Leveraging the Power of Off-the-Shelf Windows Applications

by Kristin N. Johnson and Steven K. Lathrop

Dynamic Data Exchange Gives M a Boost

The tools are now in place to allow the M programmer to write applications for the Microsoft Windows environment. One particularly interesting consequence of this development is that M applications can now take advantage of Windows' built-in interprocess communication protocol—Dynamic Data Exchange (DDE). If you have ever wanted to control Windows applications from M or share data with and among those applications without user intervention, DDE makes this possible.

DDE allows applications to "talk" to one another by sending data and instructions back and forth. It can be used for tasks as simple as placing values in the cells of a spreadsheet to projects as complex as coordinating the activity of all the applications on the Windows desktop. Many popular Windows applications support DDE, such as Microsoft Excel, Microsoft Word for Windows, and Borland's ObjectVision. Using DDE, M software can now moderate a user's access to most, if not all, of the functionality present in these (and other) Windows applications—thus seamlessly incorporating the features of off-the-shelf Windows applications into M systems. DDE gives M Technology a powerful link to heavily tested, commercial-quality software, thus obviating the need to develop code to perform these functions.

DDE support enhances M Technology in two major areas: language *extension* and *usage*. The functionality of M can be augmented more conveniently than with the traditional ZCALL, which is nonstandard and can be difficult to implement. With DDE, a rich set of functions provided by other applications can be embedded in M code. For example, Microsoft Excel 3.0 comes with a 238-page *Function Reference Manual*—every function or macro in that manual can now be called directly from M routines. Building on the superior database and routine development capabilities of M, DDE support positions M as a Windows desktop manager, as well as a database server accessible from other applications.

This article gives an overview of Dynamic Data Exchange as seen from the Windows system level and from its incarnation in DataTree's DT-Windows product at the M level (note that other M implementors such as Micronetics should soon have commercially available products with the same features as DT-Windows—if they do not already). Two examples are presented. One example shows the basics of establishing a DDE conversation and obtaining information from a Windows application. The other demonstrates the use of Microsoft Excel as a graphical spreadsheet and chart presentation tool.

Dynamic Data Exchange—An Overview

DDE is based on the event-driven nature of Microsoft Windows. Windows maintains an event queue for the applications running on the desktop. Whenever an event occurs in the system, such as a mouse click or menu selection, the appropriate application is notified. The application then responds by performing whatever processing is necessary. DDE uses this internal messaging system to allow two Windows programs to carry on a "conversation" by posting messages to one another. The program initiating a conversation is called the "client" (or destination) and the other is called the "server" (or source). A DDE server is a program that provides data and functions that may be useful to client programs.

To establish a DDE conversation, the client sends a WM_DDE_INITIATE message to the system event queue. This message specifies which application the client wishes to speak with and the topic of the conversation. The application specified must be the name of a Windows program, such as "EXCEL" for Microsoft Excel. The topic can be the name of any topic about which the DDE server is willing to converse—this is usually the name of a document file that was created by the server. In the case of Excel, a valid topic would be the name of a worksheet (e.g., "MY_SHEET.XLS"). If the DDE server application is available and the topic is valid, the server sends a WM_DDE_ACK message to the client acknowledging the receipt and processing of the WM_DDE_INITIATE message. Along with this message, the DDE client receives the "address" of the "mailbox" at which the server can be reached. This mailbox takes the form of a window handle—an integer identifier that is unique to each window. DDE clients and servers create a distinct (usually hidden) window for each DDE conversation that acts as a "mailbox" for messages—thus allowing client applications to carry on multiple DDE conversations with servers and vice versa.

With DDE, a rich set of functions provided by other applications can be embedded in M code.

Once a DDE conversation has been initiated, a number of messages may pass back and forth between client and server. Usually, the messages consist of client requests for data items and server responses. In order for the client to receive data from the server, a particular data item must be requested. In the case of Excel, a data item might be the name of a particular cell in a worksheet, such as "R1C1" for the cell in row one, column one. A one-time request for a data item is made by posting a WM_DDE_REQUEST message to the server. The server responds by posting a WM_DDE_DATA message to provide the client with the requested data item value. This is called a "cold" (or manual) DDE link, since no further discussion goes on between client and server concerning the specified data item.

The client application also can elect to send the server one of two other types of data-requesting messages. If the client wants an immediate response from the server (as in the case of the cold link described above) in addition to continued notification of changes in the data item's value and automatic transmission of the new value, the client posts one flavor of the WM_DDE_ADVISE message to the server. In response, the server posts a WM_DDE_DATA message (along with the new value) to the client whenever the value of the data item changes. This is known as a "hot" (or automatic) DDE Link. Another flavor of the WM_DDE_ADVISE message may be posted to the server if the client merely wants to be notified of changes in the data item's value. This is known as a "warm" (or notify) DDE Link. In this case, the server posts a WM_DDE_DATA message (without the value of the data item) to the client whenever the value of the data item changes. And as you might expect, to actually retrieve the new value of the data item, the client posts a WM_DDE-_REQUEST message to the server. A warm link is useful when the processing involved in continually transmitting new values of the data item to the client is prohibitive—as the case may be if the data item is large or unwieldy.

There are two other very useful DDE messages that the client may post to the server. One, the WM_DDE_EXECUTE message, allows the client to request that the server execute a procedure in the server's native programming, macro, or script language. The other message, WM_DDE_POKE, allows the client to send unsolicited data items to the server. For example, you could use this message to place values directly into the cells of an Excel worksheet or the fields of an ObjectVision form. The power of these two messages has not generally been exploited as fully as it might—even in the Windows community.

Of course, just being aware of the Windows messages involved in carrying on a DDE conversation does not get you even halfway down the road toward using DDE in applications. In order to use these system-level messages themselves you must master Windows programming in a language such as C. You must learn how to write a "WinMain()" function, register a Window Class, create windows, and write Window Procedures. But don't despair if this view of DDE seems a bit too complicated to dive into with great eagerness—even Microsoft acknowledges that there are better ways to manage DDE conversations than with these raw messages. With Windows 3.1 Microsoft included a new, higher-level interface to the DDE messages called the Dynamic Data Exchange Management Library (DDEML). And, as M programmers, we have come to expect a high level of abstraction between ourselves and the guts of a particular technology—this is one of the strengths of M. Fortunately, there is at least one M system that supports DDE effectively—DataTree's DT-MAX or DTM-PC system with the DT Windows API.

Using these DDE device functions in M . . . is a great deal easier than using the raw DDE messages themselves in a Windows application developed in C.

The Road to DDE

DataTree's DT Windows product is one tool that allows the M programmer to create Windows applications (the reader may be curious about how DataTree M, being a DOS application, can "talk" to Windows at all—please refer to the text box on *How DataTree M Talks to Windows*).

Continued on page 36

DT Windows provides access to many of Windows' system-level functions from M through the use of appropriate device functions (currently, using the WRITE /DevFuncName( ) syntax, but subject to the forthcoming M windowing API standard being developed by a MUMPS Development Committee (MDC) Task Group). DDE is supported with a "/WDataExchange" device function. There are a number of subfunctions to the "/WDataExchange" device function that roughly correspond to the raw Windows DDE messages (figure 1 shows the correspondence between the DDE messages described previously and the /WDataExchange subfunctions). Using these DDE device functions in M, however, is a great deal easier than using the raw DDE messages themselves in a Windows application developed in C.

Figure 2 shows an example of how to use the DT Windows API to initiate a DDE conversation with Excel and request the value of a data item.

Windows DDE Message	**DTWindows /WDataExchange Equivalent**
`WM_DDE_INITIATE`	`WRITE /WDataExchange(1, "Application", "Topic")`
`WM_DDE_REQUEST (Cold Link)`	`WRITE /WDataExchange(2, "Item")`
`WM_DDE_ADVISE (Hot Link)`	`WRITE /WDataExchange(3, "Item")`
`WM_DDE_ADVISE (Warm Link)`	`WRITE /WDataExchange(4, "Item")`
`WM_DDE_UNADVISE (Turn Off Hot or Warm Link)`	`WRITE /WDataExchange(5, "Item")`
`WM_DDE_TERMINATE (Close Conversation)`	`WRITE /WDataExchange(6)`
`WM_DDE_EXECUTE`	`WRITE /WDataExchange(7, "[CommandString]")`
`WM_DDE_POKE`	`WRITE /WDataExchange(8, "Item", "Value")`

Figure 1. DDE messages and equivalent subfunctions.

```
DDE1      ; Initiating a DDE Conversation with Excel and Requesting a Data Item
          ; Assumes that Excel is already running on the desktop and that "MY_SHEET.XLS" is loaded
          ; Open and Use a DT Window Device which handles the API functions
         O 40 U 40
          ; Direct device functions to the DT Window Manager and create a top-level (main)
          ; window that will act as the DDE Client in a conversation with Excel
         W /WUse(-1)
         W /WCreate(1,0,0,100,200,"Excel DDE Client")
          ; Direct device functions to the DDE client window and initiate a conversation with Excel
         W /WUse(1)
         W /WDataExchange(1,"Excel","MY_SHEET.XLS")
          ; Perform a one-time request for the value of the data item "R1C1" -- i.e., the value in
          ; cell A1 of the worksheet "MY_SHEET.XLS"
         W /WDataExchange(2,"R1C1")
          ; Pull the Data Exchange Result message (82) off the message queue and place the
          ; value of the data item in 'val'
         FOR W /WGetMessage(.x) IF +x=82,$p(x,$c(22),8)="R1C1" s val=$p(x,$c(22),9) QUIT
         C 40
         U 0
         W !,"The value in MY_SHEET, R1C1 is """_val_""""
         QUIT
```

Figure 2. M code: a DDE conversation with Excel.

To initiate a DDE conversation, the client application must create a window that acts as a "mailbox" for DDE messages. In DT Windows, this is accomplished by opening the device that accepts API functions, directing them toward the parent window desired (in this case, the parent is the DT Window Manager, since the window is to be a main, top-level window), and issuing the "/WCreate" device function. The next step is important, but easy to overlook: direct the "/WDataExchange" device toward the DDE Client "mailbox" window. The rest is explained in figure 2. On behalf of programming Windows from M in general, it can be said that creating a DDE-capable program in about ten (or what could actually be about five) lines of code speaks volumes for the superiority of M as a Windows development language. The same process in C requires somewhere in the range of eighty to one hundred lines of code. For more information on the specifics of programming with the DT Windows API, see DataTree's documentation.

A slightly more sophisticated Excel example shows just how much can be accomplished from M via DDE with relatively little effort. Admittedly, this code does not perform very complicated interactions with the DDE server (M could act as a DDE client at the same time and synchronize operations, etc.), but a little testing and experimentation could produce a much more sophisticated DDE-distributed system. Alternatively, you wouldn't want to try writing the code in DT Windows to produce a three-dimensional pie chart using low-level graphics calls. It all comes down to using the right tool for the job.

Creating a DDE-capable program . . . speaks volumes for the superiority of M as a Windows development language.

The example is based on DT Windows, but the details of how things are accomplished with DataTree's product are left out. In one way or another, these techniques can be made available from any PC-based M implementation capable of running under Windows.

Using Excel as a Chart Presentation Tool

Let's assume there is an online M system that must track connect-time and you want to produce a chart presenting the breakdown of connect time geographically. You might have an M global structure resembling figure 3.

Global Node	Value (Connect-time in hours)
`^X("CT", "Eastern")`	1,000
`^X("CT", "Central")`	700
`^X("CT", "Mountain")`	500
`^X("CT", "Pacific")`	900

Figure 3. M global structure for tracking connect-time geographically.

In order to chart this connect-time information using Excel, you must establish a DDE conversation with Excel as we did in the simple example above. In addition, you either start a pre-written Excel macro to do the charting once the data is in the cells of a worksheet, or you embed the Excel macro commands in the M code itself (the latter is a great deal more interesting—so that is what the code in figure 4 does).

The embedded Excel macro commands are transmitted to Excel via the WM_DDE_EXECUTE message. The M data is placed in an Excel worksheet using the WM_DDE_POKE message. Figures 5a and 5b show the results of the code in figure 4. Please refer to these figures on page 38.

Wrapping It Up

We hope that our look at Windows and DDE leaves you with a desire to examine the functions in any number of off-the-shelf applications appearing on desktops everywhere. Users who demand both glitzy graphical applications with powerful data presentation tools and the power that M technology provides in terms of data storage, manipulation, and retrieval, may find that leveraging the features of the applications they are already running is an excellent solution. Even if you are only now considering the move to Windows, you may streamline development efforts considerably by letting someone else's shrink-wrapped code do much of the work for you via Dynamic Data Exchange. **M**

Kristin N. Johnson is senior vice president of Sentient Systems, Inc. Steven K. Lathrop is the Microsoft Windows development team leader at Sentient. Steve holds a B.S. degree in mathematics from Grove City College, Grove City, PA. His work at Sentient has involved extending and integrating M Technology with other tools, languages, and environments such as Visual Basic, C/C++, and Microsoft Windows.