A Glimpse into the Object-M Future

by Fred Boles

Abstract:

The software development world around us has begun to evolve into a world of interacting objects. If we wish to be able to participate in this new world, we must begin to take the new paradigm seriously. This article will explore one possible view of the role of M in the world of interacting objects.

Introduction

All signs indicate that the future holds Object Technology for the M programmer. From the M standards body, to the lone major M implementer to M tool vendors, we M developers are overwhelmed by the presence of Object Technology. Given the constant reminders that Objects are the technology of choice for the future of M, it is time to take a serious look at how we can expect this to affect each of us.

The Object Landscape

The M standards body (MDC) has launched the OMEGA project to introduce Object Oriented Development to M. This project, however, is still on the drawing board with very little to show. InterSystems, now the lone major implementer of M Technology, has introduced Caché ObjectScript. While maintaining compatibility with the current M language standard, Caché ObjectScript provides Object Oriented extensions to the language. Although several beta releases of Caché ObjectScript have been made available during the past year, InterSystems has yet to ship a production release. EsiObjects™, from ESI Technologies, enjoys a large advantage in terms of product longevity. This product has had several years to mature and, with the release of version 3.0 just around the corner, brings many new and exciting features to the Object-M developer.

Although no Object-M environment has yet received broad-based acceptance, EsiObjects is the only currently available production Object-M environment. This article will explore the features found in the current product release as a basis for what we might expect from an Object-Oriented M world. With the many new features EsiObjects brings to the Object-M developer, there are a few stand-outs that will be discussed here.

EsiObjects Language Enhancements

The first thing to consider is that EsiObjects is a new programming language. As in the extension from C to C++, adoption of an Object-Oriented paradigm into any language can be expected to produce a new programming language. Although compatibility with current language standards is a must, one cannot expect to take advantage of the Object paradigm without changing some of the core elements of a procedural language.

EsiObjects accomplishes this rather simply. By adding only a few new commands and functions (Table 1) and by augmenting some existing syntactical elements, programming in EsiObjects remains very much the same as programming in standard M. Though it is not recommended in any Object-Oriented programming environment, you can still program using the same procedural language you have grown accustomed to. You can certainly execute any existing piece of M code from within EsiObjects.

Typeless Objects

Consistent with the M programming environment,

Commands	Functions	Special Variables
CR{EATE}¤	$ASN{VECTOR}¤	$CALLER¤
DE{STORY}¤	$AS{SOCIATE}¤	$CLASS¤
EV{ENT}¤	$CALL{BACK}¤	$ENV{IRONMENT}¤
WA{TCH}¤	$COPY¤	$HANDLE¤
IG{NORE}¤	$EX{IST}¤	$IN{TERFACE}¤
ZA{PPLY}¤	$EXTCALL{BACK}¤	$LIB{RARY}¤
¤	$FREECB¤	$MAXNUM¤
¤	$GETENT{RYREF}¤	$MESSAGE¤
¤	$INFO¤	$MINNUM¤
¤	$ISA¤	$PARAM{ETERS}¤
¤	$LIB{RARY}¤	$PARAMETERLIST¤
¤	$LOOKUP¤	$POINTER¤
¤	$NORMALIZE¤	$POOL¤
¤	$OIDPTR¤	$PRIV{ILEGED}¤
¤	$PTROID¤	$RET{URN}¤
¤	$PTRSTR¤	$SELF¤
¤	$QUO{TE}¤	$SUPER¤
¤	$VALID¤	$SYSPOOL¤
¤	$WALK¤	$ZVIRDATA¤
¤	$WATCH{DETECT}¤	¤
¤	$ZL{ENGTH}¤	¤
¤	$ZP{ECE}¤	¤

Table 1. EsiObjects Language Extensions

EsiObjects is a typeless environment. In this way, an object handle may reference an object of any class. This is not to say that EsiObjects doesn't support data typing. In fact, EsiObjects extends the typing capability of M to include user-defined data types. Data types in M are limited to character strings, integers and floating point numbers (depending on the context). EsiObjects provides runtime type identification, which extends context interpretation to all objects.

Variable Scoping

One common pitfall in M programming is the global scope of program variables. Unless explicitly stacked within a function or procedure call, any variable created within the module is visible to all other modules. EsiObjects provides scope indicators that allow the programmer to control the visibility of variables. A variable may be visible to all executing processes (Object domain), to all modules in a single executing process (Local), to all objects in a class (Class), to a specific object instance (Instance) or only during method execution (Accessor/Parameter/Temporary). The default scope of a variable (that which is effective when scope is unspecified) is Temporary.

Positional, Keyword and Array Parameters

EsiObjects object methods receive parameters in much the same form as the standard M procedure. This form of parameter passing assigns values to parameters positionally. EsiObjects extends the parameter passing mechanism to include Keyword and Array assignments. The "Input Specification" of EsiObjects allows the association of a keyword with an argument. Values may be assigned to a keyword argument (regardless of its position) by specifying the keyword in the method call. Array Parameters allow a virtually unlimited number of arguments. All arguments that positionally follow an Array Parameter are indexed into the parameter in an array structure.

Function vs. DO Call

EsiObjects eliminates the differentiation between a function call and a procedure call. Any object method can be invoked as a procedure or as an extrinsic function. Methods with no explicit return value will return an empty string when invoked as an extrinsic function. When invoked as a procedure, any method return value is simply ignored.

Commands

In addition to SETting and KILLing variables, EsiObjects provides CREATEing and DESTROYing objects. As one would expect, the CREATE and DESTROY commands are similar to their standard M counterparts. Creating an object, however, does require a little more effort than setting a variable. To this end, the EsiObjects syntax allows more information to be provided to the CREATE command. The initial state of an object can be determined at creation time through the use of keywords and parameter and property assignments.

The Model-View-Controller architecture supported by EsiObjects provides substance to the philosophy of separating business, database and user interface logic. This architecture requires an event-driven application model. EsiObjects supports the Model-View-Controller model at the language level with the provision of WATCH and IGNORE commands. With these commands, any object can WATCH for events that may occur in any other object. When the WATCHing object is no longer interested in this event, it may IGNORE the event. This architecture simplifies object interfacing and allows loose coupling of interacting objects. Rather than burdening an object with the task of managing side effects, the object method simply does its assigned task, then notifies the environment that an event has occurred. Any object that is interested in this event will be notified and can then take the appropriate action.

Functions

Several functions are introduced by EsiObjects to facilitate the Object-M extension. As mentioned above, EsiObjects is a typeless object environment. $ISA() provides type identification at execution time. $EXIST determines whether a variable is a handle for a valid object. $INFO retrieves various pieces of information about an object, including such things as the objects class and persistence state.

Other functions have been added to simplify some common tasks. These functions include $QUOTE, which will produce a quoted string from the text argument (doubling embedded quotes where necessary). $ZLENGTH provides a quote sensitive $LENGTH (delimiters enclosed in quotes are ignored), and $ZPIECE is the corresponding $PIECE function.

Environment

The perception of the quality of any programming language is based heavily on the libraries available for that language. In object programming languages, this depends primarily on the base classes provided with the

language. EsiObjects provides an extensive set of base classes that provide much of the foundation for building an object database application. Among these classes are the Collection and related utility classes, Documentation and Text classes and Immutable classes.

Immutable classes are the basis for object representations of M primitive data types. The Immutable classes provide "lightweight" objects, called virtual objects, which add an object oriented interface to M primitives including a simple Mvariable, NameValuePair and dates and time related classes. These classes are considered lightweight because the only data space allocated to the object is the object handle itself. The object wrapper provides properties and methods that interface with value of the object handle. Virtual objects also provide the foundation for legacy database wrappers in much the same fashion.

Documents and related classes provide handling for text blocks. These classes provide several string and document-related utility methods. A consistent interface to multi-line text blocks provides insertion, removal and searching capabilities. These classes can provide a convenient basis for supplying flexible support for large text blocks to any application.

Collection classes (and their related utility classes) are the core elements of any database application. The database itself is a collection of objects. Most data used by the application are processed in collections (lists, arrays, etc.). For this reason, the Collection class is probably the single most useful class in the library. The Collection is an abstract placeholder for more specialized collections (lists, arrays, bags and maps). Each of these specialized collections is managed and accessed differently. However, the Collection class defines a consistent interface to these collections. This interface includes the Iterator, for sequential traversal of the objects, and Criteria objects to conditionally select objects from the collection.

The base classes are made available through the two provided class libraries Base and ESI (Figure 1). In addition to these libraries, EsiObjects allows the developer to create custom class libraries. An EsiObjects library may be Concrete or Virtual. Concrete libraries contain the actual class definitions. Virtual libraries provide a means to group classes from the concrete libraries. Libraries are typically created to group classes by application. Utility libraries, such as the EsiObjects Base Library can be created for reuse among multiple applications. An optimal way to use these libraries would be to create Concrete

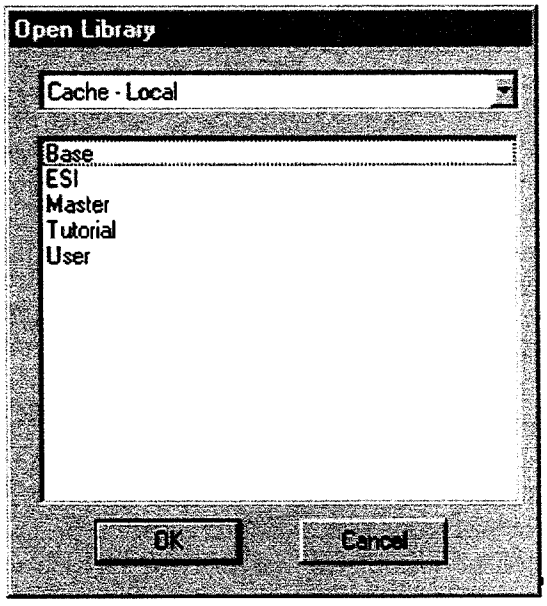

Figure 1

utility libraries and assemble Virtual libraries for applications.

Tools

The most visible part of any development environment is the tool set. EsiObjects doesn't disappoint here. EsiObjects provides an integrated development environment including Class Editors, the Execute Shell, Object Browsers and a Source Level Debugger.

The first tool to become familiar with is the Class Editor (Figure 2). The Class Editor is a combined Class Browser and Editor. Facilities are available for searching, creating, editing and deleting classes. The class editor is opened in association with a class library. The editor presents a three-pane view of classes in the library. The class hierarchy can be traversed using the tree view presented in the Class Editor's left pane. When an element (class, variable or interface) is selected in this tree view, the right pane displays pertinent details. Documentation is displayed (and can be edited) in the lower pane.

The Method and Property editors use a variation on this three-pane view providing the code, documentation and revision history in separate, visible panes. The code is edited in place in the left pane. A Right-click in this pane provides an extensive list of options including syntax checking, saving and compiling options. The right pane displays documentation for the method being edited and, as with the class editor, can be edited directly in this pane. The lower pane of this view displays the revision history of the method. A new version of the method can be created at any time in order to preserve existing working versions. Any

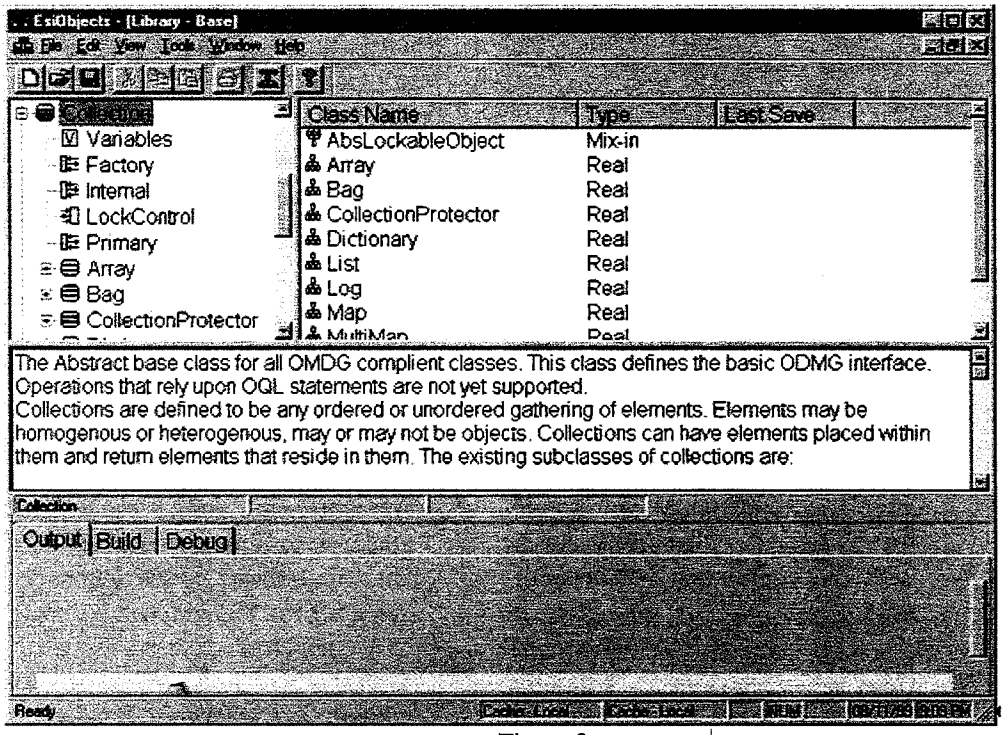

Figure 2

version of the method can be displayed by selecting the desired version from the revision history.

The Execute Shell (Figure 3) provides a programmer prompt-like interface to EsiObjects' object environment. From the shell you can interact with the environment and create and manipulate instances of classes (objects). This is a convenient means of testing and debugging class methods and properties.

The Object Browser is available from the Execute Shell. An Object Browser can be opened for any existing object. Within the Object Browser, you can view the entire contents of the active object in the object view pane. The Object Browser can also "step into" objects contained in the active object, allowing easy exploration of related objects. In addition, the Object Browser contains its own Execute Shell. Using this shell, you can execute commands and methods from within the scope of the active object. This is a very convenient way to test and debug classes. The effect that method execution has on the object can be seen immediately in the contents pane.

To add even more debugging capabilities, version 3.0 of EsiObjects introduces the Source Level Debugger. While the Object Browser allows you to view the overall effect of method execution, the Source Level Debugger allows you to step through the execution of a method. The debugger can be activated within any object that has been compiled in debug mode. The debugger provides controls for continuing execution, stepping into, over and out of commands and variable watches. The debugger also displays the contents of the objects (similar to the Object Browser) and the class call stack, providing a complete context to the executing method.

Conclusion

While it is difficult to determine exactly what impact the Object world will have on our world of M, EsiObjects provides many hints into the possibilities. The seamless extensions that provide Object-based services without sacrificing compatibility with the standard M language suggests that a relatively smooth migration is possible. The tools provided by EsiObjects are a shining example of the potential capabilities of the underlying environment. Using EsiObjects as a guide, it is not difficult to envision M succeeding in the Object Database world just as it has in the Relational Database world.

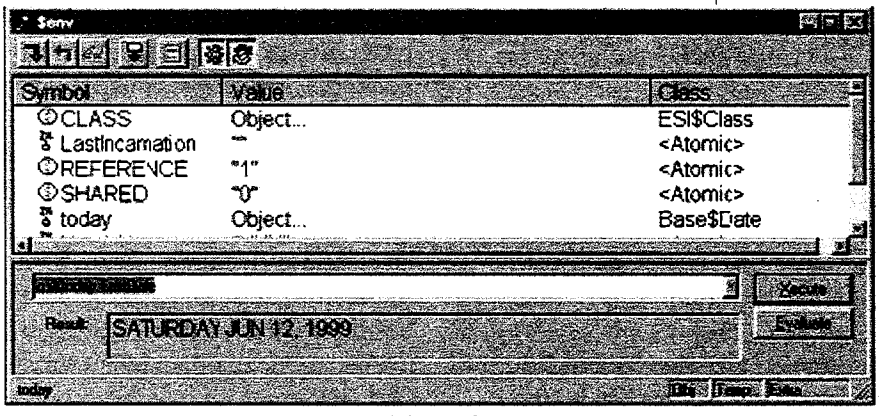

Figure 3

Fred Boles, (fdboles@home.com) is a TekMetrics certified Master M Programmer with nine years of M programming experience in the finance industry, as well as five years in technology research specializing in Object Oriented Software Development technologies.