OpenView Windows: HP's New Foundation for Network Management

Kathleen Gannon

Hewlett-Packard Company
Information Networks Division
19420 Homestead Road
Cupertino, CA 95014

With the announcement of OpenView Windows, Hewlett-Packard starts a new era in Network Management. In the old era, users had to learn many different interfaces for their network management utilities, and often had just as many monitors to watch in their control rooms. With OpenView Windows, network management utilities can be combined on one station, with a unified user interface based on Microsoft Windows. Furthermore, a Developer's Kit is available to allow third parties and in-house developers to develop their own applications under OpenView Windows.

This paper focuses on the OpenView Windows Developer's Kit. First, it describes the features provided by OpenView Windows to an application. Next, it discusses the overall architecture. Finally, it features sample code modules to demonstrate the use of the intrinsic and message interface.

OPENVIEW WINDOWS FEATURES

From a developer's point of view, what are the benefits of OpenView Windows? The most obvious is that its easy to use interface, based on an industry standard, improves the saleability of the developer's product. Additionally, OpenView Windows provides services to the developer that reduce the amount of work necessary to get a product to market. These include the OpenView Windows network map, status management, menu integration and help utility.

The network map is the heart of OpenView Windows. It consists of a collection of pictures, linked togther by subnet symbols. A picture contains symbols representing network components for a portion of a network. A subnet symbol is a special symbol that represents one of the pictures ("subnets") in the map. It performs two functions for that picture: first, if the user double-clicks on the subnet symbol with the mouse, the corresponding picture is displayed; second, the subnet symbol takes on the color of the most critical alarm in its corresponding picture, thus providing a status summary. Like all symbols, subnet symbols can be used freely throughout the map. Using OVDRAW, the end user can draw a map of his network as he views it, grouping nodes and networks into pictures according to whatever scheme makes the most sense to him.

The symbols in the network map perform two functions. By changing color to display status, they provide a quick overview of the state of the network. They are also the key to

---

the integrated, object-oriented command style of OpenView Windows. Users simply click on the symbol representing the network component they wish to work on, then select the desired function from the menus. Behind the scenes, OpenView Windows determines which menu items to enable, based on the type of symbol selected and the network management applications installed, then routes the user's command to the appropriate application to perform. In this way, users need to learn only one command to peform a function on many different types of network components.

OpenView Windows provides five levels of status: Critical, Warning, OK, Offline and Unknown. Once a network management application has determined, using its own methods, which of these states a component is in, all the application needs to do is tell OpenView Windows the object's name and state. OpenView Windows will change the color of all instances of that object in the map to the new state. In addition, if it is a change to Critical or Warning, a message will be displayed to the user to notify him of the event, and a time- and date-stamped entry will be made in the OpenView Event Log. A menu item is available for the user to view this event log.

One of the main factors contributing to OpenView Windows' ease of use is its ability to overlay the functions of several network management applications onto a single menu item. The user no longer has to remember different syntaxes to perform similar functions on different components. OpenView Windows has defined some generic functions that should be common to many components. If the application wishes to provide the functionality for one of the generic functions, it registers for the type of object it manages, then registers for the desired menu item. When the user clicks on the registered object and menu item, the command will be routed to the application for it to execute. If an application wishes to provide additional functionality, it can add its own menus and menu items.

On-line help has been shown to greatly improve the usability of a product. OpenView Windows provides the same On-line Help Utility as HP's NewWave product. Application writers simply write the textual help file for the application, and assign context numbers to each topic. A program is then used to prepare this file for use by the Help Utility.

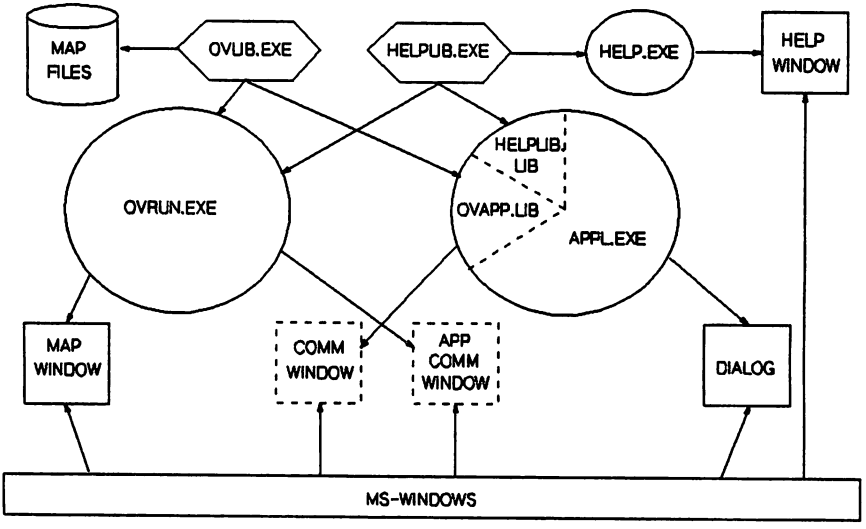
When users are running the application, they can request an index to the help topics. The application makes one procedure call and OpenView Windows handles the rest, allowing the users to browse through the topics to their heart's content. Likewise, if the application wants to show the user a particular topic, there is another call to start up the help window, automatically displaying the desired page. Once the window is displayed, all user interactions with it are managed by the Help Utility. No more effort is required on the part of the application.

OPENVIEW WINDOWS ARCHITECTURE

From the end user's point of view, there are three programs: OVRUN, OVDRAW and OVADMIN. OVRUN would typically be running continuously in a network control room, displaying the network map with the appropriate status colors. It contains functions for monitoring, diagnosing and controlling the network. OVDRAW is the utility used to draw the network map. OVADMIN is used to configure such things as passwords and defaults in the network management software itself.

When users start up one of the three programs, that program consults the initialization file, WIN.INI, and spawns each network management application installed there. Figure 1 shows the communications between one OpenView Windows program (OVRUN) and one network management application. OpenView Windows talks to the application via the Microsoft Windows message interface while the application talks to OpenView Windows via a procedural interface. This procedural interface masks the real implementation of the calls from the application. Some of the calls are implemented by the two dynamic libraries, NETWLIB and HELPLIB, while others are converted into messages. Two invisible windows act as message queues, one for the OpenView process and one for the application. The map window and the application's dialog windows are visible windows which comprise the user interface. In addition to OpenView messages, all windows will also receive messages from Microsoft Windows.

OpenView Windows Architecture



SAMPLE CODE MODULES

The following code segments were taken from a sample application called "Grape". The entire source for Grape would be too long to fit here, but a few procedures have been included to illustrate the use of the OpenView Windows programmatic interface. Procedures and messages starting with "OV" or "HELP" are part of this interface.

"Grape" performs the following functions:

1. Register for OBJ_COMPUTER symbol
2. Enable generic menu item, "Describe".

3. Add its own menu, "Measure".
4. Add its own menu item to the Measure menu, "My menu item".
5. Add a "Grapevine Index" menu item to the Help menu and enable the Help subsystem.
6. Respond to OV_GETVERSION message by returning a version string.
7. Re-start its status polling when it receives an OV_NEWMAP message.

```
/************************************************************************
 * InitApp                                                              *
 *                                                                      *
 *    This routine is called as soon as the application starts up,      *
 *    before it drops into its WinMain loop.  Since OpenView            *
 *    Windows waits until it receives OVInitComplete before             *
 *    spawning the next application, a well-behaved application         *
 *    does as little as possible here.                                  *
 *                                                                      *
 *    The things to be done in this procedure:                          *
 *         1.  Call OVInit to create the comm window for the app.       *
 *         2.  (If desired) register for one or more symbols types.     *
 *         3.  Add menus and menu items.                                *
 *         4.  Call OVInitComplete.                                     *
 *         5.  Initialize the Help subsystem.                           *
 *         6.  Any other application-specific initialization.           *
 ************************************************************************/
int InitApp( hInstance, hPrevInstance, lpszCmdLine, cmdShow )
HANDLE hInstance, hPrevInstance;
LPSTR lpszCmdLine;
int cmdShow;
{
        HANDLE hTask;
        int nError;
        char buf[40];
        OVPARM far * parmp = (OVPARM far *)lpszCmdLine;
        char szHelpDir[128];

        /* store the instance handle in a global variable for
           later use */
        hInst = hInstance;

        nError = OVInit( hInstance, (FARPROC)CommWndProc, lpszCmdLine,
                                (LPHWND) &hCommWnd );
        if( nError != OV_SUCCESS )
                MessagePrintf( 0, "Init %d", nError );

        /* MainAddMenu registers for an object and adds menu items */
        if ( MainAddMenu( hCommWnd ) != OV_SUCCESS)
                MessagePrintf(0, "Error adding menu.");

        nError = OVInitComplete();
        if( nError != OV_SUCCESS )
                MessagePrintf( 0, "Init-complete %d", nError );
```

```
        /* This is a Microsoft Windows call to retrieve a string
           from the initialization file, WIN.INI */
        GetProfileString( (LPSTR)"OpenView", (LPSTR)"HelpDir",
                          (LPSTR)"", (LPSTR)szHelpDir,
                          sizeof(szHelpDir) );

        /* Tell the helplib where your help text file is.  It returns
           a handle which must be used in all subsequent calls to the
           help facility */
        hHelp = HELP_Initialise( hCommWnd, hInstance,
                                 (LPSTR)szHelpDir,
                                 (LPSTR)"grape.hlp",
                                 (LPSTR)"Grapevine", FALSE );

        return TRUE;
}

/*************************************************************************
 *  MainAddMenu                                                          *
 *      Called during initialization to register to manage a             *
 *      particular type of object and to add menu items.                 *
 *************************************************************************/
MainAddMenu( hWnd)
HWND hWnd;
{
    char szBuf[MED_BUF_SZ + 1];
    int iResult;
    int iMenuID;

    /* Register for the object type that the application will
       manage. */
    iResult = OVRegister( OBJ_COMPUTER );
    if (iResult != OV_SUCCESS)
        return MainError( iResult, (LPSTR)"MainAddMenu",
                (LPSTR)"OVRegister" );

    /* Monitor menu */
    /* DESCRIBE: generic menu item */
    iResult = OVMenuAddItem( 0, (LPSTR)"", OV_IDMDESCRIBE,
                        OVM_ENABLED | OVM_OBJSPEC );
    if (iResult != OV_SUCCESS)
        return MainError( iResult, (LPSTR)"MainAddMenu",
                (LPSTR)"Describe" );

    /* New menu  "Measure" */
    LoadString( hInstance, IDSGV_MEASURE, (LPSTR)szBuf, MED_BUF_SZ);
    iResult = OVMenuAdd( (LPSTR)szBuf, (LPINT)&iMenuID);
    if (iResult != OV_SUCCESS)
        return MainError( iResult, (LPSTR)"MainAddMenu",
                (LPSTR)"Measure Menu" );
```

```
    /* Add to Measure menu: "My Menu Item" */
    LoadString(hInstance, IDSGV_MYMENUITEM, (LPSTR)szBuf, MED_BUF_SZ);
    iResult = OVMenuAddItem( iMenuID, (LPSTR)szBuf,
                        GV_OBJSPEC,
                        OVM_ENABLED | OVM_OBJSPEC );
    if (iResult != OV_SUCCESS)
        return MainError( iResult, (LPSTR)"MainAddMenu",
                (LPSTR)"My menu item" );

    /* Add HELP INDEX to Help Menu, Application specific */
    LoadString( hInstance, IDSGV_HELP, (LPSTR)szBuf, MED_BUF_SZ);
    iResult = OVMenuAddItem( OV_IDMHELP, (LPSTR)szBuf, GV_HELP,
                OVM_ENABLED );
    if (iResult != OV_SUCCESS)
        return MainError( iResult, (LPSTR)"MainAddMenu",
                (LPSTR)"Help Index" );

    return OV_SUCCESS;

}

/*************************************************************************
 *  CommWndProc                                                          *
 *    This procedure handles all messages sent to the application's      *
 *    communication window.  In this case, it only handles OpenView      *
 *    messages, and passes on any it doesn't care about to the           *
 *    default window procedure OVDefCommWndProc.   Since it is           *
 *    an invisible window, it doesn't need to act on most                *
 *    Microsoft Windows messages, however the application                *
 *    could if it desired.                                               *
 *************************************************************************/
long FAR PASCAL CommWndProc(hWnd, message, wParam, lParam)
HWND        hWnd;
unsigned    message;
WORD        wParam;
LONG        lParam;
{
      HANDLE hMem;
      LPSTR lpMem;

      switch (message) {
        case OV_SHUTDOWN:
            /* This message tells the application that
                OpenView Windows is going away, so it needs
                to shutdown. */
            HELP_Done(hHelp);
            OVDone(hHelp);
            PostQuitMessage(0);
            break;

        case OV_COMMAND:
            /* This message results from the user choosing a menu item
```

```
            which the application had enabled. */
        MainCommand(hWnd, wParam, HIWORD(lParam), LOWORD(lParam));
        break;

    case OV_GETVERSION:
        /* This message is generated when the user brings up the
           "About" box.  The version strings of all installed
           applications are displayed in a listbox within the
           About dialog. */
        hMem = GlobalAlloc( GMEM_MOVEABLE | 0x2000, (LONG) 256 );
        lpMem = GlobalLock( hMem );
        farstrcpy( lpMem, (LPSTR)"Grapevine   X.00.00" );
        while (*lpMem++)
            ;
        *lpMem = '\0';
        GlobalUnlock( hMem );
        return (LONG) hMem;

    case OV_NEWMAP:
        /* This message tells the app that the user has loaded
           a new map, so all previous status information has
           been erased.  The application needs to restart
           whatever method its using to monitor status on the
           components it is managing. */
        MainStatusInit();
        break;

    default:
        /* Let the default window procedure handle any other
           messages that may come in */
        return OVDefCommWndProc( hWnd, message, wParam, lParam );

    } /*end switch*/

  /* A window proc should always return something */
  return(1L);
}


/*********************************************************************
 *  MainCommand                                                      *
 *      Called when a OV_COMMAND message comes in, indicating that   *
 *      the user has selected a function from the menu.              *
 *********************************************************************/
int MainCommand(hWnd, wParam, wHiWord, wObjId)
HWND        hWnd;
WORD        wParam;
WORD        wHiWord;
WORD        wObjId;
{
  int iResult;
```

```
    switch (wParam) {
        case OV_IDMDESCRIBE:
            /* Display the Describe dialog box */
            NSDetailShow(hWnd);
            break;
        case GV_OBJSPEC:
            /* User selected "My menu item", so we put up a
               message box saying "My Menu Item". */
            MessagePrintf(0, "My menu item");
            break;
        case GV_HELP:
            /* User has invoked Help Index menu item */
            iResult = HELP_Index( hHelp);
            if (!iResult)
                MessagePrintf( 0, "Grapevine Help Index Error" );
            break;
        default:
            MessagePrintf( 0, "Grape_COMMAND, not recognized: %d",
                                                wParam);
        }
    return TRUE;
}

/*************************************************************************
 *  MainError                                                            *
 *      Error reporting is up to the application, but OpenView           *
 *      provides two intrinsics to help.  OVErrorMsg returns             *
 *      a text string to explain a OpenView Windows error number.        *
 *      OVLogWrite writes a message to a log file on disc which          *
 *      can be used by support personnel to trouble-shoot the            *
 *      Network Management software later.                               *
 *************************************************************************/
int MainError( nError, lpProcName, lpMsg)
int nError;
LPSTR lpProcName;
LPSTR lpMsg;
{
    char szMsg[ MED_BUF_SZ + 1];

    if ( lpMsg == (LPSTR)NULL ) {
        OVErrorMsg( nError, (LPSTR)szMsg, MED_BUF_SZ );
        lpMsg = (LPSTR)szMsg;
        }

    OVLogWrite( (LPSTR)"Grapevine", lpProcName, nError, lpMsg);
    MessageBox( 0, (LPSTR)lpMsg, (LPSTR)"Grapevine", OV_MSGERROR );

    return nError;
}
```