OPC for Windows® CE
A Whitepaper

Al Chisholm, OPC Technical Chairman

06/30/99

Introduction

Windows CE is a scaled down version of Microsoft Windows intended for use in embedded systems. It features a small footprint, good real time capability and the availability of powerful development tools. The fact that it can be burned into ROM greatly speeds startup of the system and also greatly reduces the need for the end user to setup and configure the system. Vendors incorporating CE into their hardware can optimize functionality and size by choosing which system components to include in the baseline sysgen and can also choose from several families of processors including x86, MIPS and SH3 based on cost, performance and their own preference and experience. CE also features off the shelf support for reliable, low cost hardware and software options including keyboards, displays, Comm ports and network adaptors. As anyone who has developed or used Ethernet for a PLC is aware, the availability of an off the shelf PCMCIA ethernet card and TCP/IP protocol stack can greatly reduce the cost and time required to add this capability to a PLC. The result is a reduction in time to market, development cost and manufacturing costs for new products. This translates into lower end user purchase and maintenance costs and frequently into a reduction in ongoing maintenance costs for the vendor as well.

OPC, which stands for OLE for Process Control and Factory Automation, is bringing the same ability to leverage standard off the shelf components to the Process Control and Factory Automation Software industry and it's users. Specifically, it defines software interfaces that allow easy transfer of data between programs that provide automation data (Servers) and programs that use automation data (Clients). OPC has demonstrated success and interoperability on Windows NT, Windows 95 and 98 and even on NT running on the DEC Alpha Platform. With the growing popularity and cost effectiveness of Windows CE, there is obviously a lot of interest in leveraging OPC technology on the CE platform.

This paper briefly describes the technical challenges in porting OPC Code to CE.

The Challenge

So, does OPC work on CE? The answer is a qualified yes. The main issue is that at this time, CE supports only an in-process server (DLL). This is one of the three models (in-process, local and remote) that are supported by COM and by OPC. As we will see below, there are a number of other minor issues with CE but none of them are critical. The remainder of this paper will discuss the missing features, their impact (if any) on interoperability and workarounds for each issue. This information should serve as a guide to anyone porting OPC Code to windows CE.

General Issues when porting code to Windows CE

In general Windows CE is a 'windows like' operating system. The Win32 API supported by CE is similar to the one on NT, but some things are missing, some new things are added and a few things behave differently. More important to most programmers (although less so to OPC programmers) is the User Interface which, from the programmer's point of view, is very different from NT. All of this is not surprising under the circumstances (completely new hardware platform and CPU set) however it is not terribly good news for programmers trying to port code. What this means is that existing code can be ported to CE more easily than it could be ported to say OS/2 or UNIX but, depending on the nature of the code, it could be a lot of work.

A discussion of specific issues that affect OPC follows.

No DCOM:
There is no DCOM support built into CE at this time. This means that only in-process servers will work on standard Windows CE. This does not really create any interoperability issues. In any case, in-process servers are the most efficient and on CE with its somewhat limited CPU capability this can only help. In the mid to long term, there are already third party solutions which provide capabilities similar to DCOM on CE and Microsoft has announced tentative plans to support DCOM in the next major release of CE (Version 3.0) later in 1999 or 2000.

No CoGetMalloc:
This COM function gives the programmer very efficient access to a global memory allocater. It was used in the original OPC Sample code and is frequently used in servers. However the functions CoTaskMemAlloc and CoTaskMemFree perform the same function and can easily be used in its place. The various SysAlloc functions used for BSTR manipulation are all present. Again there are no interoperability issues.

No Component Categories:
Newer OPC Servers use Component Categories as part of their registration. This allows clients to easily browse for available servers. This capability is not provided in CE at this time. However this is not critical to OPC functionality. The only effect is to limit the ability of a Client program on CE to offer a list of servers to the user in a list box. There are several easy work arounds for this. The Client can scan the registry for the 'OPC' key documented in the 1.0A specification, the Client could 'hard code' the name of the server to which it will connect, or the client can simply instruct the User to consult the documentation provided with the installed OPC servers and manually enter the name of the server.

Minimal Registry function support:
The CLSIDFromPROGID function is missing of windows CE. This allows a client to easily convert a symbolic server name such as "Intellution.OPCEDA.1” into a CLSID that can be passed to CoCreateInstanceEx. Two fairly simple alternatives to this are (a) to enter the CLSID into the client (which is awkward since it is a 32 digit hex number) or (b) the Client can implement a local version of the CLSIDFromPROGID function using

lower level registry functions. A version of this function for CE, generously provided by Microsoft, is included at the end of this paper.

No CoFileTimeNow support:
The CoFileTimeNow function is missing of windows CE. This is used in the same code and frequently used in servers. A version of this function for CE, generously provided by Jim Luth of Iconics is also included at the end of this paper.

No CreateDataAdviseHolder:
This library function is useful (but not required) when implementing the IDataObject interface. It is not present on CE. The sample code shows does not use this function but actual implementations may do so. The techniques used in the sample code will work on CE.

Summary
In summary CE offers many advantages to the hardware developer. In its current form it also offers excellent support for In-Process OPC Servers. The OPC Interfaces can be fully supported on Windows CE. In addition, existing OPC Server code generally will not rely on features of NT or CE that are in conflict and therefore should be relatively easy to port to CE. Both OPC Client and OPC Server code can be expected to offer the same high level of interoperability between components from different vendors when running on Windows CE as has been seen already on Windows NT and 9X.

Appendix A: CE Version of CLSIDFromPROGID

```
#define cchReg (140)   // max ProgID size
//
// WINCE OLE doesn't implement CLSIDFromProgID
//
HRESULT WceCLSIDFromProgID(LPCOLESTR lpszProgID, LPCLSID lpclsid)
{
    // check for CLSID\{progid}

    TCHAR wszClsid[cchReg];
    HKEY hkeyReg;
    HRESULT hr;
    LONG    ret;
    DWORD cb, dwType;

    ret = RegOpenKeyEx(HKEY_CLASSES_ROOT, lpszProgID,
        0, KEY_READ, &hkeyReg);
    if(ret == ERROR_SUCCESS)
    {
        cb = sizeof(wszClsid);
        ret = RegQueryValueEx(hkeyReg, TEXT("CLSID"),
            NULL, &dwType,(BYTE*) wszClsid, &cb);
        if(ret == ERROR_SUCCESS && (dwType == REG_SZ))
            hr = CLSIDFromString(wszClsid, lpclsid);
        else
        {
            HKEY hkeyClsid;
            ret = RegOpenKeyEx(hkeyReg, TEXT("CLSID"),
                0,KEY_READ, &hkeyClsid);
            if (ret == ERROR_SUCCESS)
            {
                ret = RegQueryValueEx(hkeyClsid,
                    TEXT(""), NULL, &dwType,
                    (BYTE*) wszClsid, &cb);
                if(ret == ERROR_SUCCESS)
                {
                    if (dwType == REG_SZ)
                         hr = CLSIDFromString(wszClsid,
                             lpclsid);
                    else
                        hr = E_UNEXPECTED;
                }
                else
                    hr = HRESULT_FROM_WIN32(ret);
            }
            else
                hr = HRESULT_FROM_WIN32(ret);
        }
    }
    else
        hr = HRESULT_FROM_WIN32(ret);

    return hr;
}
```

Appendix B: CE Version of CoFileTimeNow

```
HRESULT CoFileTimeNow(  FILETIME * lpFileTime  )
{
    SYSTEMTIME st;
    GetSystemTime( &st );
    return( SystemTimeToFileTime( &st, lpFileTime ) ? S_OK :
E_FAIL);
}
```