Object Library 1.1

Beta

Copyright Exile Software 1995
Compuserve 100267,546
scling@hk.super.net

Installation

Create a directory and copy the following files to it:

- objlib.exe
- objlib.hlp
- objlib.ldb
- objlib.mdb
- objlib.vbx
- mlist.vbx - move to system directory if you already have an earlier version
- objlib.ini

The library also requires the VB 3.0 database files.

Overview

Consult the help file topic 'Overview' for a quick introduction to using the library.

Object Library is a code database which allows you to store and reuse objects implemented in native VB code. Object Library provides a means of storing VB source code in a format which closely resembles the Visual Basic 3.0 design environment. Using the library, you may design Objects in Visual Basic which can be copied out of the project in which they are created, stored in the database, and inserted back into future projects with little or no rewriting required. While much the same thing can be accomplished by adding Forms and Modules to a project and copying and pasting code and controls as needed, this involves a considerable amount of hassle, is error-prone, and is seldom less work than rewriting unless the original code was specifically designed to be reused in this way. Object Library is intended to assist in the design and storage of code.

Any manner of code may be stored in the library database, from simple routines to entire forms or modules, but the library is specifically designed to allow you to store groups of related procedures and controls which together function as components or 'Objects'. A VB 'Object', for the purposes of this discussion, is a collection of VB controls and native VB source code which together comprises a unit of functionality. The library provides a number of features to help you define, store, and reuse VB objects. To use the Library most effectively, you must plan for reuse by designing modular code units employing techniques of abstraction and encapsulation when you write code. These code units may then be easily imported into the Library, stored for reuse, and later exported to other VB projects.

The Library interface emulates the VB design environment; viewing and editing is accomplished by the same kinds of operations you use in VB's design mode. A main window contains the menu and toolbar. A list of Items in the Library is displayed in a seperate 'Contents' window. The Fields of each item in the Library are displayed using code windows similar to VB code windows.

The Library is organized into Groups of Objects, corresponding to Tables of Records in the Library database. A Group is a Table used to store related Object records. You may create as many Groups as you need to. Each Group may contain many individual Object records: each item in the Library is a single Record in one of the Group tables. A Record is refered to in the documentation as an 'Object' or 'Object record'. Since an Object record needs to store controls, control procedures, and general procedures, it encompasses the features of both Forms and

Modules: it might be likened to a Form/Module pair.

An Object record can have up to 6 Fields:

- Name - the name of the Object record
- Notes - a brief desciption
- Form - binary control data
- FormCode - VB source code for Forms
- Module - VB source code for Modules
- Class - extra field for VB source code, text, text control data, comments, etc.

The Name and Notes fields are shown in the Contents window where you select an item to view. The other fields are displayed in seperate code windows. All the code windows are alike in functionality except the Form window. The Form window displays a graphic representation of controls, the other windows display text.

Using the Library

To use the library, simply paste text or controls to one of its code windows just as you would to a Form or Module in the VB design environment.. The help file explains how to create new Group tables and Object records and describes specific editing commands. Several samples are provided with the Library to illustrate both storage and the concept of a VB 'Object'.

The Library imposes no requirements on code; you can store any kind of VB code in the Library in any way you choose. At the simplest level, a code library is a means of storing code techniques, tips, solutions, etc. An object, on the other hand, must act as a cohesive unit. If you want to make the task of importing complex blocks of code into the Library easier, you should design your code to be modular. Such code will be easier to reuse if you also keep it in a single Object record, placing general code in the Module field and Form-level code in the FormCode field.

You can only store one set of controls in each Object record, but if you have an Object which uses two or more sets of controls, you can create extra records for them.

For smaller blocks of code such as techniques which involve only one or two procedures, you can put many such items together in a single record.

Very large and complex blocks of code which involve several Forms and many procedures may warrant their own Group table. This would allow you place some of the procedures into seperate Module fields, or allow you to store code for each Form seperately, without worrying about getting confused about which records go together.

Other Features

The Library allows you to open VB source code files in code windows. If the file is a VB '.FRM' file, you may view the controls graphically or as text. You may copy code from these files to the clipboard. You can also open bitmaps and icons in these windows. Drag a file from File Manager and drop it on the Main window to open it.

The Library provides two wizards to make the task of importing code into the Library and exporting code to a VB project a little easier. The wizards allow you to select multiple items and transfer them in a single step or series of steps. Also provided is facility for renaming procedures and controls before exporting them to a VB project where naming conflicts may exist.

The Library includes a Project Browser which allows you to open a VB project and get a quick overview of the project's structure. Using the Browser, you can see lists of symbols and procedures in the project and view their source code with a button click. You can also quickly locate all occurences of a symbol and all references to a variable, control, or procedure. Form files may be viewed graphically as well.

The Library includes a Screen Manager for controlling the placement of windows in the VB design environment.

About VB Objects

First, the term 'VB Object' as used herein refers to a collection of controls and native VB code which together function as a unit and which are suitable for reuse in future projects. No other properties or qualities are implied. A VB Object in this sense is unrelated to the Objects exposed by Visual Basic or OLE servers.

An object implemented in native VB code is not like a class; it is more like a VBX or OLE object. A VB Object is something more than a technique or a subroutine, but it isn't a formally defined structure. Isolating and defining an object typically means identifying a collection of controls, property settings, event code and subroutines which are necessary for the proper operation of a block of code. Some essential components of a bit of functionality will almost always be scattered among click, repaint, and event procedures and will depend upon property settings of controls to operate properly. The 'boundaries' of this collection cannot be formally specified; the essential feature of a VB Object is that it functions as a unit, because this is what saves you from having to edit (or rewrite) it.

The nature of a VB object is more or less prescribed by the language; you can't really create any formal classes or objects with VB 3.0, but you can use the Set statement in conjunction with the objects exposed by VB. What you can do by way of object-oriented programming this way is frankly not that much; emulations of OOP techniques are often confounded by peculiarities of VB and the Set statement is full of bugs. But some techniques, such as encapsulated modules, state-machines, psuedo-methods, and even message-based emulations of classes have been demonstrated. Another possibility involves the creation of interface elements, discussed below. OO-influenced techniques can also be helpful or even necessary when projects grow in size. These techniques work, so it is worthwhile to do whatever can be done.

Interface elements are particularly suitable for an Object implementation because they are easy, they are required by almost every project, and they reduce your reliance on external modules. Almost any custom control which involves graphic output can be implemented in VB code using the same API calls and in most cases will be indistinguishable in performance. To create this kind of object, find examples written in C and use these as a base. Expect some adaption to be required. Avoid the temptation to use VB functions when doing screen output unless you are sure they are comparable in speed. Most poorly performing VB code is bad because it unintentionally executes code repeatedly or otherwise does unnnecessary things, so check the path of execution through paint and resize routines and read the optimization tips on the MSDN CD. When you create code which implements an interface object, place all the code which manipulates data in a single Module, and place the object's data in a structure in the Form where the object is displayed. This allows you to create multiple instances of the object in a project, or to add the object to new projects by just adding the module, a couple of controls, and a handful of simple function calls. This model of a native VB Object is similar to the MDI sample and is the model the library is perhaps most useful for.

More information on using the library

You can use the library as a VB source file viewer without opening the database by either launching it with the ' - ' (minus sign) switch or by closing the library. If this package includes OBJLIB.VBX, the Main window will allow you to open a file by dropping it on the window. Dropping a makfile will open the Project Browser. (The VBX provides provides drag and drop functionality only; you may reuse it but not redistribute it seperately.)

The right mouse button will perform certain shortcut actions in some circumstances. Lists with checkboxes can be selected or deselected in their entirety. Lists of procedures allow viewing the selected procedure by right-clicking the list.

Notice

This software is shareware. You are free to evaluate it for as long as you see fit, provided you consent to the following terms:
When you feel you have arrived at some conclusion regarding its suitablity, you should register the package in either one of two ways:
1. Pass along any criticisms, suggestions for improvements and enhancements, or descriptions of how it is useful (or useless) which you may arrive at. Send E-mail to the addresses listed at the top of this document.
2. Fee not applicable for this version. See #1.

I will upload fixes as I am made aware of problems, but I accept no responsibility for its use. If you encounter problems, please contact me, and I will try to help.

Limitations

Object Library is written entirely in Visual Basic so there are a couple of things to keep in mind when using it.

Textboxes are a big problem; the code windows use ordinary Windows textboxes and their capacity can vary. If a single procedure is greater than 32K, consider breaking it up into several smaller procedures for storage. The most likely problem to occur is the inability of the library to display text descriptions of controls, since these regularly exceed a textbox's capacity. I am looking for a replacement textbox, but for this version the Library will disable some actions or will try to warn you if are attempting something which might be a problem.

Strings are limited to 64K, but since VB procedures are limited to 64K, I haven't seen the need to go outside VB in most cases. Control text sometimes exceeds 64K; I hope I have trapped the crashes but I haven't provided a workaround. Avoid transfering large blocks of text through the clipboard, use the wizards instead.

The Screen Manager may produce undesirable side effects under certain circumstances, such as invoking it while a messagebox is displayed or when some utilities are running which cannot be moved or resized. Discontinue use of it if and notify me if you have problems.

Controls are drawn not as VB does but in a kind of X-Ray view so that as many as possible may be seen. Sometimes a control rendered opaque will obscure underlying controls. Bitmaps are not rendered. Each control is labelled with either its name or caption. Third party VBX's may not be rendered properly. Menus are not rendered. Note: The controls are misaligned in W4W on my system, but not in NT and I can't find a problem in the code.

Forms and Menus cannot be copied to the Clipboard and therefore cannot be stored in the library as binary control data. The text description of these items can be stored in any text field.

Toggling between graphic and text views of controls is not fully implemented in this version but future versions will allow it if the facility is not already part of VB 4.0.

The Rem statement and line numbers are not supported. Certain other legal but unusual coding styles may not be recognized properly. If you name Form files with the extension '.BAS' or Module files with the extension '.FRM' some functions will misrecognize their type. If you paste scrambled code you will produce parsing errors; this is true of VB as well, but the errors may not handled in the same way.

This program was written and tested on NT 3.5. It seems to behave much the same on W4W and W95.

This is the first full version, so expect a few bugs, and please report them! I intend to rewrite this in Pascal and any suggestions for needed changes or improvements would be appreciated. The Project Browser and wizards will be redesigned so any suggestions are welcome.

More information on OOP in VB

Download the following article for more information on emulating classes in VB:
'Simulating Object Oriented Programming in Visual Basic', Compuserve text posted by Pete Washburn (73750,3141) January 1994

Visual Basic Programmer's Journal and their Compuserve forum contain articles and discussions of coding techniques for state machines and OOP.

Windows Tech Journal and OOP Alley in CLMFORUM may be of interest.

Examples included with the library

Step-by-Step instructions for Tabs Object

The Group 'Dialogs' includes one Object record, named 'Tabs'. The Tabs Object creates a tabbed dialog.

1. Create a new project in VB.
2. Select Dialogs in the Contents window of the library.
3. Select **Object | Export Wizard** from the library Main menu.
4. Click **Next** twice to skip to the last step. (For a new project, we don't need to alias the controls or check for conflicts.)
5. The Object will be copied in 3 steps, one for each Field. Select **Controls** in the Field combobox. Click **Copy Field**. The controls are copied to the clipboard.
6. Click Form1 in the new VB project to make it active. Now select **Edit | Paste** from the VB menu to paste the controls on the Form.
7. Select **Form Procedures** in the Export Wizard Field combobox. Again, click **Copy Field** to copy all the Form procedures to the clipboard.
8. Open the code window for Form1 in the VB project. Select **Edit | Paste** form the VB menu.
9. Stop for a moment and check the VB code window's Form_Load procedure. The declarations may need to be moved and the extra headers deleted. (Sorry, I'm working on this.)
10. Select **Module Procedures** in the Export Wizard Field combobox. Again, click **Copy Field** to copy all the Module procedures to the clipboard.
11. Create a new Module in VB. Select **Edit | Paste** to paste the code to the Module.
That's it. If no aliasing is needed, an Object can be copied in 3 steps.
12. Now attempt to run the project. Probably some errors will occur if the declarations in the Module are not in the correct order. Fix any other problems now before proceeding with development on the project to be sure you have correctly pasted all the Object's code and controls.
13. If all is well, a simple tabbed dialog should appear. This sample is for demonstration purposes; it illustrates how to organize code for easy reuse.

Step-by-Step instructions for Importing the ChangeIcon Dialog

That was easy enough, so let's try something a little more difficult. Run Visual Basic and open the Sample1 project which accompanies this software. Run the project. A Form with an icon is displayed. Click **Select** to open the ChangeIcon dialog. This dialog was modeled on Program Manager's; we are going to import it into the library.

1. Select **Dialogs** in the Contents window of the library.
2. Create a new Object record by clicking the page icon or selecting **Object | New Object**. Give the new record a name like 'ChangeIcon Dialog'.
3. Select **Object | Import** Wizard. If it is already open, click **Refresh** to point it to the new library Object record.
4. Click **Next** to go to the second step, 'Select the Source Files'. Click **Browse** and find the files for sample1 which came in this package. Select SAMPLE1.MAK.
5. Click **Next** to go to the third step, 'Select the Procedures'. The Object we are importing is the ChngIcon form, so select CHNGICON.FRM from the combobox. The procedures contained in the file are now shown in the top listbox.
6. Take a moment to browse through the file. Select a procedure and then right-click it to view the procedure; click the textbox to dismiss it.
7. You must now decide which procedures to import, and which library field to import then to. In the case of a dialog form, you normally want all the procedures, so select all the procedures in the top listbox. Since this is Form code we should add it to the Form field: check to see that the Form list is the active list (the active list will have a white background). Click it to make it active. Now click **Add**. (You may select procedures one by one by doubleclicking the procedure.) The procedures should now be listed in the Form list below.
8. We also need a routine from Module1, so select MODULE1.BAS from the combobox. The Module's 3 procedures should now be shown in the upper listbox. Click the Module list to make it active. We need the function 'GetFile' , so select it and doubleclick it. The function should now be listed in the Module list below.
9. Click **Next** to go to the fourth step, 'Select the Declarations'. Make sure Module1 is still selected in the combobox and that the Module list is still active. Now select any declarations the dialog will require. I happen to know that the Object needs all these declarations, so select them all and click **Add**.
10. Click **Next** to go to the final step, 'Import the Code'. The selections you have made are listed in the comboboxes and can be previewed by selecting them. To import the code, click **Import**. After a moment a messagebox should inform you that the code was successfully imported.
11. Close the Import wizard and open the ChngIcon Form in the VB design environment. Resize the Form until you can see the two little pictureboxes. Now select all the items on the Form by clicking in the upper left corner and dragging a box which encompasses all the controls on the Form. Click **Edit | Copy** from the VB menu.
12. Now paste the controls into the library. Select **Edit | Paste Controls**. (You may have to open the library's Form window to get the menu to update and enable the Paste Controls selection.) The Form window should now show the controls you pasted.
13. That's it! If you open the FormCode window, you will notice that the event procedures for the controls you just pasted are still listed in the (general) (declarations) section. Reselect the record in the Contents window. This causes the item to be saved and reread from the database. The parser should now recognize the routines as control routines and display them appropriately.

Note: The property settings for the ChngIcon form were not copied to the library because Forms can't be copied to the clipboard. In most cases this won't matter. However, in the case of a dialog form or a form with menus, the form's data may also be needed. This version does not allow you to import controls as text using the Import wizard, but you can use the extra Text field to store text descriptions of controls. Instead of pasting controls as described above, open the file in a text editor and copy the controls to clipboard, then paste them into the library as text.

Step-by-Step instructions for Importing the OwnerDraw Listbox Object

The following example illustrates the typical usage of the library. We are going to extract an Object from a project and store it for reuse, then reuse it in a new project.
Run VB and open the Sample2 project. Sample2 is a simple program launcher (everyone seems

to love building their own program launcher). Run the project. A simple menu is displayed. Click **Groups** and select a group. A listbox is displayed containing the items from a Program Manager group. (By the way, the slow speed is caused by the DDE link to Progman, not by the listbox.) Click an item to launch it. OK, enough fun.

We'll now import the code for the listbox. I should mention that extensive modifications of the source code were required to free it of dependencies on other elements, since the listbox Object did not exist as such in the original project. The code for the listbox was reorganized into a seperate module and rewritten where necessary. Unless you are a very disciplined coder, this will be a necessary prelude to storing an Object in the library.

1. Create a new Group in the library called 'Listboxes' by selecting **Object | New Group**. Accept the default fields.
2. When the Group table is created, a new record is added called 'Untitled'. Change the name to 'Ownerdraw Listbox'.
3. Select **Object | Import wizard**. Make sure the new Object is shown as the import target.
4. Click **Next** to go to step 2 and select the Sample2 makefile.
5. Click **Next** to go to step 3. Select LISTWIN.FRM in the combobox. Select all the Form's procedures and click **Add**. Be sure the procedures are added to the Form field.
6. Now select LISTWIN.BAS in the combobox. Select all the Modules procedures and click **Add** to add them to the Module field.
7. Click **Next** to go to step 4. Select LISTWIN.BAS in the combobox if it is not still selected. Select all the declarations and click **Add** to add them to the Module field.
8. Select LISTWIN.FRM in the combobox and add all the Form's declarations to the Form field.
9. Click **Next** to go to the final step. Click **Import**. The Object's code is now in the library.
10. Open the LISTWIN.FRM in the VB design environment and copy all its controls to the clipboard. (The Form has only two controls, a picturebox and a scrollbar.)
11. Paste the controls into the library.

It's a good idea to test whether you imported everything you need, because even a simple Object such as this one will likely be missing a few things. Create a new VB project and add a Module. Select **Object | Export** Wizard from the library menu. Click to the third step and copy each of the 3 fields to the new VB project's Form1 and Module1. Run the project. OK, we missed some things.

In fact, what we have in the library is not yet an Object. The idea here is that after extracting code from a project with the Import wizard, you can paste it into a new VB project and refine it. If you comb thru this code, you'll find numerous project-specific references and operations which must be altered to convert the code into a truly modular Object. (Rather than specifying all the steps to fix it, you can find the finished code in the Sample3 directory.) After the code has been modified to function independently, it can be copied back into the library, ready to be reused in future projects.

Several other examples may be included with the library. The purpose of all these examples is not to demonstrate coding techniques or styles, but to illustrate methods of structuring and organizing code which you may find helpful.