A MENTOR FOR VISUAL BASIC-BASED COMPONENT DEVELOPMENT

For experienced COM/COM developers, the benefits of Visible Designer are significant. For newcomers to components, they are simply invaluable. At the price, large firms should have no trouble justifying the cost to test Visible Developer on an upcoming project. For small consultancies and lone developers the case is even more compelling. Visible Developer, in my opinion, has a very solid future.

Object-oriented programming has been with us for thirty years now, long enough that we can speak of “traditional” Object Oriented development. The move to component-based development is a genuine paradigm shift: unsettling, radically different, and full of promise. That many developers are only beginning to grasp this new way of thinking is really not the problem. The real problem is that the tools to design software components are only beginning to appear. Development tools are just beginning to change, or to put it less charitably, they’re woefully behind our current ambitions.

Enter Visible Developer, from Visible Systems, an object-relational mapping tool that reads a database schema and generates business objects. It’s almost completely automatic, and it guides the developer through the difficult process of dividing an application into distinct layers that communicate with each other. These layers become ActiveX objects (DLLs or EXEs) and can reside virtually anywhere. Visible Developer also makes smart decisions about which chunks of an application belong in which layer – and then generates all the code required for the layered objects to communicate with each other.

For this review, I tested Version 1.1 and concluded that Visible Developer will pay for itself in less than a week. A Visual Basic add-in, Visible Developer is only visible while Visual Basic is running on the add-in menu. The first step in creating a new project is to identify the database on which to base the project. Next, you select the tables you want to turn into business objects. Visible Developer then reads the database and creates a tree-view with a list of business objects and their properties, methods and relationships.

Each table column becomes a property of the related object. At this point you can customize the properties, supplying text labels and so on. You can add methods to each object, if you wish, and also specify the nature of the relationships among the tables. Next you decide how to structure the output – into one, two or three layers. Finally, you click build and then sit back and watch Visible Developer generate your code.

201 Spring Street • Lexington, MA 02421
(800) 6VISIBLE • Fa x (781) 778-0208
Email: sales@visible.com • www.visible.com

Visible SYSTEMS CORPORATION

BENEFITS

Depending on the options you select, Visible Developer can generate a single executable, a 2-tier or a 3-tier application. The first choice places all the objects in a single, monolithic project – a good first step for newcomers. The other choices divide objects into layers according to the following scheme:

- **User Interface Layer –** contains all the forms that participate in the project. Objects in this layer know nothing about business rules or data access or database schemas. All they do is communicate to the user, forward requests to the second layer, and return the results of those requests to the UI.
- **Logical Business Object Layer –** contains the business objects which know all about business rules and data validation and know nothing at all about either the UI or the data access mechanisms beneath them in the layer three.
- **Physical Business Object Layer –** contains the persistence objects that read from, and write to the database. The objects in this layer know all about the physical storage underlying the application including: column and table names, relationships among tables and referential integrity constraints.

The valuable stuff is in layers two and three – the user interface layer is mere proof of concept. The forms are useful starting points, but in the average application none of them will survive intact. Visible Developer makes no pretense of anything more. It's not an application generator - it focuses on the hard stuff.

During my testing, I pushed Visible Developer to limits it may never have seen, and found its response thoroughly solid. Our application's database involves more than 400 tables in a SQL Server 6.5 database. I ran Visible Developer against the database and successfully generated over 1600 business objects based on it. (Not surprisingly, it took a while.) Visible Developer successfully extracted all the table, index and relationship information from the database, and created the appropriate business objects in each category – over a million lines of code, not counting white space. I distributed all these objects in a series of projects corresponding to the modules that comprise our existing monolithic application.

Application Packaging

Visible Developer packages the resulting code in one, two or three Visual Basic projects:

- **Single Project –** Places all generated code in a single Visual Basic project that is a standard executable.
- **Two Projects –** Places generated Forms in a standard executable; and logical and physical Business Object classes in an ActiveX DLL.

- **Three Projects** – Places Forms in a standard executable. Logical Business Object classes are put in an ActiveX DLL; and physical Business Object classes in an ActiveX DLL.

Containers
Another significant benefit is Visible Developer's recognition of contained objects. A contained object is based on an identifying relationship, for example the relationship between "Orders" and "Order Items." The latter have no meaning without the former – they are identified by their belonging to a particular order and no other. By default, Visible Developer identifies one-to-many relationships as reference, and does not create contained objects – but the developer can use the Relationship Builder form in Visible Developer to change a reference relationship to an identifying relationship. This change results in the creation of a collection variable within the parent object, that houses the instances of the contained object – or, in relational terms, the many rows in the child table.

Finally, a major benefit of Visible Developer is what it calls "edit points" – clearly marked points in the code where you might want to place customizations or embellishments. The comments even suggest what sort of code should go there! This becomes significant on regeneration of your objects. Any custom code added at the edit points can be preserved and incorporated in subsequent generations. This feature allows developers to work iteratively, refining their objects little by little and regenerating the code several times in the course of a project.

Conclusions
For Experienced COM/COM developers, the benefits of Visible Developer are significant. For newcomers to components, they are simply invaluable. At the price, large firms should have no trouble justifying the cost, just to test Visible Developer on an upcoming project. For small consultancies and lone developers, the case is even more compelling. Visible Developer, in my opinion, has a very solid future.

By Arthur Fuller, Computing Canada

The Visible Developer was formerly called VBMentor. 3tSoftware was acquired by Visible Systems Corp.

201 Spring Street • Lexington, MA 02421
(800) 6VISIBLE • Fa x (781) 778-0208
Email: sales@visible.com • www.visible.com

Visible SYSTEMS CORPORATION