

UseTabstops (Boolean)

If true, causes the list box to recognize and expand tab characters when drawing its strings.

WantKeyboardInput (Boolean)

If true, the owner of the list box receives messages¹²² whenever the user presses a key and the list box has the input focus. This enables an application to perform special processing on the keyboard input.

NoData (Boolean)

If true, it's a no-data list box. Useful when the count of items in the list box will exceed one thousand. A no-data list box resembles an owner-drawn list box except that it contains no string or bitmap data for an item.

NoRedraw (Boolean)

If true, the list box's appearance is not updated when changes are made.

EDIT

This class defines an edit child window. It's a rectangle containing editable text based on dimension of the child window. An edit box typically allows the user to enter and edit text by typing on the keyboard.

Alignment ({left,right,center})

Specifies if the text in a single-line or multiline edit control is left-justified, right-justified or centered in the rectangle.

Multiline (Boolean)

If true, it's a multiline edit control. By default, the control has a single-line entry field.

AutoHorizontalScroll (Boolean)

If true, the control automatically scrolls text to the right when the user types a character at the end of the line. When the user presses the ENTER key, the control scrolls all text back to position zero. Otherwise, only text that fits within the visible area is allowed for single-line edit controls. For multiline edit controls, if false, the text is wrapped to the beginning of the next line when necessary.¹²³ If an edit control has `HorizontalScrollBar` set to true, this property is applied automatically.

¹²¹ The `LB_GETTEXT` message.

¹²² `WM_VKEYTOITEM` messages.

¹²³ If true, the user must press the ENTER key to start a new line.

AutoVerticalScroll (Boolean)

If true, the edit control automatically scrolls text up when there is more text than can be displayed within the control. This style is applicable to multiline edit controls only. With this style off, the edit control ignores input that cannot be displayed. If an edit control has `VerticalScrollBar` set to true, this property is applied automatically.

PasswordField (Boolean)

If true, displays all characters as an asterisk (*) as they are typed into the edit control (valid only for single-line edit controls).

LowerCase (Boolean)

If true, converts all characters to lowercase as they are typed into the edit control.¹²⁴

UpperCase (Boolean)

If true, converts all characters to uppercase as they are typed into the edit control.

NoHideSelection (Boolean)

Normally, an edit control hides the selection when the control loses the input focus and inverts the selection when the control receives the input focus. If true, delete this default action (the selected text is inverted, even if the control does not have the focus).

ReadOnly (Boolean)

If true, prevents the user from entering or editing text in the edit control.

Number (Boolean)

If true, allows only digits to be entered into the edit control.

WantReturn (Boolean)

When the multiline edit control is in a dialog box, pressing the ENTER key has the same effect as pressing the default push button. If true, a carriage return is inserted when the user presses the ENTER key while entering text into a multiple-line edit control.¹²⁵

OEMConvert (Boolean)

If true, text entered in the edit control is converted from the Windows character set to the OEM character set and then back to the Windows character set. This ensures proper character conversion when the application calls the function to convert a Windows string in the edit control to OEM characters.¹²⁶

RICHEDIT

This class defines a rich edit control. It's a window in which the user can enter, edit, format, print, and save text. The text can be assigned character and paragraph formatting, and can include embedded Component Object Model (COM) objects. Rich edit controls support almost all of the messages and notification messages used with multiline edit controls. Thus, applications that already use edit controls can be easily changed to use rich edit controls.

¹²⁴ For combo box (see further), text in both the selection field and the list is converted.

¹²⁵ This style has no effect on a single-line edit control.

¹²⁶ This style is most useful for edit controls that contain filenames that will be used on file systems that do not support Unicode.

Additional messages and notifications enable applications to access the functionality unique to rich edit controls.

The following attributes are unique to rich edit controls.¹²⁷

DisableNoScroll (Boolean)

If true, the scroll bar is disabled instead of hide when it is not needed.

NoCallOleInitialize (Boolean)

If true, prevents the control from calling the OleInitialize function when created (this function initializes global data for the session and prepares the OLE libraries to accept calls).

NoIME (Boolean)

If true, disables the Input Method Editor (IME) operation (available for Asian language support only).

SelfIME (Boolean)

If true, directs the rich edit control to allow the application to handle all IME operations (available for Asian language support only).

Sunken (Boolean)

If true, displays the control with a sunken border style so that the rich edit control appears recessed into its parent window.

Vertical (Boolean)

If true, draws text and objects in a vertical direction (available for Asian-language support only).

COMBOBOX

A combo box is a combination of an edit box and a list box. It can use the AutoHorizontalScroll, LowerCase, OEMConvert and UpperCase attributes from the EDIT class, and the Sort, DisableNoScroll, HasStrings, NoIntegralHeight, OwnerDrawFixed and OwnerDrawVariable attributes from the LISTBOX class. For example, if HorizontalScroll is true, and if the combo box's edit control is completely filled with text and the user enters more text at the end of the edit control line, the existing text is automatically scrolled. Else, if the edit control is completely filled with text, no more text is allowed to be entered into the edit control.



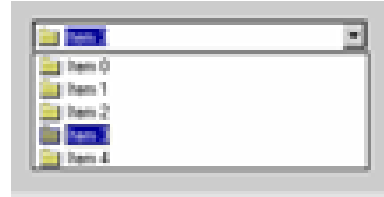
Type ({simple,dropDown,dropDownList})

There are three types of combo boxes. For a simple combo box, the list box is displayed at all times. The current selection in the list box is displayed in the edit control. A drop-down combo box is similar to simple one, except that the list box is not displayed unless the user selects an icon next to the edit control. A drop-down list box is similar to drop-down one, except that the edit control is replaced by a static-text item that displays the current selection in the list box (edit control is set to read-only).

¹²⁷ A rich edit control inherits from EDIT class, but can not have the LowerCase, OEMConvert and UpperCase attributes set to true (it does not support these styles).

Extended (Boolean)

Additionally, some combo box controls that display icons with items¹²⁸ (extended combo box controls) can use the next attributes. Note that for these controls, the EDIT- and LISTBOX-specific attributes are not supported.

CaseSensitive (Boolean)

If true, searches in the list will be case sensitive. This includes searches as a result of text being typed in the edit box.

NoEditImage (Boolean)

If true, the edit box and the dropdown list will not display item images.

NoSizeLimit (Boolean)

If true, allows the control to be vertically sized smaller than its contained combo box control. If it is sized smaller than the combo box, the combo box will be clipped.

PathwordBreakProc (Boolean)

If true, the edit box will use the slash (/), backslash (\), and period (.) characters as word delimiters. This makes keyboard shortcuts for word-by-word cursor movement effective in path names and URLs.

STATIC

This class defines a static child window. They are used to put information in the dialog and often act as labels for other controls (the text is given in the window class), but can be used to draw frames or lines separating other controls, or to display icons. They do not expect user input and do not send back messages to their parent.

Type ({text,frame,image,enhancedMetafile,ownerdraw })

The control can display text, a frame, an image or an enhanced metafile¹²⁹. The owner of the static control can also be responsible for drawing the control.

Sunken (Boolean)

If true, a half-sunken border is drawn around a static control.

Notify (Boolean)

If true, allow the parent window to receive notification messages¹³⁰ when the user clicks the control.

¹²⁸To make item images easily accessible, another type of combo box controls in Windows provide image list support (see the LISTVIEW class for image lists). These controls provide the functionality of a standard combo box without having to manually draw item graphics (they create a child combo box and perform owner draw tasks based on the assigned image list). If an image list is not assigned to the control, it displays item text only.

¹²⁹ A metafile is a collection of structures that store a picture in a device-independent format. Device independence is the one feature that sets metafiles apart from bitmaps (drawback: they are generally drawn more slowly than bitmaps, so if an application requires fast drawing and device independence is not an issue, it should use bitmaps instead of metafiles). The given text of the control is the name of a metafile. An enhanced metafile static control has a fixed size. The metafile is scaled to fit the static control's client area.

¹³⁰ Messages with STN_CLICKED, STN_DBLCLK, STN_DISABLE, and STN_ENABLE notifications.

Several attributes exist to define more precisely its layout. The first defines the basic text styles for static controls.

TextStyle (*{left,right,center,simple,leftNoWordWrap,undefined}*)

As in edit child window, the three first values specify if the text is left-justified, right-justified or centered in the rectangle. Words that extend past the end of a line are automatically wrapped to the beginning of the next line. Words that are longer than the width of the control are truncated. *simple* designates a simple rectangle and displays a single line of left-justified text in the rectangle. The line of text cannot be shortened or altered in any way.¹³¹ *leftNoWordWrap* designates a simple rectangle and displays the given text left-justified in the rectangle. Tabs are expanded, but words are not wrapped. Text that extends past the end of a line is clipped.

The next attributes can modify a static control that has any of the previous text styles.

NoPrefix (*Boolean*)

Normally, Windows will interpret an ampersand (&) character in the control's text to be a prefix character for the keyboard access key: the ampersand is removed and the next character in the string is underlined.¹³² This feature is not always wanted. If true, the static control displays an ampersand as an ordinary character.

EndEllipsis (*Boolean*)

If true, replaces part of the given string (characters at the end of the string) with ellipses, if necessary, so that the result fits in the specified rectangle.

PathEllipsis (*Boolean*)

If true, replaces part of the given string (characters in the middle of the string) with ellipses, if necessary, so that the result fits in the specified rectangle. If the string contains backslash (\) characters, preserves as much of the text after the last backslash as possible.

WordEllipsis (*Boolean*)

Truncates any word that does not fit in the rectangle and adds ellipses.

Static controls can be used to draw frames.

Frame (*{black,gray,white,undefined}*)

Defines a box that is not filled, and does not display text. The three first values do not necessarily mean that the color are black, gray and white. It is based on a system color.

black : drawn with the color used to draw window frames, the default is black.

gray : drawn with the color used to fill the screen background (desktop), the default is gray.

white : filled with the color used to fill the window background, the default is white.

Rectangle (*{black,gray,white,undefined}*)

Like Frame, except that the rectangular outline is filled (with the color used to draw window frames, with the color used to fill the desktop or with the color used to fill the window background).

Etched (*{frame,horizontal,vertical,undefined}*)

¹³¹ The control's parent window must not process the WM_CTLCOLOR message.

¹³² And the control displays two ampersands (&&) as a single ampersand.

The frame, the top and bottom edges or the left and right edges of the static control are drawn using the `EDGE_ETCHED`¹³³ edge style.

Static controls can also be used to display images.

Icon (Boolean)

If true, an icon is displayed in the dialog box. The given text is the name of the icon (not a filename). The control is automatically sized to fit the icon when it is displayed (the width and height values specified are ignored).

Bitmap (Boolean)

If true, a bitmap is displayed in the dialog box. The given text is the name of the bitmap (not a filename). The control automatically sizes itself to accommodate the bitmap (the width and height values specified are ignored).

CenterImage (Boolean)

If true and if the bitmap or icon is smaller than the client area of the static control, the rest of the client area is filled with the color of the pixel in the top left corner of the bitmap or icon.¹³⁴

RealSizeImage(Boolean)

If true, prevents a static icon or bitmap control from being resized as it is loaded or drawn. If the icon or bitmap is larger than the destination area, the image is clipped.

RightJustify (Boolean)

If true, the lower right corner of a bitmap or icon static control is to remain fixed when the control is resized. Only the top and left sides are adjusted to accommodate a new bitmap or icon.

SCROLLBAR

This class defines a scroll bar child window. A window can display a content (such as a document or a bitmap) that is larger than the window's client area. When provided with a scroll bar, the user can scroll a content in the client area to expand the viewing space. This is not a scrollbar added at the right and/or the bottom of a window, but it's a child window control that can appear anywhere in the parent window.¹³⁵



Type ({horizontal, vertical, sizeBox, sizeGrip})

There are two types of scroll bars: horizontal (by default) and vertical. If it is not aligned with one of its edges, the scroll bar has the height, width, and position specified by the window's attributes. The control can also designate a size box. If it is not aligned with one of its corners, the size box has the height, width, and position given in the window's attributes. A size grip is like a size box, but with a raised edge.

Alignment ({left, right, top, bottom, bottomRight, topLeft, none})

¹³³ Combination of two outer-border flags : `BDR_RAISEDOUTER` (raised outer edge) and `BDR_SUNKENOUTER` (sunken outer edge).

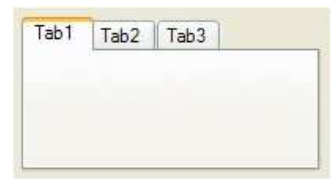
¹³⁴ If the static control contains a single line of text rather than image, the text is centered vertically in the client area of the control.

¹³⁵ Unlike a button, edit and list box control, it do not send `WM_COMMAND` messages to the parent window, but `WM_HSCROLL` and `WM_VSCROLL` messages.

left and *right* are used with a vertical scroll bar. The left (right) edge of the scroll bar is aligned with the left (right) edge of the rectangle specified when created. The scroll bar has the default width for system scroll bars. *top* and *bottom* are used with a horizontal scroll bar. The top (bottom) edge of the scroll bar is aligned with the top (bottom) edge of the rectangle specified when created. The scroll bar has the default height for system scroll bars. *bottomRight* and *topLeft* are used with size box. The lower-right (upper-left) corner of the size box is aligned with the lower-right (upper-left) corner of the rectangle specified when created. The size box has the default size for system size boxes.

TAB

A tab control is analogous to the dividers in a notebook. By using a tab control, an application can define multiple pages for the same area of a dialog box. Each page consists of a certain type of information or a group of controls that the application displays when the user selects the corresponding tab. A tab control can have specific characteristics, like the alignment and general appearance of the control's tabs.



Type (*{tabs, buttons}*)

Tabs can appear either as tabs (this is the default) or as buttons. Tabs in the second type of tab control should serve the same function as button controls (that is, clicking a tab should carry out a command instead of displaying a page). Because the display area in a button tab control is typically not used, no border is drawn around it.

Bottom (*Boolean*)

If true, tabs appear at the bottom of the control.

FixedWidth (*Boolean*)

By default, tab control automatically sizes each tab to fit its icon, if any, and its label. If true, all tabs are the same width.

FlatButtons (*Boolean*)

If true, selected tabs appear as being indented into the background while other tabs appear as being on the same plane as the background. This style is used only with button tab controls.

FocusNever (*Boolean*)

If true, specifies that the tab control does not receive the input focus when clicked.

FocusOnButtonDown (*Boolean*)

If true, the tab control receives the input focus when clicked. This attribute is typically used only with button tab controls.

ForceLeft (*{none, icon, label}*)

none centers the icon and label within each tab of the control, placing the icon to the left of the label. The icon can be aligned with the left edge of each fixed-width tab, leaving the label centered, when the attribute takes the value *icon*. Both the icon and label can be left-aligned within each fixed-width tab by the value *label*. The two last values are used only when `FixedWidth = true`.

HotTrack (*Boolean*)

If true, causes the control to exhibit hot tracking¹³⁶ behaviour (that is, an item is automatically highlighted as the mouse pointer moves over it).

Multiline (Boolean)

By default, a tab control displays only one row of tabs and the user can scroll to see more tabs, if necessary. If true, multiple rows of tabs are displayed, if necessary, so all tabs are visible at once. The tabs are left-aligned within each row (unless RightJustify = true).

Multiselection (Boolean)

If true, multiple tabs can be selected by holding down when clicking. This style is used only with button tab controls.

OwnerDrawFixed (Boolean)

If true, the parent window is responsible for drawing tabs in the control.

RaggedRight (Boolean)

If true, rows of tabs will not be stretched to fill the entire width of the control. This style is the default.

Right (Boolean)

If true, tabs appear vertically on the right side of the control. This style is used only when Vertical = true.

RightJustify (Boolean)

If true, the width of each tab is increased, if necessary, so that each row of tabs fills the entire width of the tab control. This style is used only with multiline tab controls.

ScrollOpposite (Boolean)

If true, unneeded tabs scroll to the opposite side of the control when a tab is selected.

ToolTips (Boolean)

If true, the tab control has a tool tip control¹³⁷ associated with it to provide a brief description of each tab.

Vertical (Boolean)

If true, tabs appear at the left side of the control, with tab text displayed vertically. This style is valid only when Multiline = true.

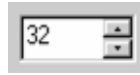
UPDOWN

An up-down control is a pair of arrow buttons that the user can click to increment or decrement an associated value (called its current position), such as a scroll position or a number displayed with a companion control (called a buddy window). To the user, an up-down control and its buddy window often look like a single control. It can be specified that an up-down control automatically position itself next to its buddy window. For example, you can

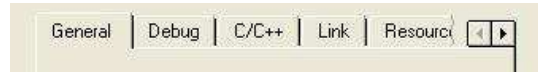
¹³⁶ An item becomes hot when the mouse passes over it. If hot tracking is enabled, the appearance of hot items changes.

¹³⁷ A tool tip is short, descriptive phrases that appear when the user holds the mouse pointer briefly over a control or another part of the user interface.

use an up-down control with an edit control as its buddy window to prompt the user for numeric input :¹³⁸



An up-down control without a buddy window functions as a sort of simplified scroll bar. For example, a tab control sometimes displays an up-down control to enable the user to scroll additional tabs into view :



Alignment (*{left, right, none}*)

Positions the up-down control next to the left (right) edge of its buddy window, the buddy window is then moved to the right (left) and its width is decreased to accommodate the width of the up-down control. The control can also be unattached (*none*).

ArrowKeys (*Boolean*)

If true, provides a keyboard interface by causing the up-down control to increment and decrement the position when the buddy window has the focus and the UP ARROW and DOWN ARROW keys are pressed.

AutoBuddy (*Boolean*)

If true, automatically selects the previous window in the z-order as the up-down control's buddy window.

Horizontal (*Boolean*)

If true, causes the up-down control's arrows to point left and right (instead of up and down) for horizontal scrolling.

HotTrack (*Boolean*)

If true, the arrows on the control are highlighted as the pointer passes over them.

NoThousands (*Boolean*)

If true, a thousands separator is not inserted between every three decimal digits.

SetBuddyInt (*Boolean*)

If true, causes the up-down control to set the text of the buddy window whenever the current position changes.¹³⁹

Wrap (*Boolean*)

By default, the current position does not change if the user attempts to increment it or decrement it beyond the maximum or minimum value. If true, causes the position to "wrap" to the opposite extreme if it is incremented or decremented beyond the ending or beginning of the range.

TRACKBAR

A track bar is a window that contains a slider (also called a thumb) and optional small indicators (called tick marks). Track bars are useful when you want the user to select a discrete value or a set of consecutive values in a range.

¹³⁸ A combination that is sometimes referred to as a spinner control.

¹³⁹ If the buddy window is a list box, an up-down control sets its current selection instead of its caption.

AutoTicks (Boolean)

A track bar displays tick marks at its beginning and end (unless the NoTicks attribute take the true value). If true, the control displays additional tick marks at regular intervals along the track bar. By default, it displays a tick mark for each increment in its range of values, but a different interval can be specified.

DownIsLeft (Boolean)

By default, the track bar control uses down equal to right and up equal to left. If true, reverses the default, making down equal left and up equal right.

EnableSelectionRange (Boolean)

If true, allows the user to select a range on the track bar (by holding the SHIFT key when dragging). This selection restricts the user to a specified portion of the total range. The logical units do not change, but only a subset of them is available for use. The tick marks at the starting and ending positions of a selection range are displayed as triangles (instead of vertical dashes), and the selection range is highlighted.¹⁴⁰

FixedLength (Boolean)

If true, allows the size of the slider to be changed.

NoThumb (Boolean)

If true, the track bar control does not display a slider.

NoTicks (Boolean)

If true, the control does not display any tick marks.

Orientation ({horizontal, vertical})

The track bar control can be oriented horizontally (this is the default orientation) or vertically.

Reversed (Boolean)

Normally, it is assumed that zero (0) percent is at the top of a vertical slider and at the left of a horizontal slider. This causes a problem when the slider's maximum (100 percent) is at the top or left side. If true, switches the values for the minimum and maximum slider positions. It has no effect on the control, but is simply a label that can be checked to determine whether a track bar is normal or reversed.

TickMarks ({bottom, top, left, right, both })

The slider is configured with a set of values from a minimum to a maximum. Therefore, the user can specify a value included in that range. Equipped with tick marks, the slider can be used to control exact values that the user can select in the range. *bottom* and *top* cause tick marks to be displayed below or above a horizontal track bar control. *left* and *right* cause tick marks to be displayed to the left or to the right of a vertical track bar control. *both* causes tick marks to be displayed on both sides of the control.¹⁴¹ There are examples of track bar controls using the *bottom* and the *both* values respectively :



¹⁴⁰ This can be useful for example to select a certain portion of a sound or video file to use.

¹⁴¹ With this values, the thumb of the slider appears as a rectangular box (with the others, one of its borders appears as an arrow).

ToolTips (Boolean)

If true, the control has a default tool tip control that displays the slider's current position.

PROGRESSBAR

A progress bar control is a window that an application can use to indicate the progress of a lengthy operation. It consists of a rectangle that is gradually filled with the system highlight color as an operation progresses.

Type ({normal, marquee, smooth})

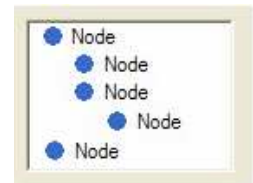
marquee causes the progress bar to move like a marquee (usually used as an indefinite progress bar). *smooth* causes the control to display a contiguous progress bar instead of the default segmented bar (*normal*). This illustrates the result of the values :

Vertical (Boolean)

If true, the progress bar displays progress information vertically, from bottom to top.

TREEVIEW

A tree view control is a window that displays a hierarchical list of items, such as the headings in a document, the entries in an index, or the files and directories on a disk. Each item consists of a label and an optional bitmapped image, and each item can have a list of subitems associated with it. By double-clicking an item¹⁴² (or by clicking its button if available), the user can expand or collapse the associated list of subitems (that is, the child items are displayed below the parent item or the child items are not displayed).¹⁴³ The starting item of the tree is called the root and represents the beginning of the tree (it is not unusual to have a tree list that has many roots). Each item (including the root) that belongs to the tree is referred to as a node.

CheckBoxes (Boolean)

If true, check boxes appear next to each item in the tree view control.

DisableDragDrop (Boolean)

If true, prevents the tree view control from sending a notification message¹⁴⁴ to the parent window when the user starts to drag an item with the left mouse button (so, prevents the control to support drag-and-drop operations).

EditLabels (Boolean)

¹⁴² By single-clicks an item label, the tree-view control selects and sets the focus to the item. Then the user can press the direction keys to navigate in the hierarchy (left and right to collapse and expand, up and down to move the focus vertically in the tree), move the mouse before releasing the mouse button to optionally drag-and-drop the item or even click again the label of the focused item to optionally edit it (a timer can make it possible for the tree view to avoid entering edit mode immediately if the user double-clicks the label).

¹⁴³ An item toggles between the expanded and collapsed state when the control sends the [TVM_EXPAND](#) message.

¹⁴⁴ With the TVN_BEGINDRAG notification.

If true, makes it possible for the user to edit the labels of tree view items.

FullRowSelect (Boolean)

If true, enables full-row selection in the tree view. The entire row of the selected item is highlighted, and clicking anywhere on an item's row causes it to be selected.

HasButtons (Boolean)

With a tree view control, the user can expand or collapse a parent item's list of child items by double-clicking the parent item. If true, adds plus (+) and minus (-) buttons to the left side of parent items showing the user whether the item is expanded or collapsed. The user can click the button once instead of double-clicking the parent item to expand or collapse a parent item's list of child items.¹⁴⁵

HasLines (Boolean)

If true, enhances the graphic representation of a tree view control's hierarchy by drawing lines that link child items to their corresponding parent item (that is, displays lines connecting items within a branch).¹⁴⁶

InfoTips (Boolean)

If true, the control sends a notification¹⁴⁷ when it is requesting additional text information to be displayed in a tool tip.

LinesAtRoot (Boolean)

If true, displays lines connecting items at the root level, showing which items play the roles of roots.¹⁴⁸

NoHorizontalScroll (Boolean)

If true, disables horizontal scrolling in the control. The control will not display any horizontal scroll bars.

NonevenHeight (Boolean)

If true, sets the height of the tree view items to an odd height. By default, the height of items must be an even value.

NoScroll (Boolean)

If true, disables both horizontal and vertical scrolling in the control. The control will not display any scroll bars.

NoToolTips (Boolean)

If true, disables the automatic tool tip feature of tree view controls. This feature automatically displays a tool tip, containing the title of the item under the mouse cursor, if the entire title is not currently visible.

RightToLeftReading (Boolean)

¹⁴⁵ Does not add buttons to items at the root of the hierarchy. To do so, HasLines and LinesAtRoot must also be set at true.

¹⁴⁶ Does not link items at the root of the hierarchy. To do so, LinesAtRoot must also be set at true.

¹⁴⁷ The TVN_GETINFOTIP notification.

¹⁴⁸ This value is ignored if HasLines = false.

Usually, windows display text left-to-right (LTR). Windows can be mirrored to display languages such as Hebrew or Arabic that read right-to-left (RTL). A tree view text is displayed in the same direction as the text in its parent window. If true, causes text to be displayed in the opposite direction from the text in the parent window.

ShowSelectionAlways (Boolean)

Normally, when the user clicks another control or another application, the node that was selected loses its selection. If true, causes a selected item to remain selected when the tree view control loses focus.

SingleExpand (Boolean)

If true, a previously expanded item will collapse automatically when a new item is expanded. If the user holds the CTRL key down while clicking an item, the item being unselected will not be collapsed.

TrackSelect (Boolean)

If true, enables hot tracking in a tree view control. Each tree node label takes on the appearance of a hyperlink (underlined with a blue color) as the mouse pointer passes over it.

LISTVIEW

A list view control is a specialized window that displays a set of items. Each item consists of an icon and a label. This window allows the user to arrange and display items (such as files) in four ways : using their large icons, using their small icons, as a list, or as a report. Note that, a resource file describing a control when the dialog box is first invoked, only one particular view can be specified, but I suppose that specific properties of other future views can still be specified.

View (*{icon, smallIcon, list, report}*)

A list view control can display items in four different views. This attribute specifies the current view. In icon view, each item appears as a standard icon with a label below it. In small icon view, each item appears as a small icon with a label to the right of it. In list view, each item appears as a small icon with a label to the right of it and is displayed in a single-column list on the screen. In report view, additional information about each item is displayed in columns to the right of the small icon and label. The user can drag the items to an arbitrary location in the window only in icon or small icon view.

Additional attributes provide other options, such as whether a user can edit labels or select more than one item at a time, whatever the view type.

Align (*{left, top}*)

Specifies the way items are arranged in icon and small icon view. Items are either aligned with the left or with the top (by default) of the control.

EditLabels (Boolean)

If true, the item text¹⁴⁹ can be edited in place. Otherwise, the user can read only.

OwnerData (Boolean)

¹⁴⁹ Only the first field of each row in the report view.

If true, the control is a virtual list view control, being able to handle millions of items because the owner receives the burden of managing item data. This allows using a list view control with large databases of information, where specific methods of data access are already in place. The control does not store any item information itself. Except for the item selection and focus information, the owner of the control must manage all item information.

ShareImageLists (Boolean)

If true, an [image list](#)¹⁵⁰ will not be deleted when the control is destroyed. This style enables the use of the same image lists with multiple list view controls.

ShowSelectionAlways (Boolean)

If true, the selection (if any) is always shown, even if the control does not have the focus.

SingleSel (Boolean)

By default, multiple items may be selected. If true, only one item at a time can be selected.

SortAscending (Boolean)

If true, item indexes are sorted based on item text in ascending order.¹⁵¹

SortDescending (Boolean)

If true, item indexes are sorted based on item text in descending order.¹⁵²

Other attributes are sometimes used to provide enhanced options such as check boxes and hot-tracking.

CheckBoxes (Boolean)

If true, displays a checkbox with each item.¹⁵³

OneClickActivate (Boolean)

If true, hot tracking is enabled (that is, when the cursor moves over an item, it is highlighted), and the user must still click the item once to select it (only one click is required to select any item, so all items may be selected).¹⁵⁴

TrackSelect (Boolean)

If true, enables hot-track selection in a list-view control. Hot track selection means that an item is automatically selected when the cursor remains over the item for a certain period of time.

¹⁵⁰ An image list is a collection of images of the same size stored in memory, each of which can be referred to by its index. Image lists are used to efficiently manage large sets of icons or bitmaps. By default, a list view control does not display item images. To display icons, image lists must be created and associated with the control. Upon creation, each image list is empty. The program repeatedly adds icons to the list, and each icon is assigned a sequential number starting at 1. This is the number to which the program refers to display a particular icon in a row or column header.

¹⁵¹ In other words, rows in list and report view will be sorted in that way. For each of its items, a list view control typically stores the image list index of the item's icons. Because in list and report views items are displayed in the same order as their indexes, the results of sorting are immediately visible to the user. In icon and small icon views item indexes are not used to determine the position of icons, and then the results of sorting are not visible.

¹⁵² idem

¹⁵³ The control then creates and sets a state image list with two images. State image 1 is the unchecked box, and state image 2 is the checked box.

¹⁵⁴ An item may be selected when it is in a state in which a single click will select it.

TwoClickActivate (Boolean)

If true, hot tracking is enabled, and the user must still click the item twice to select it (the item may be selected only after it has been clicked once).

The next tree attributes apply to list view controls in icon or small icon view.

AutoArrange (Boolean)

If true, icons are automatically kept arranged in icon and small icon view.

NoLabelWrap (Boolean)

By default, item text may wrap in icon view. If true, item text is displayed on a single line in icon view.

NoScroll (Boolean)

If true, scrolling is disabled. All items must be within the client area in icon and small icon view.

The remaining attributes apply to list view controls in report view.

NoColumnHeader (Boolean)

By default, each column has a header in report view.¹⁵⁵ If true, avoids displaying column headers.

NoSortHeader (Boolean)

If true, column headers do not work like buttons. This style can be used if clicking a column header in report view does not carry out an action, such as sorting.

OwnerDrawFixed (Boolean)

If true, the owner window can paint items in report view.¹⁵⁶

The following attributes enable enhanced options in report view.

FullRowSelect (Boolean)

If true, when a row is selected, all its fields are highlighted (item and subitems¹⁵⁷).

GridLines (Boolean)

If true, displays gridlines around rows and columns.

HeaderDragDrop (Boolean)

If true, enables drag-and-drop reordering of columns in the control.

SubItemImages (Boolean)

If true, allows images to be displayed for fields beyond the first (subitems).

HEADER

¹⁵⁵ If the user clicks one of them, the list is sorted based on the sort criterion specified for the associated column.

¹⁵⁶ In response to WM_DRAWITEM messages

¹⁵⁷ A subitem is a string that, in report view, is displayed in a column separate from the item's icon and label. All items have the same number of subitems (determined by the number of columns in the control).

A header control is a window that is usually positioned above columns of text or numbers. It contains a title for each column. The user can drag the dividers that separate the parts to set the width of each column. As example, this header has labelled columns giving detailed information about files in a directory :

Name	Size	Type	Modified
------	------	------	----------

Buttons (Boolean)

Each item in the control looks and behaves like a push button. This style is useful if an application carries out a task when the user clicks an item in the header control. For example, an application could sort information in the columns differently depending on which item the user clicks.

DragDrop (Boolean)

If true, allows drag-and-drop reordering of header items.

FilterBar (Boolean)

If true, include a filter bar as part of the standard header control (this bar allows users to conveniently apply a filter to the display).

Flat (Boolean)

If true, the header control is drawn flat when Microsoft Windows XP is running in classic mode.

FullDrag (Boolean)

If true, the header control displays column contents even while the user resizes a column.

Hidden (Boolean)

If true, indicates a header control that is intended to be hidden (it's sometimes useful to use the control as an information container instead of a visual control).¹⁵⁸

Horizontal (Boolean)

If true, the header control is oriented horizontal.

HotTrack (Boolean)

If true, enables hot tracking.

TOOLBAR

A toolbar control is a window that contains a group of buttons that bring the key functionality of an application closer to the user: we can perform common tasks with a simple click rather than performing various steps to access a menu. Typically, the buttons in a toolbar correspond to items in the main menu, providing an additional way for the user to activate an application's commands. Each button can include a bitmapped image, but also a string in addition to, or instead of, the image.

ALTDrag (Boolean)

¹⁵⁸ In fact, this does not hide the control. Instead, when the message is sent to a header control to know its layout, the control returns zero as height. It is hide by setting its height to zero.

If true, allows users to change a toolbar button's position by dragging it while holding down the ALT key. Otherwise, the user must hold down the SHIFT key while dragging a button. The Adjustable attribute must be true to enable buttons to be dragged.

CustomErase (Boolean)

If true, the control notifies its parent window about drawing operations when the window background must be erased (for example, when a window is resized) to prepare an invalidated portion of a window for painting.

Toolbar controls support a transparent look that allows the client area under the toolbar to show through. There are two kinds of transparent toolbars, ones with flat buttons and ones with three-dimensional buttons :

Flat (Boolean)

If true, it's a flat toolbar. In a flat toolbar, both the toolbar and the buttons are transparent and hot tracking is enabled. Button text appears under button bitmaps.

Transparent (Boolean)

If true, it's a non-fat transparent toolbar. In a transparent toolbar, the toolbar is transparent but the buttons are not. Button text appears under button bitmaps.

List (Boolean)

If true, it's a flat toolbar with button text to the right of the bitmap (if no text is added to the image, it is identical to Flat = true).

RegisterDrop (Boolean)

If true, the control requests a drop target object when the pointer passes over one of its buttons.¹⁵⁹

ToolTips (Boolean)

If true, adds tool tips to the toolbar control.

WrapAble (Boolean)

If true, it's a toolbar that can have multiple lines of buttons. The buttons can wrap to the next line when the toolbar becomes too narrow to include all buttons on the same line. When the toolbar is wrapped, the break will occur on either the rightmost separator or the rightmost button if there are no separators on the bar.¹⁶⁰

The size and position of the toolbar window automatically set itself. The height is based on the height of the buttons in the toolbar, the width is the same as the width of the parent window's client area and the control is positioned along the top (or bottom if specified) of the parent window's client area. Also, the toolbar window procedure automatically adjusts the size of the toolbar whenever the size of the parent window changes. The toolbar default sizing and positioning behaviours is turned off if the two next attributes are set to true.¹⁶¹

NoResize (Boolean)

¹⁵⁹ By sending a `TBN_GETOBJECT` notification message.

¹⁶⁰ The attribute must be true to display a vertical toolbar control when the toolbar is part of a vertical [rebar control](#) (see after).

¹⁶¹ This is the case for toolbar controls that are hosted by rebar controls (see after) because the rebar control sizes and positions the toolbar.

If true, prevents the control from using the default width and height when setting its initial size or a new size. Instead, the control uses its specified width and height.

NoParentAlign (Boolean)

If true, prevents the control from automatically moving to the top or bottom of the parent window. Instead, the control keeps its position within the parent window despite changes to the size of the parent.

Bottom (Boolean)

By default, the toolbar appears at the top of the parent window's client area. The toolbar control can also be positioned along the bottom of the parent window's client area (if true).

NoDivider (Boolean)

If true, prevents a two-pixel highlight from being drawn at the top of the control.

Adjustable (Boolean)

If true, the toolbar is customizable.. The user can drag a button to a new position or remove a button by dragging it off the toolbar. In addition, the user can double-click the toolbar to display the Customize Toolbar dialog box, which enables the user to add, delete, and rearrange tools to select only the ones they need and organize them in a convenient way.

REBAR

Rebar controls act as containers for other child windows (often toolbar controls). A rebar control hosts one or more bands, and each band can have any combination of a gripper bar, a bitmap, a text label, and a child window. However, bands cannot contain more than one child window. With both toolbar and rebar controls, applications are more flexible. Toolbars can be moved, repositioned, minimized, and maximized within the rebar control. This is for example a rebar control with two bands one that contains a combo box and another that contains a toolbar :



AutoSize (Boolean)

If true, the layout of a band will automatically change when the size or position of its control changes.

BandBorders (Boolean)

If true, the rebar control displays narrow lines to separate adjacent bands.

DoubleClickToggle (Boolean)

If true, the rebar band will toggle its maximized or minimized state when the user double-clicks the band. Otherwise, the maximized or minimized state is toggled when the user single-clicks on the band.

FixedOrder (Boolean)

If true, the rebar control always displays bands in the same order. A user can move bands to different rows, but the band order is static.

NoDivider (Boolean)

If true, prevents a two-pixel highlight from being drawn at the top of the control.

RegisterDrop (Boolean)

If true, the control generates notification messages when an object is dragged over a band in the control.¹⁶²

VarHeight (Boolean)

If true, the control displays bands at the minimum required height, when possible (the control can then have variable band height). Otherwise, the rebar control displays all bands at the same height, using the height of the tallest visible band to determine the height of other bands.

VerticalGripper (Boolean)

If true, the size grip will be displayed vertically instead of horizontally in a vertical rebar control.

Vertical (Boolean)

If true, the control is displayed vertically.

STATUTBAR

A status bar is a horizontal window at the bottom of a parent window in which an application can display various kinds of status information. The status bar can be divided into parts to display more than one type of information (like help text and the current cursor position). You can see such control in the main window of Rational Rose shown above.

SizeGrip (Boolean)

If true, the status bar control will include a sizing grip at the right end of the status bar. A sizing grip is similar to a sizing border. It is a rectangular area that the user can click and drag to resize the parent window.

ToolTips (Boolean)

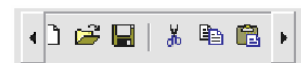
If true, enables tool tips.

Top (Boolean)

The default position of a status bar is along the bottom of the parent window's client area. If true, the control appears at the top of the parent window's client area.

PAGER

A pager control is a window container that is used with a window that does not have enough display area to show all of its content. The pager control allows the user to scroll to the area of the window that is not currently in view. For example, if an application has a toolbar that is not wide enough to show all of its items, the toolbar can be assigned to a pager control and the user will be able to scroll to the left or right to access all of the items.

AutoScroll (Boolean)

¹⁶² The RBN_GETOBJECT notification message.

If true, the pager control will scroll when the user hovers the mouse over one of the scroll buttons.

DragAndDrop (*Boolean*)

The contained window can be a drag-and-drop target. The pager control will automatically scroll if an item is dragged from outside the pager over one of the scroll buttons.

Direction (*{horizontal, vertical}*)

Indicates if the pager control scrolls horizontally or vertically.

DATETIMEPICKER

A date and time picker (DTP) control provides a simple and intuitive interface through which to exchange date and time information with a user (the user can select a value instead of typing it). The control relies on a format string to determine how it will display fields of information. For example, to display the current time with the format "07:50:23 PM" and the current date with the format "Monday June 20, 2005", format strings can be "hh':'m':'s tt" and "dddd MMM dd', ' yyyy".¹⁶³ Date and time format elements will be replaced by the actual date and time. Shown thereafter are two separate DTP controls.



AppCanParse (*Boolean*)

The user may want to edit the date value of the control. A DTP control is equipped to verify the types of values that can be entered.¹⁶⁴ If true, doesn't let the control to make this task, but allows the owner to parse user input and take necessary action. It enables users to edit within the client area of the control when they press the F2 key.

Format (*{longDate, shortDate, shortDateCentury, time }*)

There are three preset formats available for displaying the date and one for displaying time (they cannot be combined).¹⁶⁵ *longDate* displays the date in long format (like "Monday, June 20, 2005"). *shortDate* displays the date in short format (like "6/20/05"). *shortDateCentury* is similar except the year is a four-digit field (like "6/20/2005"). *time* display the time (like "7:50:23 PM").

RightAlign (*Boolean*)

By default, the DTP control displays a combo box. If the user clicks the arrow on the control, a drop-down month calendar displays (see after). If true, the calendar will be right-aligned with the control instead of left-aligned, which is the default.

ShowNone (*Boolean*)

It is possible to have no date currently selected in the control. If true, the control displays a check box that users can check once they have entered or selected a date. Until this check box is checked, the application will not be able to retrieve the date from the control because, in essence, the control has no date.

¹⁶³ "h" acts for one- or two-digit hour in 12-hour format, "m" for one- or two-digit minute, "s" for one- or two-digit second, "tt" for two-letter AM/PM abbreviation, "dd" for two-digit day (single-digit day values are preceded by a zero) and "dddd", "MMMM" and "yyyy" for full weekday, full month name and full year respectively.

¹⁶⁴ For example, the user cannot type the name of a month, only a number and the control would display the corresponding name of the month.

¹⁶⁵ If the preset formats are not sufficient, a custom format can be created.

UpDown (Boolean)

If true, displays an up-down control to the right of the control (which is divided in different sections that can each be changed individually) in place of the drop-down month calendar, as shown here:

MONTHCALENDAR

A month calendar control implements a calendar-like user interface. This provides the user with a very intuitive and recognizable method of entering or selecting a date. The title bar of the control provided by the Win32 API displays two buttons and two labels. The left (right) button allows the user to select the previous (next) month.¹⁶⁶ The left (right) label displays the currently selected month (year). To select any month of the current year, the user can click the name of the month. This displays the list of months and allows the user to choose one. To select a year, the user clicks the year number. This changes the year label into a spin button. To select a date on the control, the user clicks the desired date, which changes from the previous selection. The user can still click the bottom label to return to today's date if at one time the calendar is displaying a date other than today.

DayState (Boolean)

If true, the control requests information about which days should be highlighted by displaying them in bold.¹⁶⁷

Multiselection (Boolean)

If true, the control allows the user to select a range of days. By default, the user can select seven contiguous days maximum.

NoToday (Boolean)

If true, the control doesn't display the label at the bottom of the control (the "today" date).

NoTodayCircle (Boolean)

If true, the control no longer circles the current day.

WeekNumbers (Boolean)

If true, the control displays week numbers (1-52) to the left side of each row of days.

ANIMATION

An animation control is a window that displays an AVI clip¹⁶⁸ that does not contain audio. One common use for an animation control is to indicate system activity during a lengthy operation. This is possible because the operation thread continues executing while the AVI clip is displayed. For example, the Find dialog box of Explorer displays a moving magnifying glass as the system searches for a file. This control is still used when a file copy is underway:

AutoPlay (Boolean)

¹⁶⁶ The control can also display more than one month (if its width and height provide space).

¹⁶⁷ By sending MCN_GETDAYSTATE notifications to know how individual days should be displayed.

¹⁶⁸ An AVI (Audio-Video Interleaved) clip is a series of bitmap frames like a movie.

If true, starts playing the animation as soon as the AVI clip is opened.

Center (Boolean)

If true, centers the animation in the animation control's window.

Transparent (Boolean)

If true, allows an animation's background color to match that of the underlying window, creating a "transparent" background.¹⁶⁹

CUSTOMCONTROL

This class regroups any other controls that may appear in a dialog box. So far I've introduced existing controls (corresponding to predefined classes in Windows programming¹⁷⁰), but nothing prevents us from using a customized child window.¹⁷¹ An example is the calendar used by Windows XP in the Date and Time Properties dialog box¹⁷²:



This class may also include other existing Windows controls not listed here.¹⁷³ A custom control inherits the attributes specific to the WINDOW class (because this is a child control, has Style=child) and the CONTROL class (defining its classes position), and has one more attribute `ClassName`.

ClassName (String)

Designates the name of the class defining the control (example: `msctls_hotkey32` for a hot key control).

MENUBAR

The next classes concern the menu of an application. This one defines a window's menu bar, also called the main menu or the top-level menu. A menu bar is displayed below the caption bar and shows a list of menus (which in turn can show submenus) that lets a user select commands. Items on the menu bar are almost always pop-up menus (rarely command items).

¹⁶⁹ The control will send a message to its parent. It interprets the upper-left pixel of the first frame as the animation's default background color, and will remap all pixels with that color to the value supplied in response of the message.

¹⁷⁰ These classes already exist within Windows. The programmer does not first define and register its own window class to create a control based on one of these classes. The programmer simply uses the window class name (like "button" or "static"), and some style flags, as parameters in the function to create a window.

¹⁷¹ By registering your own window class called *ClassName*, and using your own function to process messages for your customized control.

¹⁷² The class name is "CalWndMain".

¹⁷³ Like the IP address control implemented in recent versions of `Comctl32.dll` (and defined in `Commctrl.h`), or a hot key control (a window that enables the user to enter a combination of keystrokes to perform an action quickly). They have no specific style.

MenuID (*either a unique name or a unique 16-bit unsigned integer value in the range 1 to 65,535*)

Is the name identifying the menu (used to find menu data in program resources).

Language (*String*)

Is a language identifier that specifies the language of the menu.

Sublanguage (*String*)

Is a sublanguage identifier.

POPUPMENU

A popup menu can contain menu items and other popup menus (which are displayed when the user highlight it). Items on the menu bar almost always invoke a popup menu (also called drop-down menus).

Text (*String*)

This is what appears in the menu. An ampersand (&) inside the value designates the letter that follows as the mnemonic.

State (*{enabled, disabled, grayed}*)

A popup menu can be enabled (by default), disabled or grayed. When an item is not available to the user, it is grayed or disabled. A disabled item looks just like an enabled item. When the user clicks on a disabled item, the item is not selected, and nothing happens. A grayed item is displayed in gray text.¹⁷⁴

Position (*Integer*)

Each popup menu is located in a specific position. The leftmost item in the menu bar, or the top item in a popup menu, has Position = 1. The position value is incremented for subsequent items.

MENUITEM

This class define a command item, which is checked or cleared to indicate whether an option is (not) in effect, or which can invoke a dialog box to obtain input from the user that can't be easily managed through the menu

ItemID (*Integer*)

Is the number with which the item is identified by the program.

Text (*String*)

This is what appears in the menu. The string can contain this escape characters: \a to right-align the following text and \t for a tab. An ellipsis (...) is habitually added to the end to indicate that a menu item invokes a dialog box. An ampersand (&) designates the letter that follows as the mnemonic.

Checked (*Boolean*)

¹⁷⁴ The corresponding submenu is not displayed when the item is disabled or grayed.

If true, a check mark (✓) is to the left of the menu text. This is useful to choose different program options from the menu.

State (*{enabled, disabled, grayed}*)

Menu items can be enabled (by default), disabled or grayed. Enabled or disabled items look the same to the user (a disabled item is displayed but it cannot be selected). Disabled and grayed item are used when options are not currently valid (but a grayed item is displayed in gray text to let the user know the option is not valid).¹⁷⁵

Position (*Integer*)

Each menu item is located in a specific position. The leftmost item in the menu bar, or the top item in a popup menu, has Position = 1. The position value is incremented for subsequent items.

SEPARATOR

This class defines an inactive item that serves as a dividing bar between two items.

Position (*Integer*)

Each separator is located in a specific position. The leftmost item in the menu bar, or the top item in a popup menu, has Position = 1. The position value is incremented for subsequent items, including separators.

¹⁷⁵ The WM_COMMAND message is not send to the owner window when the item is disabled or grayed.

Appendix D

Source code

```

/*****
// This class, containing a main method, can be used to test my implementation without launching GrafiXML.
// It has not to be placed in the .jar when integrating the plug-in into GrafiXML.
// Put before running it a valid resource file in C:\\ named fichier.rc
*****/

import importresources.RcFile;
import importresources.SdfFile;
import importresources.XibFile;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.File;

public class plugin {

    public static void main(String[] args) {

        File resourceFile = displayDialog();

        if ((resourceFile.getName()).endsWith(".rc")){
            // a Windows .rc file is imported
            int[] filePointers;
            try{
                // scan the file to locate resources and mark those selected by the user
                filePointers = RcFile.scan(resourceFile);
                if (filePointers.length != 0){
                    // one or more resource has been selected by the user
                    try{
                        RcFile.process(filePointers);
                    }
                    catch (FileNotFoundException e) {
                        System.out.println("File " + resourceFile.getName() + " not found!");
                    }
                    catch (IOException e) {
                        System.out.println(e);
                    }
                }
            }
            catch (FileNotFoundException e) {
                System.out.println("File " + resourceFile.getName() + " not found!");
            }
            catch (IOException e) {
                System.out.println(e);
            }
        }

        // Other formats of resource files
        // for future developement
        else
            // Apple resource files
            if ((resourceFile.getName()).endsWith(".nib")) NibFile.process(resourceFile);
            // Screenshot definition files
            else SdfFile.process(resourceFile);
    }

    public static File displayDialog(){ // fichier pour tests
        return new File("C:\\fichier.rc");
    }
}

```



```

//*****
// RcFile.java
// This class contains methods to read a Windows .rc file and to process it in CUI into GrafiXML
//*****
Author : Julien Marion

package importresources;

import java.io.File;
import java.io.FileReader;
import java.io.LineNumberReader;
import java.io.IOException;
import java.io.StreamTokenizer;
import java.util.StringTokenizer;
import java.util.Vector;

public class RcFile {

    private static File fileName;
    private static boolean dialogSelected = false;
    private static FlagTrie T = null;
    //-----
    // Scan a specified file for resources and return an array of line numbers that locate resources the user has chosen
    // to import in GrafiXML
    //-----
    public static int[] scan(File f) throws IOException{

        fileName = f;

        // Create the LineNumberReader to read from the supplied filename.

        // Converts the original input stream into a character input stream.
        FileReader r = new FileReader (fileName);
        // Transform it into a LineNumberReader, which allows to use the readLine method to get an entire line of
        // character input in one operation, and keeps track of line numbers.
        LineNumberReader lnr = new LineNumberReader (r);

        // This vector contains Resource objects.
        // It will dynamically grow as a new resource (of type menu or dialog)is found in the file.
        // It will dynamically shrink as the user decide to import less resources than those found.
        Vector resources = new Vector(25);

        // Scan the file for resources (fill vector).
        readLines(lnr, resources);

        r.close();

        /*test*/
        for (int i = 0; i < resources.size(); i++){
            System.out.println(((Resource)resources.elementAt(i)).toString());
            System.out.println("-----");
        }
        System.out.println("\n\n");

        // Call the method which create the dialog box for resources selections.
        displayDialog(resources);

        /*tests*/
        for (int i = 0; i < resources.size(); i++) {
            System.out.println(((Resource)resources.elementAt(i)).toString());
            System.out.println("-----");
        }
        System.out.println("\n\n");

        // Return the line numbers where are the selected resources to process.
        int[] pointers;
        int n = resources.size();
        pointers = new int[n]; // If resources is empty, pointers.length() equals zero.
        for (int index = 0; index < n; index++){
            // There is at least one resource to process.
            pointers[index] = ((Resource)resources.elementAt(index)).getLine();
            if (!dialogSelected && ((Resource)resources.elementAt(index)).getType() == 'D')
                dialogSelected = true;
        }

        return pointers;
    }

    //-----
    // Scan the file for resources.
    // Paragraph are separated by line without token, comments at begining are not part of paragraph.
    //-----
    private static void readLines(LineNumberReader r, Vector v) throws IOException{
        String delims = " \t";
        String line; // Contents of a line, not including any line-termination characters.
        String firstToken, secondToken;
        StringTokenizer st;
        while((line = r.readLine()) != null){ // Line is not null, the end of the stream has not been reached.
            //if (line.length() == 0);
            // '/n' (part of paragraph separator), not usefull because st.hasMoreToken()==false

```

```

st = new StringTokenizer (line, delims);
if (st.hasMoreTokens()) {
    // line contains at least one token (that is, line is not part of a paragraph
    // separator containing '\n' and eventual withe space charachters).
    firstToken = st.nextToken();
    if (firstToken.startsWith("//"));
    // Text after // and that extends to the end of the line is discarded.
    else
        if (firstToken.startsWith("/*"))
            // Text between successive occurrences of /* and */, and after */ until EOL,
            // is discarded.
            while (line != null && line.indexOf("*/") == -1) line = r.readLine();
        else{
            // Beginning of a paragraph (after eventual skiped comments).
            if (st.hasMoreTokens()){
                secondToken = st.nextToken();
                // If there is matching, store the resource with the current line.
                if (secondToken.equals("DIALOG") ||
                    secondToken.equals("DIALOGEX") ||
                    secondToken.equals("MENU"))
                    // The current line is normally r.getLineNumber()-1, because
                    // after reading a line, the position in the file is set at
                    // the beginning of the following line. But this value has to
                    // be considered for consistency with the streamTokenizer
                    // (which counts the lines starting from 1 and not 0)
                    // used to process resources.
                    v.add(new Resource(r.getLineNumber(),
                                        firstToken,secondToken.charAt(0)));
            }
            // Skip lines of the same paragraph.
            do line = r.readLine();
            while(line != null && line.length() != 0);
        }
    }
}

}

//-----
// Create a dialog box to get user choices. For each object in the vector resources, an item is created in a list.
// If the user pushes on the Ok button, the vector is updated (suppress not selected objects).
// If the user pushes on the Cancel button, the vector becomes empty.
//-----
private static void displayDialog(Vector v){
    ResourcesSelection frame = new ResourcesSelection(v);
    frame.show();
}

//-----
// process the selected resources
//-----
public static void process(int[] p) throws IOException{
    // Create the tokenizer to read from the supplied filename
    // (note that a StreamTokenizer can return the line number)
    LineNumberReader lnr = new LineNumberReader (new FileReader(fileName));
    StreamTokenizer st = new StreamTokenizer (lnr);

    // Prepare the tokenizer for recourse script style tokenizing rules (specifies how tokens
    // are recognized). By default, the tokenizer already remove quotes around strings when reading
    // them ('\'' denotes a quotes string by default) and parse number.

    // End of lines are treated as tokens (the nextToken method returns TT_EOL and also
    // sets the ttype field to this value when an end of line is read). The default syntax
    // table is configured to treat an end of lines character as white space, not as a separate token.
    st.eolIsSignificant(true);
    //
    // Specifies a range of characters to be treated as part of words
    // 'A' to 'Z' and 'a' to 'z' are already (by default) considered to be alphabetic.
    st.wordChars('_', '_'); // to recognize flags (e.g. WS_POPUP)
    //
    // Specifies the range of characters that serve to delimit tokens
    // ' ' (space) and '\t' (tab) are by default delimiters
    st.whitespaceChars('|', '|');
    st.whitespaceChars('=', '='); // Restorator use EXSTYLE=<ex_style>*
    //
    // Specifies the range of characters that are never part of tokens and should be returned as-is
    st.ordinaryChars(' ', ' ');
    st.ordinaryChars('{', '{');
    st.ordinaryChars('}', '}');
    //
    // Causes comments (//<line> and /*<text>*/) to be ignored
    st.slashSlashComments(true);
    st.slashStarComments(true);

    lnr.setLineNumber(1); // the first line is 1 (0 by default)

    // Process the resources

```

```

if (dialogSelected) T = new FlagTrie();// creer l'arbre des flags

String id;
// For each stored line number.
for (int i = 0; i < p.length; i++){
    // Reach the resource beginning at line i.
    System.out.println("Next stored line in the vector: " +p[i]); int ln ;
    while ((ln = lnr.getLineNumber()) < p[i]) {lnr.readLine();System.out.println("line " +ln);}
    System.out.print("From line " +ln+": ");

    // Read the identifier of the menu or of the dialog.
    st.nextToken();
    if(st.ttype == StreamTokenizer.TT_NUMBER) id = String.valueOf((int)st.nval);
    else id = st.sval;

    // Read the type of the resource (the token is a word).
    st.nextToken();
    if (st.sval.equals("MENU")){
        System.out.println("MENU "+id);
        processMenu(id, lnr, st);
    }
    else{
        // It's a resource of type dialog box.
        System.out.println("DIALOG "+id);
        processDialog(id, st.sval.endsWith("EX"), lnr, st);
    }
}
lnr.close();
}

//-----
// Process a resource of type menu.
//-----
private static void processMenu(String id, LineNumberReader lnr, StreamTokenizer st){
}

//-----
// Process a resource of type dialog box.
//-----
private static void processDialog(String id,boolean extended,LineNumberReader r,StreamTokenizer st)throws IOException{

    short x, y, width, height;
    int styles = 0, extendedStyles = 0;
    String text = null, language = null, fontName = "Tahoma";
    byte fontSize = 8;
    boolean bold = false, italic = false;
    String ctrlClass = null;

    // Parse the position and dimention numbers.
    st.nextToken(); x = (short)st.nval; System.out.println("x-coordinate = "+x);
    st.nextToken(); //skip ','
    st.nextToken(); y = (short)st.nval; System.out.println("y-coordinate = "+y);
    st.nextToken(); //skip ','
    st.nextToken(); width = (short)st.nval; System.out.println("width = "+width);
    st.nextToken(); //skip ','
    st.nextToken(); height = (short)st.nval; System.out.println("height = "+height);
    // Go to the beginning of the next line (optional [,helpId] is skiped).
    while(st.nextToken() != StreamTokenizer.TT_EOL);
    boolean hasControls = true;

    // Read the first keyword of each line and extract related information until the control definition
    // part is reached.
    while (st.nextToken() != '{') {
        switch (st.ttype) {
            case StreamTokenizer.TT_WORD:
                if (st.sval.equals("STYLE")){
                    // Extract the styles of the dialog box.
                    while(st.nextToken() != StreamTokenizer.TT_EOL){
                        styles |= T.match(st.sval).getFlagValue(); // bitwise OR operation
                        st.nextToken();
                    }
                    System.out.println("styles:      "+styles);
                }
                else if (st.sval.equals("CAPTION")){
                    // Extract the text that appears on the caption bar.
                    st.nextToken();
                    text = st.sval;
                    System.out.println("caption = " + text);
                    // Go to the the next line.
                    st.nextToken(); // End-of-line token.
                }
                else if (st.sval.equals("EXSTYLE")){
                    // Extract the extended styles of the dialog box.
                    while(st.nextToken() != StreamTokenizer.TT_EOL){
                        extendedStyles |= T.match(st.sval).getFlagValue(); // bitwise OR operation
                        st.nextToken();
                    }
                }
        }
    }
}

```

```

        System.out.println("extended styles:    "+extendedStyles);
    }
    else if (st.sval.equals("LANGUAGE")){
        // Extract the language used in the dialog box.
        st.nextToken();
        language=(st.sval.substring(st.sval.indexOf("_")+1,st.sval.length())).toLowerCase();
        System.out.println("language = "+ language);
        // Go to the the next line.
        while(st.nextToken() != StreamTokenizer.TT_EOL);
    }
    else if (st.sval.equals("FONT")){
        // Extract the font that is used for controls in the dialog.
        st.nextToken();
        fontSize = (byte)st.nval;
        st.nextToken(); st.nextToken();
        fontName = st.sval;
        if(st.nextToken() == ','){// There is information about weight and italic.
            st.nextToken();
            if (st.ttype == StreamTokenizer.TT_WORD)
                // FW_SEMIBOLD, FW_DEMIBOLD, FW_BOLD, FW_EXTRABOLD,
                // FW_UTRABOLD, FW_HEAVY or FW_BLACK
                if (st.sval.indexOf('B') != -1 || st.sval.equals("FW_HEAVY")) bold = true;
            else
                // st.ttype == StreamTokenizer.TT_NUMBER
                if (st.nval >= 550.0) bold = true;
            st.nextToken(); st.nextToken();
            if (st.sval.equals("TRUE")) italic = true;
            //Go to the the next line.
            while(st.nextToken() != StreamTokenizer.TT_EOL);
        } // Else it was an end of line token.
        System.out.println("font = "+ fontSize + ", " + fontName + " " + bold + " " + italic);
    }
    else if (st.sval.equals("BEGIN")){ // Begin...END is sometimes used by Resource Builder instead of {...}
        st.ttype='!';
        // Causes the next call to the nextToken method to reread the token but return the
        // current value in the ttype field, and not to modify the value in the nval or sval field.
        st.pushBack(); }
    else if (st.sval.equals("MENU") || st.sval.equals("CLASS")){
        // Other unfrequent but possible keywords in the dialog template (not implemented),
        // the line is skiped. Example of scenario :
        // ...
        // 400 DIALOG 30, 73, 275, 84
        // STYLE DS_SETFONT|DS_MODALFRAME|DS_3DLOOK|WS_POPUPWINDOW|WS_CAPTION
        // CAPTION "Find"
        // FONT 8, "MS Shell Dlg"
        // MENU 403 //include a menu in the dialog box. it has to be defined in the menu resource
        // ...
        r.readLine();
    }
    else {
        // It is an id. Another resource has been reached because an empty line resources separator has
        // not been respected and will not be processed (because not referenced in the pointers array).
        // Example of scenario :
        // ...
        // 400 DIALOG 30, 73, 275, 84
        // STYLE DS_SETFONT|DS_MODALFRAME|DS_3DLOOK|WS_POPUPWINDOW|WS_CAPTION
        // CAPTION "Find"
        // FONT 8, "MS Shell Dlg"
        // ABOUT DIALOG 26, 41, 350, 242
        // ...
        hasControls = false;
        st.ttype='!';
        st.pushBack();
    }
    break;
case StreamTokenizer.TT_EOF: // Dialog box without child controls definitions, end of file has been reached.
    hasControls = false;
    st.ttype='!';
    st.pushBack();
    break;
case StreamTokenizer.TT_EOL:
    // Skip an empty line that has been reached (not normally occurs in a valid syntax dialog box template).
    // Example of scenario:
    // ...
    // 400 DIALOG 30, 73, 275, 84
    // ...
    // STYLE DS_SETFONT|DS_MODALFRAME|DS_3DLOOK|WS_POPUPWINDOW|WS_CAPTION
    // ...
    // CAPTION "Find"
    // FONT 8, "MS Shell Dlg"
    // ...
    // {
    // ...
break;
case StreamTokenizer.TT_NUMBER: // It's an id. Another resource has been reached because an empty line
    hasControls = false; // resources separator is not respected in the file. Example of scenario:
    st.ttype='!'; // ...
    st.pushBack(); // 400 DIALOG 30, 73, 275, 84

```

```

        break;                // CAPTION "Find"
    }                        // 401 DIALOG 36, 44, 285, 110
    }                        // ...

// Generate into GrafiXML this dialog box.
Grafi.generateDialog(id, width, height, text, styles, extendedStyles);
// à faire (renvoyer aussi référence du container),
// à mettre dans une variable passée en paramètre lors des appels de création des composants

// Specify the font that will be used in the container generated.
Grafi.fontName = fontName;Grafi.fontSize = fontSize;Grafi.bold = bold;Grafi.italic = italic;

// Process the child window controls.
if(hasControls){
    // ( or BEGIN) has been encountered in the template. There is then a controls definition part.

    styles = 0; extendedStyles = 0;
    text = ""; // The size of the text field (always specified in the generic notation) is 0 when no value.

    // TO DO: A first scan storing x, y, w, h in a tab of vector to construct boxes into the CUI model.
    // r.mark(1024); // enough?

    // Read the next token different from EOL.
    while(st.nextToken() == StreamTokenizer.TT_EOL);

    while ( st.ttype != ')' ) {
        // It remains at least one line defining a child window control. Process the next line.
        // st.ttype = StreamTokenizer.TT_WORD (the key word CONTROL or a control type)
        String word = st.sval;
        if (word.equals("CONTROL")){
            System.out.println("héhoouo ?? bordel!");
            // The line format is:
            // CONTROL text,id,ctrlClass[,styles],x,y,width,height[,extendedStyles[,helpId]]
            st.nextToken(); text = st.sval;
            st.nextToken(); //skip ','
            System.out.print("text = "+text);
            st.nextToken();
            if(st.ttype == StreamTokenizer.TT_NUMBER) id = String.valueOf((int)st.nval);
        else id = st.sval; System.out.print(", id = "+id);
            st.nextToken(); //skip ','
            st.nextToken(); ctrlClass = st.sval.toLowerCase();
            System.out.print(", classe = "+ctrlClass);
            st.nextToken(); //skip ','
            st.nextToken();
            if(st.ttype != StreamTokenizer.TT_NUMBER){
                while(st.ttype != ','){
                    if (st.sval.equals("NOT")){
                        st.nextToken();
                        // be sure it is present (else will add the flag)
                        styles |= T.match(st.sval).getFlagValue();
                        styles ^= T.match(st.sval).getFlagValue();
                    }
                    else styles |= T.match(st.sval).getFlagValue();
                }
                st.nextToken();
                System.out.print(", styles:      "+styles);
                st.nextToken();
            }
            x = (short)st.nval; System.out.print(", x-coordinate = "+x);
            st.nextToken(); //skip ','
            st.nextToken(); y = (short)st.nval; System.out.print(", y-coordinate = "+y);
            st.nextToken(); //skip ','
            st.nextToken(); width = (short)st.nval; System.out.print(", width = "+width);
            st.nextToken(); //skip ','
            st.nextToken(); height = (short)st.nval; System.out.print(", height = "+height);
            st.nextToken();
            if (st.ttype == ',') st.nextToken(); //skip optional ','
            while(st.ttype != StreamTokenizer.TT_EOL && st.ttype != ','){
                extendedStyles |= T.match(st.sval).getFlagValue(); // bitwise OR operation
                st.nextToken();
            }
            System.out.println(", extended styles: "+extendedStyles);
            // Go to the beginning of the next line (optional [,helpId] skipped).
            while(st.ttype != StreamTokenizer.TT_EOL)st.nextToken();
        }
    else {
        // Shortcut notation.
        //The line format is:
        // <control_type> [text,]id,x,y,width,height[,styles[,extendedStyles[,helpId]]]
        styles |= 0x50000000; // WS_CHILD and WS_VISIBLE for any type of control.
        // Text field in the line is excuded for EDITTEXT, LISTBOX, COMBOBOX and SCROLLBAR.
        boolean textField = true;
        switch (word.charAt(0)){
            case 'A':
                ctrlClass = "button";
                if (word.charAt(4)=='C')
                    // AUTOCHECKBOX, implies the flags BS_AUTOCHECKBOX and WS_TABSTOP.

```

```

        styles |= 0x10003;
    else
        // AUTORADIOBUTTON, implies the flag BS_AUTORADIOBUTTON.
        styles |= 9;
    break;
case 'C':
    if (word.charAt(1)=='H'){
        // CHECKBOX, implies the flags BS_CHECKBOX and WS_TABSTOP.
        ctrlClass = "button";styles |= 0x10002;
    }
    else if (word.charAt(1)=='O'){
        // COMBOBOX, implies the flags CBS_SIMPLE and WS_TABSTOP.
        ctrlClass = "combobox";styles |= 0x10001;textField = false;
    }
    else {
        // CTEXT, implies the flags SS_CENTER and WS_GROUP.
        ctrlClass = "static";styles |= 0x20001;
    }
    break;
case 'D':
    // DEFPUSHBUTTON, implies the flags BS_DEFPUSHBUTTON and WS_TABSTOP.
    ctrlClass = "button";styles |= 0x10001;break;
case 'E':
    // EDITTEXT, implies the flags ES_LEFT (value 0), WS_BORDER and WS_TABSTOP.
    ctrlClass = "edit";styles |= 0x810000;textField = false;break;
case 'G':
    // GROUPBOX, implies the flag BS_GROUPBOX.
    ctrlClass = "button";styles |= 7;break;
case 'I':
    // ICON, implies the flags BS_ICON and WS_GROUP.
    ctrlClass = "static";styles |= 0x20003;break;
case 'L':
    if (word.charAt(1)=='T'){
        // LTEXT, implies the flags SS_LEFT (value 0) and WS_GROUP.
        ctrlClass = "static";styles |= 0x20000;
    }
    else{
        // LISTBOX, implies the flags LBS_NOTIFY, WS_BORDER and WS_VSCROLL.
        ctrlClass = "listbox";styles |= 0xA00001;textField = false;
    }
    break;
case 'R':
    if (word.charAt(1)=='A'){
        ctrlClass = "button";styles |= 4;
        // RADIOBUTTON, implies the flag BS_RADIOBUTTON.
    }
    else{
        ctrlClass = "static";styles |= 0x20002;
        // RTEXT, implies the flags SS_RIGHT and WS_GROUP.
    }
    break;
case 'P':
    // PUSHBUTTON, implies the flags BS_PUSHBUTTON (value 0) and WS_TABSTOP.
    ctrlClass = "button";styles |= 0x10000;break;
default: //word.charAt(0) == 'S'
    ctrlClass = "scrollbar";textField = false;
    // SCROLLBAR, implies the flag SBS_HORZ (value 0).
}
System.out.print("class = "+ ctrlClass);

// Parse the line defining the control.
st.nextToken();
if(textField) {
    text = st.sval;st.nextToken();//skip ','
    st.nextToken();
    System.out.print(", text = "+text);
}
if(st.ttype == StreamTokenizer.TT_NUMBER) id = String.valueOf((int)st.nval);
else id = st.sval; System.out.print(", id = "+id);
st.nextToken(); //skip ','
st.nextToken();x = (short)st.nval; System.out.print(", x-coordinate = "+x);
st.nextToken(); //skip ','
st.nextToken(); y = (short)st.nval; System.out.print(", y-coordinate = "+y);
st.nextToken(); //skip ','
st.nextToken(); width = (short)st.nval; System.out.print(", width = "+width);
st.nextToken(); //skip ','
st.nextToken(); height = (short)st.nval; System.out.print(", height = "+height);
st.nextToken();
if (st.ttype == ',') st.nextToken(); //skip optional ','
int fvalue;
while(st.ttype != StreamTokenizer.TT_EOL && st.ttype != ','){
    fvalue = T.match(st.sval).getFlagValue();
    styles |= fvalue; // bitwise OR operation
    // corrective operation when CBS_DROPDOWN is specified
    //(without NOT CBS_SIMPLE as in Restorator)
    if(fvalue==2){
        if((styles & ~(~0 << 1)) == 1)
            // if the first bit is set to 1

```



```

// (that is, NOT CBS_SIMPLE has not yet been specified)
// the first bit is inversed (that is, set to 0)
styles = styles ^ 1;
    }
    st.nextToken();
}
System.out.print(", styles: "+styles);
if (st.ttype == ',') st.nextToken(); //skip optional ','
while(st.ttype != StreamTokenizer.TT_EOL && st.ttype != ','){
    extendedStyles |= T.match(st.sval).getFlagValue(); // bitwise OR operation
    st.nextToken();
}
System.out.println(", extended styles: "+extendedStyles);
// Go to the begining of the next line (optional [,helpId] skiped).
while(st.ttype != StreamTokenizer.TT_EOL)st.nextToken();
}
if(st.ttype == StreamTokenizer.TT_EOL) System.out.println("--- end of line ---");

// Generate into GrafiXML a control contained in this dialog box.
Graf.generateControl(id, ctrlClass, x, y, width, height, text, styles, extendedStyles);

    styles = 0; extendedStyles = 0;text = "";

// Go to the begining of the next line (eventual empty lines are skiped).
while(st.ttype == StreamTokenizer.TT_EOL)st.nextToken();
// BEGIN...END is sometimes used by Resource Builder instead of {...}
if (st.ttype == StreamTokenizer.TT_WORD && st.sval.equals("END")) st.ttype='';
}
}

System.out.println("current line : "+ r.getLineNumber());
System.out.println("number of line read : " + st.lineno());
}
}

}

/*****
// Resource.java Author : Julien Marion
// Represent a resource from the resource file which is proposed to the user in the 'import resources'dialog box.
// An instance stores the id of a resource, a type (dialog box or menu) and the location (line number) in the file.
// A resource object will be concerved if selected by the user, else will be destroyed.
*****/

package importresources;

public class Resource {
    private int line;
    private String id;
    private char type; // 'D' for DIALOG (or DIALOGEX), 'M' for MENU
    private boolean selected;

    public Resource (int l, String i, char t){
        line = l;
        id = i;
        type = t;
        selected = false;
    }

    public int getLine(){ return line; }

    public String getId(){ return id; }

    public char getType(){ return type; }

    public boolean isSelected(){ return selected; }

    public void select(){ selected = true; }

    public void unselect(){ selected = false; }

    // à retirer (pour tests)
    public String toString(){
        String t;
        if (type == 'D') t="dialog box"; else t="menu";
        return t + " at line " + line + ": " + id;
    }
}
}

```

```

//*****
// ResourcesSelection.java
// Define the window used to let the user to select the resources that will be processed and generated into GrafiXML.
//*****
package importresources;

import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.Vector;
import javax.swing.Box;
import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

public class ResourcesSelection extends JDialog implements ActionListener {

    private JLabel label;
    private JList list;
    private JButton ok, cancel;
    private JPanel panel0, panel1, panel2, panel3, panel4;
    private Vector v;

    //-----
    // Set up the GUI
    //-----
    public ResourcesSelection(Vector resources) {
        super((JFrame)null, "Import Resources", true);
        v = resources;
        setSize(410, 295);
        //setFont(new Font("SansSerif", 0, 8));
        setDefaultLookAndFeelDecorated(true);
        WindowListener wlistener = new WindowListener();
        addWindowListener(wlistener);
        Color gray = new Color(236, 233, 216);

        // Creates the components.

        label= new JLabel ("Select resources to import:");
        label.setAlignmentX(Component.CENTER_ALIGNMENT);
        //label.setDisplayedMnemonic('S');
        label.setBackground (gray);

        list = new JList(resources);
        list.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
        list.setLayoutOrientation(JList.VERTICAL);
        //list.setVisibleRowCount(10);
        JScrollPane listScroller = new JScrollPane(list);
        //listScroller.setPreferredSize(new Dimension(215, 375));

        ok = new JButton("    Ok    ");
        //ok.setMnemonic('O');
        cancel = new JButton(" Cancel ");
        //cancel.setMnemonic('C');
        ok.setBackground (gray);
        cancel.setBackground (gray);
        ok.setEnabled(false);
        cancel.setEnabled(true);

        // Organize the components.

        panel1 = new JPanel();
        panel1.setLayout(new BoxLayout(panel1, BoxLayout.Y_AXIS));
        panel1.add(Box.createRigidArea(new Dimension(0, 12)));
        panel1.add(label);
        panel1.add(Box.createRigidArea(new Dimension(0, 5)));
        panel1.add(listScroller);
        panel1.setBackground (gray);

        panel2 = new JPanel();
        panel2.setLayout(new BoxLayout(panel2, BoxLayout.X_AXIS));
        panel2.add(Box.createRigidArea(new Dimension(12, 0)));
        panel2.add(panel1);
        panel2.add(Box.createRigidArea(new Dimension(12, 0)));
        panel2.setBackground (gray);

```

```

panel3 = new JPanel();
panel3.setLayout(new BorderLayout(panel3, BorderLayout.X_AXIS));
panel3.add(ok);
panel3.add(Box.createRigidArea(new Dimension(5,0)));
panel3.add(cancel);
panel3.setBackground (gray);

panel4 = new JPanel();
panel4.setLayout(new BorderLayout(panel4, BorderLayout.Y_AXIS));
panel4.add(panel2);
panel4.add(Box.createRigidArea(new Dimension(0,5)));
panel4.add(panel3);
panel4.add(Box.createRigidArea(new Dimension(0,12)));
panel4.setBackground (gray);

setContentPane(panel4);

ok.addActionListener(this);
cancel.addActionListener(this);
list.addListSelectionListener(new ListListener());
}

//-----
// Handles the 'ok' and 'Cancel' buttons.
//-----
public void actionPerformed(ActionEvent e) {
    Object b = e.getSource();
    if (b==ok){
        Object[] s = list.getSelectedValues();
        Vector temp = new Vector();
        for (int i = 0; i<s.length; i++) temp.add(s[i]);
        v.removeAll(temp);
        this.dispose();
    }
    else {
        v.removeAllElements();
        this.dispose();}
}

//*****
// An inner class to detect list selection events.
//*****
private class ListListener implements ListSelectionListener {
    public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting() == false) {
            if (list.getSelectedIndex() == -1) {
                //No selection, disable the 'ok' button.
                ok.setEnabled(false);
            } else {
                //Selection, enable the 'ok' button.
                ok.setEnabled(true);
            }
        }
    }
}

//*****
// An inner class to detect if the dialog box is closed.
//*****
private class WindowListener extends WindowAdapter{

    public void windowClosing(WindowEvent e) {
        v.removeAllElements();
        dispose();
    }
}
}

```

```

//*****
// FlagTrie.java
// This class represent a trie used to identifying the styles and extended styles of a dialog box or a control when
// processing the .rc resource file.
//*****

package importresources;

import java.util.Vector;

public class FlagTrie {

    // Refer the root of the trie.
    private TNode root;
    // Store the depth of the last node returned by the match method.
    // The depth of a node v is the number of ancestors of v, excluding v itself.
    // Note that I could use a simple recursive algorithm to know the depth of a node v :
    //   if isRoot(v) then return 0
    //   else return 1 + depth(parent(v))
    // But I must then add a new attribute to each node storing the reference to its parent.
    private short depth;

    //-----
    // Set up the trie with flags relevant to my transformation. It is considered as a constant (cannot be modified once
    // created).
    //-----

    public FlagTrie(){
        root = new TNode();

        // Add the flag used in my correspondance which are default in the shortcut notation:
        // Button styles, that correspond to the 8 types of buttons
        addFlag("BS_DEFPUSHBUTTON", 0x1);addFlag("BS_CHECKBOX", 0x2);
        addFlag("BS_AUTOCHECKBOX", 0x3);addFlag("BS_RADIOBUTTON", 0x4);
        addFlag("BS_AUTORADIOBUTTON", 0x9);addFlag("BS_GROUPBOX", 0x7);
        // This last button style is still inserted (even if its value is 0)
        // because it has a common prefix with BS_PUSHLIKE
        addFlag("BS_PUSHBUTTON", 0x0);
        // Static styles (note that value of SS_LEFT is 0x0)
        addFlag("SS_RIGHT", 0x2);addFlag("SS_CENTER", 0x1);addFlag("SS_ICON", 0x3);
        // Edit style
        // value of ES_LEFT is 0x0
        // List box style
        addFlag("LBS_NOTIFY", 0x1);
        // Combobox style
        addFlag("CBS_SIMPLE", 0x1);
        // Scroll bar style
        // value of SBS_HORZ is 0x0
        // By default in the default notation (for each types)
        addFlag("WS_VISIBLE", 0x10000000);addFlag("WS_CHILD", 0x40000000);
        addFlag("WS_CHILDWINDOW", 0x40000000);
        // By default in the default notation (for some types)
        addFlag("WS_BORDER", 0x800000);addFlag("WS_TAPSTOP", 0x10000);
        addFlag("WS_GROUP", 0x20000);addFlag("WS_VSCROLL", 0x20000);

        // Add the remaining flag used in my correspondance rules :
        addFlag("CBS_DROPDOWN", 0x2);addFlag("CBS_DROPDOWNLIST", 0x3);
        addFlag("DS_SETFONT", 0x40);addFlag("DS_SHELLFONT", 0x48);
        addFlag("WS_CAPTION", 0xC00000);addFlag("WS_THIRCKFRAME", 0x40000);
        addFlag("WS_SIZEBOX", 0x40000);addFlag("WS_DISABLED", 0x8000000);
        addFlag("WS_EX_TOPMOST", 0x8);addFlag("DS_SYSDIALOG", 0x8);
        addFlag("BS_LEFT", 0x100);addFlag("BS_CENTER", 0x300);
        addFlag("BS_RIGHT", 0x200);addFlag("BS_PUSHLIKE", 0x1000);
        addFlag("BS_BITMAP", 0x80);addFlag("BS_ICON", 0x40);
        addFlag("BS_OWNERDRAW", 0xB);addFlag("BS_USERBUTTON", 0xB);
        addFlag("LBS_EXTENDESEL", 0x800);addFlag("ES_RIGHT", 0x2);
        addFlag("ES_CENTER", 0x1);addFlag("ES_MULTILINE", 0x4);
        addFlag("ES_AUTOHSCROLL", 0x80);addFlag("ES_PASSWORD", 0x20);
        addFlag("ES_NUMBER", 0x2000);addFlag("WS_HSCROLL", 0x100000);
        addFlag("ES_AUTOVSCROLL", 0x40);addFlag("PBS_MARQUEE", 0x8);
        addFlag("PBS_VERTICAL", 0x8);addFlag("SS_ENHMETAFILE", 0xF);
        addFlag("SS_OWNERDRAW", 0xD);addFlag("SS_USERBUTTON", 0xD);
        addFlag("SS_SIMPLE", 0xB);addFlag("SS_LEFTNOWORDWRAP", 0xC);
        addFlag("SS_NOPREFIX", 0x80);addFlag("SS_BITMAP", 0xB);
        addFlag("SS_REALSIZEIMAGE", 0x800);addFlag("SS_SUNKEN", 0x1000);
        addFlag("SS_BLACKFRAME", 0x7);addFlag("SS_GRAYFRAME", 0x8);
        addFlag("SS_WHITEFRAME", 0x6);addFlag("SS_BLACKRECT", 0x4);
        addFlag("SS_GRAYRECT", 0x5);addFlag("SS_WHITERECT", 0x9);
        addFlag("SS_ETCHEDHORZ", 0x10);addFlag("SS_ETCHEDVERT", 0x11);
        addFlag("SS_ETCHEDFRAME", 0x12);addFlag("TBS_VERT", 0x2);
        addFlag("UDS_AUTOBUDDY", 0x10);addFlag("DTS_TIMEFORMAT", 0x9);
        addFlag("WS_PALETTEWINDOW", 0x108);addFlag("LBS_STANDART", 0x800003);
        addFlag("WS_POPUPWINDOW", 0x80880000);

        // Add flags not in my correspondance rules but that have a common prefix with one of the flag above.
        addFlag("CBS_SORT", 0x100);addFlag("DS_SETFOREGROUND", 0x200);
        addFlag("BS_RIGHTBUTTON", 0x20);

        // Compress the trie.
    }
}

```

```

        compress(root);
    }

    //-----
    // Return the node storing the value of flag if there is a match with one of the strings of the trie.
    //-----
    public TNode match(String flag) {
        TNode v = root;
        if (flag != null) {
            depth = 0; // Depth of the node we terminate tracing the path
            boolean success = true;
            while (success && depth < flag.length()) {
                success = false;
                Vector children = v.getChildren();
                for (int i = 0; i < children.size() && !success; i++) {
                    TNode w = (TNode) children.elementAt(i);
                    if (w.element() == flag.charAt(depth)) {
                        v = w;
                        success = true;
                        depth++;
                    }
                }
            }
        }
        return v;
    }

    //-----
    // Add a new style flag identifier to the trie with a specified numeric value
    //-----
    private void addFlag (String flag, int value) {
        //depth = 0;
        TNode v = match(flag);

        if (depth < flag.length()) {
            // The search stop at an internal or external node and it remains characters of flag to store in
            // the trie. Add a chain of node after v corresponding to the characters from flag[depth] through
            // flag[length-1]
            TNode w = expand(v, flag.substring(depth));
            // Store the value in the last node of the chain.
            w.setFlagValue(value);
        }
        else
            // depth == flag.length()
            // The search stop at an internal node (since flag is not already represented in the trie).
            // Since the trie is not yet compressed, only leaves have a positive flag value.
            v.setFlagValue(value); // Add the value in v.
    }

    //-----
    // If v is an external Node, transform v into an internal node by creating a new chain with the characters of s
    // starting from v. If v is an internal node, create a new chain with the characters of s at the good place starting
    // from v.
    // Return the last node composing the new chain. The string s is composed by at least one character (s.length > 0).
    //-----
    private TNode expand (TNode v, String s) {
        TNode w = null;
        if (isExternal(v))
            for (int i = 0; i < s.length(); i++) {
                w = new TNode(s.charAt(i));
                v.getChildren().addElement(w);
                v = w;
            }
        else {
            // v is an internal node and its children are not labelled with the first character of s.
            Vector children = v.getChildren();
            int i = 0;
            while (i != children.size() && ((TNode) children.elementAt(i)).element() < s.charAt(0) i++);
            TNode u = new TNode(s.charAt(0));
            children.insertElementAt(u, i);
            if (s.length() > 1) w = expand(u, s.substring(1));
            else w = u;
        }
        return w;
    }

    //-----
    // Compress the subtree rooted at v
    //-----
    private void compress (TNode u) {
        Vector p = u.getChildren();
        for (int i = 0; i < p.size(); i++) {
            TNode v = (TNode) p.elementAt(i);
            if (isInternal(v)) {
                Vector q = v.getChildren();
                TNode w = v;
                while (isInternal(w) && q.size() == 1 && w.getFlagValue() == 0) {
                    w = (TNode) q.elementAt(0);
                }
            }
        }
    }
}

```

```

        q = w.getChildren();
    }
    if (isExternal(w)){
        //w is a leaf, and the chain of nodes after v consists of redundant nodes
        v.setFlagValue(w.getFlagValue());
        v.suppressChildren();
    }
    else
        // q.size()>1 || w.getFlagValue() !=0
        compress(w);
    }
}

//-----
// Test whether node v is a leaf.
//-----
private boolean isExternal (TNode v){ return v.getChildren().isEmpty(); }

//-----
// Test whether node v is an internal node.
//-----
private boolean isInternal (TNode v){ return !v.getChildren().isEmpty(); }

//-----
// Override the toString method to produce a parenthetic string representation of the created trie (for verification)
//-----
public String toString(){return print(root);}

//-----
// Perform a printing of the elements in the subtree rooted at node v of this trie.
//-----
private String print(TNode v){
    String s = v.toString();
    if (isInternal(v)){
        Vector children = v.getChildren();
        // Open parenthesis and recursively print the first subtree
        char open, close;
        if(children.size()>1) {open = '['; close = ']';} else {open = '('; close = ')';}
        s += open + print((TNode)children.elementAt(0));
        for (int i=1; i<children.size(); i++) // size > 0 because v is internal.
            // Recursively print the remaining subtrees
            s += ", " + print((TNode)children.elementAt(i));
        s += close;
    }
    return s;
}
}

//*****
// TNode.java
// Represent a node of the trie labelled with a character.
//*****
// Author : Julien Marion

package importresources;

import java.util.Vector;

public class TNode {

    private int flagValue = 0; // Numeric value of a flag if the node is associated with a string of the trie, else 0.
    private char label; // Each node is labelled with a character from {A;Z}U{3,_}.
    private Vector children = new Vector(1);
    // Default constructor (used for the root which store no character).
    public TNode () {label = '#';}
    // Constructor with parameter.
    public TNode (char c){label = c;}
    // Set a numeric flag value to the node.
    public int getFlagValue () {return flagValue;}
    // Set a numeric flag value to the node.
    public void setFlagValue (int v){flagValue = v;}
    // Return the label at this position in the tree.
    public char element () {return label;}
    // Return a vector containing the children of the node.
    public Vector getChildren(){return children;}
    // Suppress all the children of the node.
    public void suppressChildren(){children = new Vector(1);}
    // For verification.
    public String toString(){return ""+label+" "+flagValue;}
}

```

```

//*****
// Grafi.java Author : Julien Marion
// This class is used to implement my transformation rules and contain the interface with the code of GrafiXML.
//*****

package importresources;
// Import declaration of package be.ac.ucl.isys.grafixml.gui.editor.component be will be defined here.
// See http://www.usixml.org/javadocs/grafixml/.

public class Grafi {

    public static String fontName;
    public static byte fontSize;
    public static boolean bold, italic;

    //-----
    // Generate a dialog box into GrafiXML.
    //-----
    public static void generateDialog(String id,short width,short height,String text,int styles,int extendedStyles){
        System.out.println("Dialog box "+id+" generated into GrafiXML");
        System.out.println();
        // TO DO
        // change also the type void (return reference of the container)
    }

    //-----
    // Generate a graphical component (or eventually a graphical container) corresponding to information given in
    // parameters into GrafiXML.
    //-----
    public static void generateControl(String id,String ctrlClass,short x, short y,short width,short height, String text,
                                     int styles,int extendedStyles){

        System.out.println("Control "+id+" generated into GrafiXML");
        System.out.println();
        // TO DO
    }

    //-----
    // Generate a menu into GrafiXML.
    //-----
    public static void generateMenu(){
        // TO DO
    }

    //-----
    // Return the n bits starting at the position p in x (0?p? 31 and n? 32-p).
    // << shift bits left, filling in with zeros
    // >>> shift bits right, filling in with zeros
    // & bitwise AND
    // ~ bitwise complement (prefix unary operator)
    //-----
    private int readBits(int x, short p, short n){
        return (x >>> p+1-n) & ~(~0 << n);
    }

    //-----
    // Read the bit at the position p in x (0<=p<=31). Return true if the bit is 1, else false.
    //-----
    private boolean readBit(int x, short p){
        return ((x >> p) & ~(~0 << 1) ) == 1;
    }
}

//*****
// For future work.
//*****
package importresources;

import java.io.File;

public class SdfFile {
    public static void process(File f){
    }
}

//*****
// For future work.
//*****
package importresources;

import java.io.File;

public class NibFile {
    public static void process(File f){
    }
}

```

