

October 2000

20001008

Prepared by:

Alain Lissoir - Technology
Consultant

Professional Services Division
Compaq Computer Corporation

Acknowledgments

Jerry Cochran (Compaq Computer)
Andrew Gent (Compaq Computer)
Vlad Joanovic (Microsoft Corporation)
Naveen Kachroo (Microsoft Corporation)
Jan Kelly (Compaq Computer)
Kevin Laahs (Compaq Computer)
Travis Muhlestein (Microsoft Corporation)
Lev Novik (Microsoft Corporation)
Dominic Pouzin (Microsoft Corporation)
Tony Redmond (Compaq Computer)
Barry Steinglass (Microsoft Corporation)
Caroline Takeuchi (Compaq Computer)
Patrick Tousignant (Microsoft Corporation)
Vladimir Vulovic (Microsoft Corporation)

Part 2: Managing Exchange with Scripts, Advanced Topics

Abstract:

This second part of the document presents two advanced WSH scripts based upon the new Exchange 2000 COM technologies. These scripts are able to perform automated management tasks in a real Windows 2000/Exchange 2000 environment.

This document uses the technology described in “Part 1 - Introduction to the use of Exchange 2000 with Windows Script Host” which is a pre-requisite for further reading.

Notice

The information in this publication is subject to change without notice and is provided "AS IS" WITHOUT WARRANTY OF ANY KIND. THE ENTIRE RISK ARISING OUT OF THE USE OF THIS INFORMATION REMAINS WITH RECIPIENT. IN NO EVENT SHALL COMPAQ BE LIABLE FOR ANY DIRECT, CONSEQUENTIAL, INCIDENTAL, SPECIAL, PUNITIVE, OR OTHER DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, OR LOSS OF BUSINESS INFORMATION), EVEN IF COMPAQ HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

The limited warranties for Compaq products are exclusively set forth in the documentation accompanying such products. Nothing herein should be construed as constituting a further or additional warranty.

This publication does not constitute an endorsement of the product or products that were tested. The configuration or configurations tested or described may or may not be the only available solution. This test is not a determination of product quality or correctness, nor does it ensure compliance with any federal, state or local requirements.

Compaq, Compaq Insight Manager, Deskpro, FASTART, NetFlex, NonStop, PaqFax, Proliant, Prosignia, QuickFind, Qvision, RomPaq, SmartStart, and Systempro/LT are registered with the United States Patent and Trademark Office.

ActiveAnswers is a trademark and/or service mark of Compaq Information Technologies Group, L.P.

Microsoft, Windows and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Intel, Pentium and Pentium® III Xeon are trademarks and/or registered trademarks of Intel Corporation.

Other product names mentioned herein may be trademarks and/or registered trademarks of their respective companies.

©2000 Compaq Computer Corporation. All rights reserved. Printed in the U.S.A.

Part 2: Managing Exchange with Scripts, Advanced Topics
Technical Guide prepared by Alain Lissoir - Professional Services Division

First Edition (October 2000)

Document Number 20001008

Table of contents

PREREQUISITES	6
ADVANCED SCRIPT SAMPLES	7
MOVING EXCHANGE 2000 MAILBOXES BASED ON AN LDAP QUERY	7
EXCHANGE 2000 SERVER ACTIVITY WATCHER.....	13
Windows 2000 level	14
Exchange 2000 level.....	15
E2K watch.Wsf script.....	16
The configuration file	16
The WMI initialization	18
WMI <i>Win32_Process</i> class	18
WMI <i>Win32_Processor</i> class	20
WMI <i>Win32_LogicalDisk</i> class	20
WMI <i>Win32_NTLogEvent</i> class	20
WMI <i>CIM_DATAFile</i> class	21
WMI <i>NTProcess</i> class	22
WMI <i>ExchangeServerState</i> class.....	22
WMI <i>ExchangeConnectorState</i> class.....	22
WMI <i>ExchangeLink</i> class	23
WMI <i>ExchangeQueue</i> class.....	23
WMI <i>ExchangeClusterResource</i> class.....	23
The event routines.....	24
WMI <i>Win32_Process</i> class	24
WMI <i>Win32_Processor</i> class	26
WMI <i>Win32_LogicalDisk</i> class	26
WMI <i>Win32_NTLogEvent</i> class	27
WMI <i>CIM_DATAFile</i> class	28
WMI <i>NTProcess</i> class	30
WMI <i>ExchangeServerState</i> class.....	30
WMI <i>ExchangeConnectorState</i> class.....	31
WMI <i>ExchangeLink</i> class	31
WMI <i>ExchangeQueue</i> class.....	32
WMI <i>ExchangeClusterResource</i> class.....	33
Added-functions not directly related to the Exchange 2000 Management	33
The possible enhancements.....	35
CONCLUSION	37

Table of Samples

Sample 1 Moving mailboxes from an <i>ADSI.User</i> object based on a LDAP Query	8
Sample 2 Moving mailboxes using a <i>CDO.Person</i> object based on a LDAP Query	10
Sample 3 A basic WMI Asynchronous event handler for the <i>Win32_Service</i> class	13
Sample 4 The WMI <i>Win32_Service</i> class asynchronous event registration	19
Sample 5 The WMI <i>Win32_Processor</i> class asynchronous event registration	20
Sample 6 The WMI <i>Win32_LogicalDisk</i> class asynchronous event registration	20
Sample 7 The WMI <i>Win32_NTLogEvent</i> class asynchronous event registration	20
Sample 8 The WMI <i>CIM_DATAFile</i> class asynchronous event registration	21
Sample 9 The WMI <i>NTPProcess</i> class asynchronous event registration	22
Sample 10 The WMI <i>ExchangeServerState</i> class asynchronous event registration	22
Sample 11 The WMI <i>ExchangeConnectorState</i> class asynchronous event registration	22
Sample 12 The WMI <i>ExchangeLink</i> class asynchronous event registration	23
Sample 13 The WMI <i>ExchangeQueue</i> class asynchronous event registration	23
Sample 14 The WMI <i>ExchangeClusterResource</i> class asynchronous event registration	23
Sample 15 The WMI <i>Win32_Service</i> class asynchronous event handler	25
Sample 16 The WMI <i>Win32_Processor</i> class asynchronous event handler	26
Sample 17 The WMI <i>Win32_LogicalDisk</i> class asynchronous event handler	26
Sample 18 The WMI <i>Win32_NTLogEvent</i> class asynchronous event handler	27
Sample 19 The WMI <i>CIM_DATAFile</i> class asynchronous event handler	28
Sample 20 The WMI <i>NTPProcess</i> class asynchronous event handler	30
Sample 21 The WMI <i>ExchangeServerState</i> class asynchronous event handler	30
Sample 22 The WMI <i>ExchangeConnectorState</i> class asynchronous event handler	31
Sample 23 The WMI <i>ExchangeLink</i> class asynchronous event handler	32
Sample 24 The WMI <i>ExchangeQueue</i> class asynchronous event handler	32
Sample 25 The WMI <i>ExchangeClusterResource</i> class asynchronous event handler	33

Table of Figures

Figure 1: The E2KWatch configuration file	16
Figure 2: A sample "E2KWatch" mail alert for the <i>increasingTime</i> value	36

Prerequisites

The reader must have an understanding of the following technologies:

- Windows 2000 global architecture
- Active Directory
- Exchange 2000 global architecture
- VBScript or JavaScript especially from the Windows Scripting Host (WSH) environment
- Active Directory Service Interfaces (ADSI)
- Windows Management Instrumentation (WMI)
- ActiveX Data Objects (ADO)
- Part 1 - Introduction to the use of Exchange 2000 with Windows Script Host

Advanced Script samples

Based on the different COM technologies discovered when exploring the Exchange 2000 COM hierarchy in Part 1, we will use these COM objects to build two advanced scripts. The presented scripts can be used as they are or can be used as a skeleton to build your own management script for Exchange 2000.

Moving Exchange 2000 Mailboxes based on an LDAP Query

In Part 1, Sample 11 and Sample 13 show how to use the *IMailboxStore* interface aggregated to an *ADSI.User* object or to a *CDO.Person*. The same scripts show where to find the *MoveMailbox* method associated with these interfaces in the Exchange 2000 COM object hierarchy. By combining, the *ADSearchFunction()* used in:

- In Part 1, Sample 10 (Retrieving the mailbox homed on a Mailbox Store and the associated mailbox-enabled object properties from ADSI)
- In Part 1, Sample 12 (Retrieving the mailbox homed on a Mailbox Store and the associated mailbox-enabled object properties from CDOEX)

It is possible to write a script that moves mailboxes between Mailbox Stores, Storage Groups and Exchange 2000 Servers based on the result of an LDAP query. For instance, if an Administrator wants to move all users in a department to one Server or to a particular Exchange Store, it is possible for him to perform an LDAP query against the Active Directory and move the users found by this query to a given Mailbox Store in a specific Storage Group on a particular Server. This is the purpose of the next sample.

As it was demonstrated before, ADSI (Sample 1 on page 8) or CDOEX (Sample 2 on page 10) can be used to access a mailbox. Both scripts use the same logic.

Sample 1 on page 8 starts by retrieving the current computer name (lines 89 to 92). The purpose is to ease the task of the operator when the help is invoked. The parameters will be shown with the current computer name in the usage samples (lines 93 to 113). This is the only reason for the *strComputerName* variable. If the script is run locally on the Exchange 2000 server and if the purpose is to move mailboxes locally on the server, the help line will be correct for a direct usage.

From lines 115 to 120, the script retrieves important required parameters to perform a mailbox move operation.

The first parameter is the LDAP Query string (line 115). This string will be used as a filter by the *ADSearch()* function to perform the query (lines 152 to 156). The syntax used for the query must be LDAP compliant. The query is executed in the Default Active Directory context (lines 126 and 153). The Default Active Directory context is determined by the domain membership of the machine where the script is run. So, only the users part of this domain will be retrieved.

Next, based on the different parameters from the command line, the *distinguishedName* of the target Mailbox Store is constructed (lines 131 to 139).

- At line 131, the Mailbox Store name retrieved from the command line (line 116) is used.
- At line 132, the Storage Group (of the given Mailbox Store) retrieved from the command line (line 117) is used.
- At line 134, the Server name (of the given Storage Group) retrieved from the command line (line 118) is used.

- At line 135, the Administration Group name (of the given Server name) retrieved from the command line (line 119) is used.
- At line 137, the Organization name (of the given Administration Group) retrieved from the *GetExchangeOrg()* function call (line 120) is used. For more information about the *GetExchangeOrg()* function, see Part 1, Sample 25.

At line 139, the Root Domain of the Tree retrieved at line 127 is used to complete the Mailbox Store *distinguishedName*. In this case, only the Root Domain name can be used because the Active Directory Configuration Naming Context is always below the Root Domain name of the Tree in the directory structure.

Sample 1 Moving mailboxes from an *ADSI.User* object based on a LDAP Query

```

1:<!-- VB Script making LDAP searches in the Active Directory on a user objectClass, -->
2:<!-- matching users are moved from their current Information Store to the -->
3:<!-- the new given store. -->
4:<!-- The script uses the ADSI User object class to be attached to the mailbox for -->
5:<!-- the move operation. -->
6:<!-- -->
7:<!-- Version 1.00 - Alain Lissoir -->
8:<!-- Compaq Computer Corporation - Professional Services - Belgium - -->
9:<!-- -->
10:<!-- Any comments or questions: EMail:alain.lissoir@compaq.com -->
..:
..:
45:<job>
46:     <script language="VBScript" src=".\Functions\ADSearchFunction1.vbs" />
47:     <script language="VBScript" src=".\Functions\GetMSExchangeOrgFunction.vbs" />
48:     <script language="VBScript" src=".\Functions\TinyErrorHandler.vbs" />
49:
50:     <script language="VBScript">
51:
52:         Option Explicit
53:
54:         ' -----
55:         On Error Resume Next
56:
57:         ..:
58:         Set WNetwork = Wscript.CreateObject("Wscript.Network")
59:         strComputerName = WNetwork.ComputerName
60:         Wscript.DisconnectObject (WNetwork)
61:         Set WNetwork = Nothing
62:
63:         Set objArguments = Wscript.Arguments
64:
65:         If objArguments.Count <> 5 Then
66:             Wscript.Echo "Usage:"
67:             Wscript.Echo " QueryAndMoveMBTo " & _
68:                 chr(34) & "(LDAPQueryFilter)" & chr(34) & " " & _
69:                 chr(34) & "TargetStore" & chr(34) & " " & _
70:                 chr(34) & "TargetStorageGroup" & chr(34) & " " & _
71:                 chr(34) & "TargetServer" & chr(34) & " " & _
72:                 chr(34) & "TargetAdministrativeGroup" & chr(34)
73:
74:             Wscript.Echo
75:             Wscript.Echo "Sample:"
76:             Wscript.Echo " QueryAndMoveMBTo " & _
77:                 chr(34) & "(givenName=J*)" & chr(34) & " " & _
78:                 chr(34) & "Mailbox Store B" & chr(34) & " " & _
79:                 chr(34) & "First Storage Group" & chr(34) & " " & _
80:                 chr(34) & strComputerName & chr(34) & " " & _
81:                 chr(34) & "First Administrative Group" & chr(34)
82:
83:             Wscript.Quit (1)
84:         End If
85:
86:         strLDAPQueryFilter = objArguments (0)
87:         strTargetStore = objArguments (1)
88:         strTargetStorageGroup = objArguments (2)
89:         strTargetServerName = objArguments (3)
90:         strTargetAdministrativeGroup = objArguments (4)
91:         strOrganization = GetExchangeOrg ()
92:
93:         ' -----
94:         ' Get the default Windows 2000 Domain name
95:         Wscript.Echo "Binding to RootDSE to get default Domain Name."

```

```

125: Set objRoot = GetObject("LDAP://RootDSE")
126: strDefaultDomainNC = objRoot.Get("DefaultNamingContext")
127: strRootDomainNC = objRoot.Get("RootDomainNamingContext")
128: Set objRoot = Nothing
129:
130: ' Check if the target store exists.
131: strTargetHomeMDB_DN = "cn=" & strTargetStore & "," & _
132: "cn=" & strTargetStorageGroup & "," & _
133: "cn=InformationStore," & _
134: "cn=" & strTargetServerName & ",cn=Servers," & _
135: "cn=" & strTargetAdministrativeGroup & _
136: " ,cn=Administrative Groups," & _
137: "cn=" & strOrganization & "," & _
138: "cn=Microsoft Exchange,cn=Services,cn=Configuration," & _
139: strRootDomainNC
140:
141: Set objTargetHomeMDB = CreateObject("CDOEXM.MailBoxStoreDB")
142: objTargetHomeMDB.DataSource.Open (strTargetHomeMDB_DN)
143: If Err.Number Then ErrorHandler (Err)
144:
145: If objTargetHomeMDB.Status Then
146:     WScript.Echo "Target Store '" & objTargetHomeMDB.Name & "' is not mounted."
147:     WScript.Quit (1)
148: End If
149:
150: ' -----
151: ' Query for the user's mailbox to move to.
152: Set objResultList = ADSearch ("LDAP://" & strDefaultDomainNC, _
153:     strLDAPQueryFilter, _
154:     "ADsPath, homeMDB", _
155:     "subTree", _
156:     False)
157: WScript.Echo
158: WScript.Echo "Number of matches for the LDAP query is " & _
159:     objResultList.Item ("RecordCount")
160: WScript.Echo
161:
162: ' -----
163: objResult = objResultList.Items
164:
165: ' We ask to return two elements from the LDAP query "ADsPath, homeMDB".
166: ' The very first element contains the number of records, so it is skipped.
167: ' (intIndice=1)
168: ' Odd elements contain "ADsPath", even elements contain "homeMDB".
169:
170: For intIndice = 1 to (objResultList.Count - 1) Step 2
171:     strUserADsPath = objResult (intIndice)
172:     strSourceHomeMDB_DN = objResult (intIndice + 1)
173:
174:     Set objUser = GetObject(strUserADsPath)
175:
176:     If LCase(strSourceHomeMDB_DN) = LCase(strTargetHomeMDB_DN) Then
177:         WScript.Echo "Skipping user '" & objUser.DisplayName & _
178:             "'. Actual Store and Target Store are the same."
179:     ElseIf _
180:         Isnull (strSourceHomeMDB_DN) Then
181:         WScript.Echo "Skipping user '" & objUser.DisplayName & _
182:             "'. It doesn't have any mailbox."
183:     Else
184:         Set objSourceHomeMDB = CreateObject("CDOEXM.MailBoxStoreDB")
185:         objSourceHomeMDB.DataSource.Open (strSourceHomeMDB_DN)
186:         If Err.Number Then ErrorHandler (Err)
187:
188:         If objSourceHomeMDB.Status Then
189:             WScript.Echo "Source Store '" & objSourceHomeMDB.Name & _
190:                 "' is not mounted."
191:             WScript.Quit (1)
192:         End If
193:
194:         WScript.Echo "Moving user '" & objUser.DisplayName & "' from:"
195:         WScript.Echo " Store '" & objSourceHomeMDB.Name & _
196:             "' to Store '" & objTargetHomeMDB.Name & "'"
197:
198:         ' Moving the mailbox.
199:         objUser.MoveMailbox "LDAP://" & strTargetHomeMDB_DN
200:         If Err.Number Then ErrorHandler (Err)

```

```

201:
202:         WScript.DisconnectObject objSourceHomeMDB
203:         Set objSourceHomeMDB = Nothing
204:     End If
205:
206:         Set objUser = Nothing
207:     Next
...:
212:     </script>
213:</job>

```

At line 141, an object used to get information about the target store specified on the command line is instantiated. This allows us to verify if the given store exists (lines 142 and 143) and if it is mounted (line 145). Once completed the LDAP query is executed (lines 152 to 156). Two important pieces of information are retrieved by the query:

- The *ADsPath* of the user object to bind to (line 174). The purpose is to use the aggregated *IMailboxStore* interface to move the mailbox (line 199). Another purpose (less important) is to show its display name during the move operation (lines 177, 181 and 194).
- The *homeMDB* property to perform some checks during the move mailbox operation:
 - A check to verify that the current mailbox store of the user is not the same as the target one (line 176).
 - A check to verify that the user is mailbox-enabled. If not, the *homeMDB* property is not set in the directory (line 180).

Note: It is also possible to verify if the user is mailbox-enabled in the LDAP query. In such a case, it is necessary to combine the user query from the command line with this supplemental condition (*homeMDB=**). This requires some string manipulations in the script. This is the only reason why this test is made here and not in the LDAP query.

- A check to verify that the current mailbox store of the user is mounted (line 188)

Once all the checks are completed successfully, the mailbox is moved (line 199).

The script then processes the next user in the list, if any (line 207).

Sample 2 below is exactly the same code as Sample 1 on page 8 up to the line 170. Line 174 uses a *CDO.Person* object instead of an *ADSI.User* object. Line 175 uses the *GetInterface* method to aggregate the *IMailboxStore* interface to the *CDO.Person* object. Except for these changes (and the related variable declarations), both scripts use exactly the same logic. Sample 2 below prints the adaptation made from line 170 to use the *CDO.Person* object.

Sample 2 Moving mailboxes using a *CDO.Person* object based on a LDAP Query

```

...:
...:
170:         For intIndice = 1 to (objResultList.Count - 1) Step 2
171:             strUserADsPath = objResult (intIndice)
172:             strSourceHomeMDB_DN = objResult (intIndice + 1)
173:
174:             Set objPerson = CreateObject ("CDO.Person")
175:             Set objMailbox = objPerson.GetInterface ("IMailboxStore ")
176:
177:             objPerson.DataSource.Open (strUserADsPath)
178:
179:             If LCase(strSourceHomeMDB_DN) = LCase(strTargetHomeMDB_DN) Then
180:                 WScript.Echo "Skipping user '" & objPerson.Fields("displayName") & _
181:                     "' . Actual Store and Target Store are the same."

```

```

182:         ElseIf _
183:             Isnull (strSourceHomeMDB_DN) Then
184:                 WScript.Echo "Skipping user '" & objPerson.Fields("displayName") & _
185:                     "'. It doesn't have any mailbox."
186:             Else
187:                 Set objSourceHomeMDB = CreateObject("CDOEXM.MailBoxStoreDB")
188:                 objSourceHomeMDB.DataSource.Open (strSourceHomeMDB_DN)
189:                 If Err.Number Then ErrorHandler (Err)
190:
191:                 If objSourceHomeMDB.Status Then
192:                     WScript.Echo "Source Store '" & objSourceHomeMDB.Name & _
193:                         "' is not mounted."
194:                     WScript.Quit (1)
195:                 End If
196:
197:                 WScript.Echo "Moving user '" & objPerson.Fields("displayName") & "' from:"
198:                 Wscript.Echo " Store '" & objSourceHomeMDB.Name & _
199:                     "' to Store '" & objTargetHomeMDB.Name & "'
200:
201:                 ' Moving the mailbox.
202:                 objMailbox.MoveMailbox "LDAP://" & strTargetHomeMDB_DN
203:                 If Err.Number Then ErrorHandler (Err)
204:
205:                 objPerson.DataSource.Save
206:
207:                 WScript.DisconnectObject objSourceHomeMDB
208:                 Set objSourceHomeMDB = Nothing
209:             End If
210:
211:             Set objMailBox = Nothing
212:
213:             WScript.DisconnectObject objPerson
214:             Set objPerson = Nothing
215:         Next
216:
217:         WScript.DisconnectObject objTargetHomeMDB
218:         Set objTargetHomeMDB = Nothing
219:
220:     </script>
221:</job>

```

Here are two command line samples to move mailboxes:

To move all the users with a first name starting with “J” to a mailbox store named “Mailbox Store B” member of the “First Storage Group” located on “MyServer” which is part of the “First Administration Group”, use the command line:

```

QueryAndMoveMBto (ADSI).vbs" "(givenName=J*)" "
                    "Mailbox Store B" "First Storage Group"
                    "MyServer"
                    "First Administrative Group"

```

Note: Each parameter has been placed on a different line to ease the reading. From the command prompt, these parameters must be typed on the same line separated by a space.

- To move all the users located on a specific Mailbox Store, you must use a more complex LDAP query. The query is based on the *distinguishedName* of the Mailbox Store homing the users. The important point here, is not to move the SystemMailbox, this is why there is a `(!(cn=Sys*))` statement in the query. Other parameters are the same as the prior sample.

Note: At first glance, we may think that the query can be made with the *objectCategory* property but the SystemMailbox is an *objectCategory* user disabled in the Active

Directory. So, a query based on the category will not sort the SystemMailbox of the list.

In this sample, only the LDAP query is different:

```
"QueryAndMoveMBto (ADSI) .vbs" "&
    (! (cn=sys*))
    (objectCategory=user)
    (homeMDB=CN=Mailbox Store A,
     CN=First Storage Group,
     CN=InformationStore,
     CN=MyOtherExchangeServer,
     CN=Servers,
     CN=First Administrative Group,
     CN=Administrative Groups,
     CN=First Organization,
     CN=Microsoft Exchange,
     CN=Services,
     CN=Configuration,
     DC=VirtualCompany,
     DC=com)
    )"
"Mailbox Store B"
"First Storage Group"
"MyServer"
"First Administrative Group"
```

Exchange 2000 Server activity watcher

The Exchange 2000 server activity watcher is based on the WMI infrastructure (See Part 1, “An overview of the Exchange 2000 WMI Providers” on page 95 for more information about WMI and Exchange 2000). In its actual implementation, the monitoring feature included in the Exchange 2000 ESM provides only the status of the component monitored. The complete information given by the WMI classes is not available through the ESM. This limitation comes mainly from the fact that the Routing Table is used to carry the information status. Due to the amount of space required by the data available from WMI and the available space inside the routing table, it is impossible to carry all the information. The routing table only carries the status.

The next script “E2KWatch” will retrieve the complete set of information available from WMI and sends the result by email (See Figure 2 on page 36). The script will make an intensive usage of the WMI infrastructure of Windows 2000 and Exchange 2000. The monitoring will be based on WMI asynchronous events.

To give a basic sample of a WMI asynchronous event watching any Windows 2000 service modification, look at Sample 3 below.

Sample 3 A basic WMI Asynchronous event handler for the Win32_Service class

```

1:' VB Script creating an asynchronous event notification and looping permanently. '
2:' When a change event occurs on a service an event notification is triggered. '
3:' '
4:' Version 1.00 - Alain Lissoir '
5:' Compaq Computer Corporation - Professional Services - Belgium - '
6:' '
7:' Any comments or questions: ' EMail:alain.lissoir@compaq.com '
8:
9:Option Explicit
..:
..:
14:Set objSink = WScript.CreateObject ("WbemScripting.SWbemSink","SINK_")
15:
16:Set objService = GetObject("WinMgmts:{impersonationLevel=impersonate, (security)}")
17:
18:objService.ExecNotificationQueryAsync objSink, _
19:    "Select * FROM __InstanceModificationEvent WITHIN 1 Where " & _
20:    "TargetInstance isa 'Win32_Service'"
21:
22:WScript.Echo "Waiting for events..."
23:
24:Do
25:
26:    WScript.Sleep (5000)
27:
28:Loop
29:
30:objSink.Cancel
31:
..:
37:' -----
38:Sub SINK_OnObjectReady (objWbemObject, objWbemAsyncContext)
39:
40:    WScript.Echo FormatDateTime(Date, vbLongDate) & " at " & _
41:        FormatDateTime(Time, vbShortTime) & ": " & _
42:        objWbemObject.TargetInstance.DisplayName & " " & _
43:        objWbemObject.TargetInstance.State & _
44:        " (" & objWbemObject.TargetInstance.Name & "). " & _
45:        "Startup mode is '" & objWbemObject.TargetInstance.StartMode & "'."
46:
47:End Sub

```

Each change made to any Windows 2000 services (startup mode, logon credentials, stop, start of the service, etc), will generate a WMI event to the *SINK_OnObjectReady()* function. The two parameters available from this function are two objects initialized by the WMI infrastructure:

- *objWbemObject* contains an object representing the component (Windows 2000 Services, logical disks, CPU, Event Log, etc.) concerned by the event.
- *objWbemAsyncContext* contains a helper object to determine the context of the WMI event. During the initialization of the WMI asynchronous event, it is possible to specify a context (which is a string or a value parameter). This parameter is passed back during the event via this object.

The “E2Kwatch” script uses exactly the same structure and the same logic as Sample 3 on page 13. As we can see, this sample can be divided into two important sections:

- The WMI initialization (lines 14 to 20)
- The WMI Event handling (lines 38 to 47)

For the “E2Kwatch” script, this logic will be applied to many different WMI classes.

Note: You should have a good WMI understanding for the next sections. For more information about WMI and its usage, please refer to the WMI SDK available from http://msdn.microsoft.com/library/psdk/wmisdk/wmistart_5kth.htm

The script can monitor many aspects of an Exchange 2000 server. These aspects can be classified in two categories:

- Windows 2000 and the associated providers
- Exchange 2000 and the associated providers

Windows 2000 level

With the help of the Windows 2000 standard WMI classes, it is possible to monitor:

Windows 2000 services:	Any service modification in the system can generate a WMI asynchronous event. (See Sample 3 on page 13). This monitoring will be used to watch the Exchange 2000 service status. If a service is stopped, the script will be notified and take the necessary actions to restart the service before sending an email notification. The script will use the <i>Win32_Service</i> WMI class.
Disk space requirements:	When running Exchange (like any other server) a basic rule is to guarantee a minimum free disk space in the system. In some case, a best practice is trying to keep the free disk space equal or superior to the size of the Store file located on that disk. This makes repairs and integrity check operations easier facility in case of a problem. The WMI infrastructure will be programmed by the script to trigger an event when a free disk space threshold is reached. The script will use the <i>Win32_LogicalDisk</i> WMI class.

Process CPU usage:	If some process use the CPU time more than a certain threshold, the WMI infrastructure will be programmed to trigger an event captured by the script. The script will use a new class created by the compilation of a Management Object File (.MOF) to get access to the process counters.
Processor CPU activity:	If the CPU utilization is higher than a certain threshold, the WMI infrastructure will be programmed to trigger an event captured by the script. This event can be completely different from the Process CPU usage because two processes can use 40% of the CPU time, which is acceptable in some circumstances, but it will result in 80% of CPU time utilization. The script will use the <i>Win32_Processor</i> WMI class.
Store size:	Based on the size of the .EDB file and the .STM file, the WMI infrastructure will be programmed by the script to trigger an alert when a certain file size is reached. If the size reaches a second (higher) limit, the corresponding Store will be dismounted (via CDOEXM). The script will use the <i>CIM_DATAFile</i> WMI class.
NT Event log:	WMI will detect any particular event created in the event log. This will trigger an alert captured by the script. The script will use the <i>Win32_NTLogEvent</i> WMI class.

Exchange 2000 level

Server Status:	The server status is the status list provided by the WMI <i>ExchangeServerState</i> class (See Part 1, Table 8). The script will initiate an asynchronous event for any properties change.
Connector status between servers:	The connector status is the status provided by the WMI <i>ExchangeConnectorState</i> class (See Part 1, Table 9). The script will initiate an asynchronous event for any properties change.
Link and Queue <i>increasingTime</i> :	The script will program a WMI asynchronous event for any <i>increasingTime</i> value higher than the prior <i>increasingTime</i> value. WMI will not trigger an event for other Link/Queue property change (even if the <i>increasingTime</i> is going to zero). This test will act as a filter. Next, if the <i>increasingTime</i> value reaches a certain threshold, the script will send an alert. The script will use the <i>ExchangeLink</i> WMI class and the <i>ExchangeQueue</i> WMI class (See Part 1, Table 10 and Table 11).

E2Kwatch.Wsf script

The “E2KWatch” script contains more than 2000 lines of code. This script reproduces the same logic used in Sample 3 on page 13. Due to the number of lines, it is not possible to publish and comment on the complete script listing. The next sections will concentrate on the key parts of the script. The script makes an extensive use of the WMI infrastructure and uses many Windows 2000 and Exchange 2000 WMI classes.

The configuration file

Because it has such a large number of parameters, the script uses a configuration file. This file holds all the parameters related to the monitoring of a particular Exchange 2000 Server. The script must run locally on the Exchange server.

The configuration file corresponds to the server DC01-CPQ visible in Part 1, Figure 6 with ESM. To get an idea of the server configuration, look at the output file in Part 2, Figure 7.

The configuration file is read by the *ReadConfigurationFile()* function part of the “E2Kwatch” script. We will not enter in the details of this function. The purpose is to look at the WMI and CDOEXM aspects to ease the Exchange 2000 Server management understanding. The Figure 1 below shows a sample of the configuration file used for the server DC01-CPQ.

Figure 1: The E2KWatch configuration file

```

1:# E2KWatch Configuration file. #
2:# #
3:# Version 1.00 - Alain Lissoir #
4:# Compaq Computer Corporation - Professional Services - Belgium #
5:# #
6:# Any comments or questions: EMail:alain.lissoir@compaq.com #
7:
8:# Internet Email address where to send the mail alerts.
9:# (By default use the current logged user with the current domain.
10: MAILALERTTO=%USERNAME%@%USERDNSDOMAIN%
11:
12:# Internet Email address where to send the mail alerts.
13:# (By default use the current logged user with the current domain.
14: MAILERRORTO=%USERNAME%@%USERDNSDOMAIN%
15:
16:# Microsoft Exchange System Attendant
17: Service=MSExchangeSA
18:# Manages Microsoft Exchange Information Storage
19: Service=MSExchangeIS
20:# Microsoft Exchange POP3
21: Service=POP3Svc
22:# Microsoft Exchange Routing Engine
23: Service=RESvc
24:# Microsoft Exchange IMAP4
25: Service=IMAP4Svc
26:# Microsoft Exchange MTA Stacks
27: Service=MSExchangeMTA
28:# Microsoft Exchange Event
29: Service=MSExchangeES
30:# Microsoft Exchange Site Replication Service
31: Service=MSExchangeSRS
32:# Simple Mail Transport Protocol (SMTP)
33: Service=SMTPSVC
34:# Microsoft Search
35: Service=MSSEARCH
36:# SNMP Service
37: Service=SNMP
38:
39:# Processor #1
40: Processor=CPU0; 80
41:# Processor #1
42: Processor=CPU1; 80
43:# Processor #2
44: Processor=CPU2; 80
45:# Processor #3
46: Processor=CPU3; 80

```

```

47:
48:# Logical Disk
49: LogicalDisk=C:; 50
50:# Logical Disk
51: LogicalDisk=D:; 50
52:# Logical Disk
53: LogicalDisk=E:; 50
54:
55:# Information Store maximum sizes
56: MDBSTORESIZE=CN=Mailbox Store A,CN=First Storage Group,CN=InformationStore,CN=DC01-CPQ,CN=Servers,CN=First Administrative Group,CN=Administrative Groups,CN=First Organization,CN=Microsoft Exchange,CN=Services,CN=Configuration,DC=VirtualCompany,DC=com; 51200; 81920
57: STMSTORESIZE=CN=Mailbox Store A,CN=First Storage Group,CN=InformationStore,CN=DC01-CPQ,CN=Servers,CN=First Administrative Group,CN=Administrative Groups,CN=First Organization,CN=Microsoft Exchange,CN=Services,CN=Configuration,DC=VirtualCompany,DC=com; 51200; 81920
..:
..:
..:
..:
83: MDBSTORESIZE=CN=Public Folder Store E,CN=First Storage Group,CN=InformationStore,CN=DC01-CPQ,CN=Servers,CN=First Administrative Group,CN=Administrative Groups,CN=First Organization,CN=Microsoft Exchange,CN=Services,CN=Configuration,DC=VirtualCompany,DC=com; 51200; 81920
84: STMSTORESIZE=CN=Public Folder Store E,CN=First Storage Group,CN=InformationStore,CN=DC01-CPQ,CN=Servers,CN=First Administrative Group,CN=Administrative Groups,CN=First Organization,CN=Microsoft Exchange,CN=Services,CN=Configuration,DC=VirtualCompany,DC=com; 51200; 81920
85:
86:# EventLog to capture
87:# EventLog=
88:# EventLog= MSEExchangeSA; 9175; MAPI Session; *; *
89:# EventLog= MSEExchangeSA; 9175; MAPI Session; Application; error
90: EventLog= MSEExchangeSA; *; *; *; *
91: EventLog= MSEExchangeMU; *; *; *; *
92: EventLog= MSEExchangeIS Public Store; *; *; *; *
93: EventLog=MSEExchangeIS Mailbox Store; *; *; *; *
94: EventLog= MSEExchangeIS; *; *; *; *
95: EventLog= MSEExchangeDSAccess; *; *; *; *
96: EventLog= MSEExchangeFBPublic; *; *; *; *
97:
98:# Exchange System Attendant process
99: Process=MAD; 50
100:# Exchange Information Store process
101: Process=STORE; 50
102:# Exchange Message Transfer Agent process
103: Process=EMSMTA; 50
104:# Any other process using more than 80% of the CPU time
105:# Process=*; 80
106:
107:# Watch the ExchangeRoutingTable WMI Provider (Class ExchangeServerState)
108: WMIEXCHANGESERVER=%COMPUTERNAME%
109:
110:# Watch the ExchangeRoutingTable WMI Provider (Class ExchangeConnectorState)
111: WMIEXCHANGECONNECTOR=To First Routing Group
112: WMIEXCHANGECONNECTOR=To Second Routing Group
113:
114:# Watch the ExchangeQueue WMI Provider (Class ExchangeLink)
115: WMIEXCHANGELINK=*; 30
116:
117:# Watch the ExchangeQueue WMI Provider (Class ExchangeQueue)
118: WMIEXCHANGEQUEUE=*; 30
119:
120:# Watch the ExchangeCluster WMI Provider (Class ExchangeClusterResource)
121:# WMIEXCHANGECLUSTER=VirtualServerName

```

This configuration contains all the parameters to define the monitoring:

- Lines 16 to 37 contain the Windows 2000 service list to be monitored. Note that the service name given is the registry key name of the service, not the display name of the service.
- Lines 39 to 40 contain the maximum CPU utilization threshold for each process present in the system.

- Lines 48 to 53 contain the minimum free disk space threshold percentage.
- Lines 56 to 84 contain the *distinguishedName* of the Store to monitor. Each keyword concerns a specific file of the Store: The “MDBSTORESIZE” keyword for the EDB file and the “STMSTORESIZE” keyword for the .STM file. If the Store file (.EDB or .STM) reaches the size given in the second parameter (after the semi-colon), WMI will trigger an event. If the size is equal or higher than the third parameter, the script will take action with CDOEXM to dismount the store.
- Lines 87 to 96 contain the EventLog for which WMI must trigger an alert. If a wildcard is used, it means “any” and the corresponding value will not be taken in consideration to filter the event.
- Lines 98 to 105 contain the process list to monitor. If the CPU process utilization is higher or equal than the specified threshold for a given process, WMI will trigger an event.

Lines 107 to 121 reference the Exchange 2000 WMI providers:

- Line 108 specifies the Exchange Server name for the *ExchangeServerState* instance that must change to trigger a WMI event.
- Lines 111 and 112 specify the Exchange connector name for the *ExchangeConnectorState* instance that must change to trigger a WMI event.
- The lines 114 to 118 specify the *increasingTime* threshold which will trigger a WMI event (See Part 1, APPENDIX “Exchange 2000 WMI Queue Provider”). Note the presence of the wildcard to attach this statement to any link and any queue present in the system. Of course, a specific link or queue name can be given.

The WMI initialization

Like Sample 3 on page 13 (lines 14 to 20), “E2Kwatch.Wsf” must register the WMI class events based on the parameters given in the configuration file. The next sections are a quick review of the WMI asynchronous event initialization for each class used in “E2KWatch.Wsf”.

WMI *Win32_Process* class

The service list to monitor is stored in an array initialized by the *ReadConfigurationFile()* function. To determine if there are some services to monitor, the script checks the array size (Sample 4 on page 19, line 360). If it is greater than 0, it means that the array contains at least one service to monitor. All the initializations procedures use the same tactic.

Lines 361 and 362 create an object containing the data required to associate the event handler routine label (“Win32_ServiceSINK_”) to the desired class (*Win32_Service*).

Lines 363 to 366 execute the WMI registration. There are four things to notice in these lines:

- The data selection statement “Select *” to define the data to be retrieved from the class. In this case, the script will retrieve all the data available from the WMI class (line 364).
- The *__InstanceModificationEvent* statement to explicitly ask to WMI to send a notification each time a modification to the specified class is made (line 364).

- The presence of the “WITHIN” statement (line 364) with the constant *cWMI_Within_Win32_Service* (line 365) to specify a polling interval when the consumer (The “E2KWatch.Wsf” script) requires change notifications of a class. This “WITHIN” statement is mandatory because the *Win32_Service* WMI class does not have an event provider. The constant *cWMI_Within_Win32_Service* is defined in the very beginning of the script. By using the WITHIN statement, the script specifies a polling frequency for the class. This polling frequency can be very important in some situations. For instance, the calculation of the *increasingTime* property from the *ExchangeLink* or *ExchangeQueue* WMI classes can be influenced by the polling frequency (See Part 1, APPENDIX for more information about the *increasingTime* value).
- The condition statement “TargetInstance isa 'Win32_Service'” (line 366) defines the class to be monitored.

Most of the samples given for the initialization process use the same type of statement. The only exception is for the *Win32_NTLogEvent*. See Sample 7 on page 20, for more information.

The major coding change between the different samples will be the condition statement. In Sample 4 below, the statement only concerns the class itself. In this case, it means that any changes to instances of the specified class (*Win32_Service*) will trigger an event.

Sample 4 The WMI *Win32_Service* class asynchronous event registration

```

...:
...:
359:     ' Watch the 'Win32_Service'
360:     If UBound (strServiceName) Then
361:         Set objWin32_ServiceSink = WScript.CreateObject ("WbemScripting.SWbemSink", _
362:             "Win32_ServiceSINK_")
363:         objWMIService.ExecNotificationQueryAsync objWin32_ServiceSink, _
364:             "Select * FROM __InstanceModificationEvent WITHIN " & _
365:             cWMI_Within_Win32_Service & " Where " & _
366:             "TargetInstance isa 'Win32_Service'"
...:
370:     End If
...:
...:

```

Of course, it is possible to specify the service status in the condition statement but, in the case of “E2KWatch”, this will be made in the event handler routine. This choice is only made for educational reasons. Each time a change to a service is made, even if the service is not mentioned in the configuration file, an event will be triggered by WMI. The event handler, based on the parameters given in the configuration file will filter the event to determine the actions to take. The same concept will be used for other event registrations and other event handlers.

This means that the filtering is made at the script level. For instance, if the event filtering should be made by WMI line 366 must be changed to:

```

...:
...:
363:         objWMIService.ExecNotificationQueryAsync objWin32_ServiceSink, _
364:             "Select * FROM __InstanceModificationEvent WITHIN " & _
365:             cWMI_Within_Win32_Service & " Where " & _
366:             "TargetInstance isa 'Win32_Service' And TargetInstance.State='Stopped'"
...:
...:

```

In this case, a WMI event will be triggered only if a service state is modified to a “stopped” state. The filtering is completed by WMI and not by the script.

Many other combinations can be used, such as the service name, the start mode, etc ... Please refer to the WMI SDK available from http://msdn.microsoft.com/library/psdk/wmisdk/wmistart_5kth.htm for more information about what classes are available and their associated properties.

WMI *Win32_Processor* class

The *Win32_Processor* class uses the exact same piece of code as the *Win32_Service* class. All the remarks made for *Win32_Service* are valid. Of course, the class is different, so, the properties retrieved will be different. See the WMI SDK for more information.

Sample 5 The WMI *Win32_Processor* class asynchronous event registration

```

...:
...:
372:     ' Watch the 'Win32_Processor'
373:     If UBound (strCPUDeviceID) Then
374:         Set objWin32_ProcessorSink = WScript.CreateObject ("WbemScripting.SWbemSink", _
375:             "Win32_ProcessorSINK_")
376:         objWMIService.ExecNotificationQueryAsync objWin32_ProcessorSink, _
377:             "Select * FROM __InstanceModificationEvent WITHIN " & _
378:             cWMI_Within_Win32_Processor & " Where " & _
379:             "TargetInstance isa 'Win32_Processor'"
...:
383:     End If
...:
...:

```

WMI *Win32_LogicalDisk* class

The *Win32_LogicalDisk* class uses the exact same piece of code as the *Win32_Service* class (See Sample 4 on page 19). All the remarks made before are valid.

Note the extension of the condition statement (“TargetInstance.DriveType=3”) to receive WMI events only for hard disks (Line 392).

Sample 6 The WMI *Win32_LogicalDisk* class asynchronous event registration

```

...:
...:
385:     ' Watch the 'Win32_LogicalDisk'
386:     If UBound (strLogicalDiskName) Then
387:         Set objWin32_LogicalDiskSink = WScript.CreateObject ("WbemScripting.SWbemSink", _
388:             "Win32_LogicalDiskSINK_")
389:         objWMIService.ExecNotificationQueryAsync objWin32_LogicalDiskSink, _
390:             "Select * FROM __InstanceModificationEvent WITHIN " & _
391:             cWMI_Within_Win32_LogicalDisk & " Where " & _
392:             "TargetInstance isa 'Win32_LogicalDisk' and TargetInstance.DriveType=3"
...:
396:     End If
...:
...:

```

WMI *Win32_NTLogEvent* class

As mentioned before, the *Win32_NTLogEvent* is a particular case and requires two changes:

- The `__InstanceCreationEvent` statement instead of the `__InstanceModificationEvent` statement (line 403) to explicitly ask WMI to send a notification each time a creation event to the specified class is made. This corresponds to the a new event being logged in the Windows 2000 application event log.
- The absence of the “WITHIN” statement because the *Win32_NTLogEvent* WMI class has an event provider.

Except these two changes, the logic is the same as before.

Sample 7 The WMI *Win32_NTLogEvent* class asynchronous event registration

```

...:
...:
398:     ' Watch the 'Win32_NTLogEvent'

```

```

399:         If UBound (strEventLogSourceName) Then
400:             Set objWin32_NTLogEventSink = WScript.CreateObject ("WbemScripting.SWbemSink", _
401:                 "Win32_NTLogEventSINK_")
402:             objWMIService.ExecNotificationQueryAsync objWin32_NTLogEventSink, _
403:                 "Select * FROM __InstanceCreationEvent Where " & _
404:                 "TargetInstance isa 'Win32_NTLogEvent'"
405:             ...
408:         End If
409:         ...
410:         ...

```

WMI CIM_DATAFile class

This WMI initialization is more complex than the prior one, but again the logic applied is the same. The difference comes from the condition statement. The CIM_DATAFile class is used to monitor the size of the .EDB and .STM Store files. The script will program a WMI event for each Store file defined in the configuration file (See Figure 1 on page 16, lines 56 to 84).

The condition is composed of different elements:

- The filename for which the WMI event must occur (See Sample 8 below, lines 419 and 420):

```
TargetInstance.Name=" " & ReplaceBy(strDBPath (intIndice), "\", "\\")
```

At the same time, the function *ReplaceBy()* is called to replace in the Store file path any single backslash by a double backslash. This is a requirement for the file path syntax.

- The event must occur only if the file size is greater than the size specified in the configuration file (lines 421 and 422):

```
"TargetInstance.FileSize > " & 1024 * CLng(intStoreAlertSize (intIndice))
```

The file size limit is expressed in Megabytes in the configuration file. This explains the “* 1024” statement.

- The event must occur only if the file grows. This is the reason for the statement (line 423):

```
TargetInstance.FileSize > PreviousInstance.FileSize
```

The *PreviousInstance* retrieves the characteristics of the object before the modification. With this statement, the condition will be true if the current file size is greater than the prior file size.

Sample 8 The WMI CIM_DATAFile class asynchronous event registration

```

...
...
410:         ' Watch the 'CIM_DATAFile'
411:         If UBound (strDNStoreDB) Then
412:             Set objCIM_DATAFileSink = WScript.CreateObject ("WbemScripting.SWbemSink", _
413:                 "E2K_StoreDBSink_")
414:             ' Register Async event for .EDB file
415:             For intIndice = 1 to UBound (strDNStoreDB)
416:                 objWMIService.ExecNotificationQueryAsync objCIM_DATAFileSink, _
417:                     "SELECT * From __InstanceModificationEvent Within " & _
418:                     cWMI_Within_CIM_DataFile & " Where " & _
419:                     "TargetInstance isa 'CIM_DATAFile' And TargetInstance.Name=" & _
420:                     ReplaceBy(strDBPath (intIndice), "\", "\\") & "' And " & _
421:                     "TargetInstance.FileSize > " & _
422:                     1024 * CLng(intStoreAlertSize (intIndice)) & _
423:                     " And TargetInstance.FileSize > PreviousInstance.FileSize"
424:                 ...
429:             Next
430:         End If
431:         ...
432:         ...

```

WMI *NTProcess* class

The *NTProcess* class is not a standard WMI class. This class is created with the help of .Mof (Managed Object Format) file. See the WMI SDK for more information about MOF files usage.

The logic usage is exactly the same as before. Because WMI must trigger an event when a process uses more than a certain threshold of CPU time, the registration excludes the *Idle* and *Total processes* from the selection. One represents the total free CPU time. The other represents the total CPU time used by all the processes.

Sample 9 The WMI *NTProcess* class asynchronous event registration

```

...:
...:
447:     If UBound (strProcessName) Then
448:         Set objNTProcessSink = WScript.CreateObject ("WbemScripting.SWbemSink", _
449:             "NTProcessSink_")
450:         objWMINTProcessService.ExecNotificationQueryAsync objNTProcessSink, _
451:             "Select * FROM __InstanceModificationEvent WITHIN " & _
452:             cWMI_Within_NTProcess & " Where " & _
453:             "TargetInstance isa 'NTProcess' and " & _
454:             "TargetInstance.Process != 'Idle' and " & _
455:             "TargetInstance.Process != '_Total'"
456:         intRC = WriteToFile (objLogFileName, "WMIEventRegistration 'NTProcess' " & _
457:             "asynchronous events registered.")
458:     End If
...:
...:

```

WMI *ExchangeServerState* class

This event registration uses the same logic as the *Win32_Service* class event registration. No change is made to the logic except the class itself. Any modification on this class will trigger an event. See Part 1, APPENDIX for more information about this Exchange WMI class.

Sample 10 The WMI *ExchangeServerState* class asynchronous event registration

```

...:
...:
478:     If UBound (strWMIExchangeServerName) Then
479:         ' Watch the 'ExchangeServerState'
480:         Set objE2K_ServerStateSink = WScript.CreateObject ("WbemScripting.SWbemSink", _
481:             "E2K_ServerStateSink_")
482:         objWMIExchangeService.ExecNotificationQueryAsync objE2K_ServerStateSink, _
483:             "Select * FROM __InstanceModificationEvent WITHIN " & _
484:             cWMI_Within_E2K_ServerState & " Where " & _
485:             "TargetInstance isa 'ExchangeServerState'"
...:
488:     End If
...:
...:

```

WMI *ExchangeConnectorState* class

This event registration uses the same logic as the *ExchangeServerState* class event registration. No change is made to the logic except the class itself. Any modification on this class will trigger an event. See Part 1, APPENDIX for more information about this Exchange WMI class.

Sample 11 The WMI *ExchangeConnectorState* class asynchronous event registration

```

...:
...:
490:     If UBound (strWMIExchangeConnector) Then
491:         ' Watch the 'ExchangeConnectorState'
492:         Set objE2K_ConnectorStateSink = WScript.CreateObject ("WbemScripting.SWbemSink",
493:             "E2K_ConnectorStateSink_")
494:         objWMIExchangeService.ExecNotificationQueryAsync objE2K_ConnectorStateSink, _
495:             "Select * FROM __InstanceModificationEvent WITHIN " & _
496:             cWMI_Within_E2K_Connector & " Where " & _
497:             "TargetInstance isa 'ExchangeConnectorState'"
...:

```

```

500:         End If
...:
...:

```

WMI *ExchangeLink* class

This event registration uses the same logic as the *ExchangeServerState* class event registration. The particularity is the inclusion in the condition statement of:

```
TargetInstance.IncreasingTime > PreviousInstance.IncreasingTime
```

The idea is the same as the *CIM_DATAFile* class. WMI must only trigger an event if the *increasingTime* value increases. See APPENDIX for more information about this Exchange WMI class. To have more information about the usage of the *increasingTime* value, refer to the same section.

Sample 12 The WMI *ExchangeLink* class asynchronous event registration

```

...:
...:
502:         If UBound (strWMIExchangeLink) Then
503:             ' Watch the 'ExchangeLink'
504:             Set objE2K_LinkSink = WScript.CreateObject ("WbemScripting.SWbemSink", _
505:                                                         "E2K_LinkSink_")
506:             objWMIExchangeService.ExecNotificationQueryAsync objE2K_LinkSink, _
507:                 "Select * FROM __InstanceModificationEvent WITHIN " & _
508:                 cWMI_Within_E2K_Link & " Where " & _
509:                 "TargetInstance isa 'ExchangeLink' And " & _
510:                 "TargetInstance.IncreasingTime > PreviousInstance.IncreasingTime"
...:
513:         End If
...:
...:

```

WMI *ExchangeQueue* class

This event registration uses the same logic as the *ExchangeLink* class event registration. Refer to the precedent paragraph for more information.

Sample 13 The WMI *ExchangeQueue* class asynchronous event registration

```

...:
...:
515:         If UBound (strWMIExchangeQueue) Then
516:             ' Watch the 'ExchangeQueue'
517:             Set objE2K_QueueSink = WScript.CreateObject ("WbemScripting.SWbemSink", _
518:                                                         "E2K_QueueSink_")
519:             objWMIExchangeService.ExecNotificationQueryAsync objE2K_QueueSink, _
520:                 "Select * FROM __InstanceModificationEvent WITHIN " & _
521:                 cWMI_Within_E2K_Queue & " Where " & _
522:                 "TargetInstance isa 'ExchangeQueue' And " & _
523:                 "TargetInstance.IncreasingTime > PreviousInstance.IncreasingTime"
...:
526:         End If
...:
...:

```

WMI *ExchangeClusterResource* class

This event registration uses the same logic as the *ExchangeServerState* class event registration. For any change to the *ExchangeClusterResource* WMI class, WMI will trigger an event. See Part 1, APPENDIX for more information about this Exchange WMI class.

Sample 14 The WMI *ExchangeClusterResource* class asynchronous event registration

```

...:
...:
528:         If UBound (strWMIExchangeClusterVName) Then
529:             ' Watch the 'ExchangeClusterResource'
530:             Set objE2K_ClusterResourceSink = WScript.CreateObject ("WbemScripting.SWbemSink", _
531:                                                         "E2K_ClusterResourceSink_")
532:             objWMIExchangeService.ExecNotificationQueryAsync objE2K_ClusterResourceSink, _
533:                 "Select * FROM __InstanceModificationEvent WITHIN " & _
534:                 cWMI_Within_E2K_ClusterResource & " Where " & _

```

```

535:           "TargetInstance isa 'ExchangeClusterResource'"
536:           intrc = WriteToFile (objLogFileName, "(WMIEventRegistration) " & _
537:                               "'ExchangeClusterResource' asynchronous events registered.")
538:           End If
...:
...:

```

The event routines

The event routine is the function called by the WMI infrastructure when the registered asynchronous event occurs. The function has two parts:

- A first part is defined during the event sink object creation and used by the WMI asynchronous event registration (See “The WMI initialization” on page 18).
- A second part is defined by the WMI infrastructure and corresponds to the event triggered by WMI. In this case, the “OnObjectReady” corresponds to an object available and provided by an asynchronous call.

The object is considered as available based on the condition statements given during the asynchronous event registration.

As explained for Sample 3 on page 13, two parameters are available inside an asynchronous event function: *objWbemObject* and *objWbemAsyncContext*.

The “E2KWatch.Wsf” script will make use of *objWbemObject* to retrieve the properties of the object for the asynchronous event.

WMI Win32_Process class

Because the configuration file may contain more than one service to monitor, the script is executing a “For Each” loop (Sample 15 on page 25, lines 679 to 706) to retrieve the matching service name from the list with the one provided by WMI.

Note: There are many different ways to program the WMI events. In this case, the script is executing a filtering of the event once the event is triggered (inside the event handler) because the WMI condition set during the WMI asynchronous event initialization is not restrictive (See Sample 4 on page 19).

As mentioned, another method is to program a more restrictive condition during the initialization. For instance, if this concerns a Windows 2000 service event registration, WMI will trigger an event only for the service matching the condition set (i.e. the services listed in the configuration file with a specific service state for which the event must be triggered). This approach is best from a performance point of view but it will require the registration of a WMI event per service to watch. This makes the registration process more complex. From an educational point of view, the first method is better because it allows much more events to occur related to any services. The WMI event handler in the script will make the filtering. This eases the understanding. This is the reason for this choice.

Once a service match is found (line 681), the script processes of the event. The script uses the *WMI_GetSvcStatus()* function to determine the state of the service. This function parses a property of the *objWbemObject* object (representing the service itself) and returns false if the service is stopped.

In such a case, the script calls the function *WMI_LoopStartServiceRetry()*. This function makes a loop to restart the service. Three attempts are made (this is a default value defined in the script). When the script exits from this function, the returned value is true if during the loop the service has been restarted. Otherwise, after the maximum number of attempts, the returned value is false.

Sample 15 The WMI *Win32_Service* class asynchronous event handler

```

...:
...:
663:' -----
664:' Win32_ServiceSink
665:' -----
666:Sub Win32_ServiceSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
667:
...:
677:     boolSvcFound = False
678:
679:     For intIndice = 1 To Ubound(strServiceName)
680:
681:         If Ucase(strServiceName(intIndice)) = Ucase(objWbemObject.TargetInstance.Name) Then
682:             boolSvcFound = True
683:
684:             ' Get the status of the service
685:             boolSvcStatus = WMI_GetSvcStatus (objLogFileName, objWbemObject.TargetInstance)
686:
687:             ' Service is running, boolSvcStatus result is True
688:             If boolSvcStatus Then
689:                 If strServiceRetryCounter(intIndice) > 0 Then
694:                     Win32_ServiceInfo objLogFileName, _
695:                                     objWbemObject.TargetInstance, _
696:                                     objWbemObject.PreviousInstance
697:
698:                 End If
699:                 strServiceRetryCounter(intIndice) = 0
700:             Else
701:                 boolSvcStatus = WMI_LoopStartServiceRetry (objLogFileName, _
702:                                                         intIndice, _
703:                                                         objWbemObject.TargetInstance, _
704:                                                         boolSvcStatus)
705:             End If
706:         End If
707:     Next
708:
709:     If boolSvcFound = False Then
710:
711:         intrc = WriteToFile(objLogFileName, _
712:                             "(Win32_ServiceSink_OnObjectReady) No action taken, " & _
713:                             "service is not on the list.")
714:
715:     Else
716:         ' Service is not running, boolSvcStatus result is False
717:         If Not boolSvcStatus Then
718:             Win32_ServiceInfo objLogFileName, _
719:                             objWbemObject.TargetInstance, _
720:                             objWbemObject.PreviousInstance
721:
722:             End If
723:         End If
724:
725:     intrc = WriteToFile (objLogFileName, _
726:                         "-----")
727:
728: End Sub
729:
...:
...:

```

If the service is not running, the information related to its current status is retrieved (line 719). All the functions in the form *ClassNameInfo()*, such as the function *Win32_ServiceInfo()* are doing the same thing:

- Retrieving the current information related to the class
- Formating the data in HTML
- Sending a mail to the address given in the configuration file (MAILALERTTO parameter in Figure 1 on page 16, line 10) with the *SendMessage()* function.

Next the event terminates.

WMI *Win32_Processor* class

The *Win32_Processor* event handler uses the same logic as the *Win32_Service* event handler to retrieve the matches based on the content of the configuration file. The difference is the action taken. If the processor name exceeds (Sample 16 below, lines 745 and 746) an associated maximum threshold (lines 747 and 748), the function *Win32_ProcessorInfo()* is invoked to format the related data (lines 749 to 751) and send a mail.

Sample 16 The WMI *Win32_Processor* class asynchronous event handler

```

...:
...:
732:' -----
733:' Win32_ProcessorSink
734:' -----
735:Sub Win32_ProcessorSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
...:
741:    intRC = WriteToFile (objLogFileName, _
742:                        "(Win32_ProcessorSink_OnObjectReady) Start Sink.")
743:
744:    For intIndice = 1 To Ubound(strCPUDeviceID)
745:        If Ucase(strCPUDeviceID(intIndice)) = _
746:            Ucase(objWbemObject.TargetInstance.DeviceID) And _
747:            Clng(intCPULoadPercentageMax(intIndice)) <= _
748:            Clng(objWbemObject.TargetInstance.LoadPercentage) Then
749:                Win32_ProcessorInfo objLogFilename, _
750:                                objWbemObject.TargetInstance, _
751:                                objWbemObject.PreviousInstance
752:            Exit For
753:        End If
754:    Next
755:
756:    intRC = WriteToFile (objLogFileName, _
757:                        "(Win32_ProcessorSink_OnObjectReady) End Sink.")
758:    intRC = WriteToFile (objLogFileName, _
759:                        "-----")
760:
761:End Sub
...:
...:

```

WMI *Win32_LogicalDisk* class

Again, the *Win32_LogicalDisk* event handler uses the same logic as the *Win32_Service* event handler to retrieve the matches based on the content of the configuration file. The modification resides in two different things:

- First, the percentage of free disk space is calculated (Sample 17 below, lines 776 and 777) before the logical disk name filtering because the WMI class does not provide this value in its properties.
- Next, during the loop (lines 779 to 789), if the logical disk name exceeds (lines 780 and 781) a maximum threshold (lines 747 and 748), the function *Win32_LogicalDiskInfo()* is invoked to format the related data (lines 749 to 751) and send a mail.

The important point to note is that WMI does not make any data calculation or data correlation. The event handler handles this task (lines 776 and 777).

Sample 17 The WMI *Win32_LogicalDisk* class asynchronous event handler

```

...:
...:
763:' -----
764:' Win32_LogicalDiskSink
765:' -----
766:Sub Win32_LogicalDiskSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
...:
773:    intRC = WriteToFile (objLogFileName, _
774:                        "(Win32_LogicalDiskSink_OnObjectReady) Start Sink.")
775:

```

```

776:         intPercentageFree = 100 * _
777:             (objWbemObject.TargetInstance.FreeSpace / objWbemObject.TargetInstance.Size)
778:
779:     For intIndice = 1 To Ubound(strLogicalDiskName)
780:         If Ucase(strLogicalDiskName(intIndice)) = _
781:             Ucase(objWbemObject.TargetInstance.Name) And _
782:             CLng(intLogicalDiskFreeSpaceMin(intIndice)) >= _
783:                 CLng(intPercentageFree) Then
784:             Win32_LogicalDiskInfo objLogFileName, _
785:                 objWbemObject.TargetInstance, _
786:                 objWbemObject.PreviousInstance
787:         Exit For
788:     End If
789: Next
790:
791:     intRC = WriteToFile (objLogFileName, _
792:         "(Win32_LogicalDiskSink_OnObjectReady) End Sink.")
793:     intRC = WriteToFile (objLogFileName, _
794:         "-----")
795:
796:End Sub
...:
...:

```

WMI Win32_NTLogEvent class

The *Win32_NTLogEvent* event handler uses the same logic as other event handlers. The sophisticated part resides in the filtering of the event. The configuration file (Figure 1 on page 16, lines 90 to 96) allows the use of a wildcard. This allows us to determine which event must be considered as important or not by the event handler. The filtering process can be summarized as a pure string manipulation (Sample 18 below, lines 819 to 848).

Once the string comparisons and substitutions are made, the validation tests (lines 850 to 861) are executed to determine if an alert must be sent via the *Win32_NTLogEventInfo()* function (line 860).

Sample 18 The WMI Win32_NTLogEvent class asynchronous event handler

```

...:
...:
798: ' -----
799: ' Win32_NTLogEventSink
800: ' -----
801: Sub Win32_NTLogEventSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
...:
814:     intRC = WriteToFile (objLogFileName, _
815:         "(Win32_NTLogEventSink_OnObjectReady) Start Sink.")
816:
817:     For intIndice = 1 To Ubound(strEventLogSourceName)
818:
819:         If strEventLogSourceName(intIndice) = "*" Then
820:             strTempEventLogSourceName = CheckIfNull(objWbemObject.TargetInstance.SourceName)
821:         Else
822:             strTempEventLogSourceName = strEventLogSourceName(intIndice)
823:         End If
824:
825:         If intEventLogEventCode(intIndice) = "*" Then
826:             intTempEventLogEventCode = CheckIfNull(objWbemObject.TargetInstance.EventCode)
827:         Else
828:             intTempEventLogEventCode = intEventLogEventCode(intIndice)
829:         End If
830:
831:         If strEventLogCategoryString(intIndice) = "*" Then
832:             strTempEventLogCategoryString=CheckIfNull(objWbemObject.TargetInstance.CategoryString)
833:         Else
834:             ' Add a CRLF because original 'objWbemObject' always has a CRLF at the end.
835:             strTempEventLogCategoryString = strEventLogCategoryString(intIndice) & vbCRLF
836:         End If
837:
838:         If strEventLogLogfile(intIndice) = "*" Then
839:             strTempEventLogLogfile = CheckIfNull(objWbemObject.TargetInstance.Logfile)
840:         Else
841:             strTempEventLogLogfile = strEventLogLogfile(intIndice)
842:         End If

```

```

843:
844:     If strEventLogType(intIndice) = "*" Then
845:         strTempEventLogType = CheckIfNull(objWbemObject.TargetInstance.Type)
846:     Else
847:         strTempEventLogType = strEventLogType(intIndice)
848:     End If
849:
850:     If UCase(strTempEventLogSourceName) = _
851:         UCase(CheckIfNull(objWbemObject.TargetInstance.SourceName)) And _
852:         CLng(intTempEventLogEventCode) = _
853:         CLng(CheckIfNull(objWbemObject.TargetInstance.EventCode)) And _
854:         UCase(strTempEventLogCategoryString) = _
855:         UCase(CheckIfNull(objWbemObject.TargetInstance.CategoryString)) And _
856:         UCase(strTempEventLogLogfile) = _
857:         UCase(CheckIfNull(objWbemObject.TargetInstance.Logfile)) And _
858:         UCase(strTempEventLogType) = _
859:         UCase(CheckIfNull(objWbemObject.TargetInstance.Type)) Then
860:         Win32_NTLogEventInfo objLogFileName, objWbemObject.TargetInstance
861:         Exit For
862:     End If
863: Next
864:
865:     intRC = WriteToFile (objLogFileName, _
866:         "(Win32_NTLogEventSink_OnObjectReady) End Sink.")
867:     intRC = WriteToFile (objLogFileName, _
868:         "-----")
869:
870:End Sub
...:
...:

```

WMI *CIM_DATAFile* class

Again, the *CIM_DATAFile* event handler uses the same logic as other event handlers but it takes some more actions. The script is monitoring the size of the specific files that represent the Store. As explained before, two thresholds are specified in the configuration file:

- The first one to send an alert via the *E2K_StoreDBInfo()* function (Sample 19 below, lines 965 to 967)
- The second one (line 888) to automatically dismount (line 954) the store with the function *DismountStoreDB()* when it is reached.

Before dismounting the store, the script is sending some alerts with the WMI information available (lines 889 to 942) and waits for a timeout (line 943). Next the store is dismounted (line 954).

Sample 19 The WMI *CIM_DATAFile* class asynchronous event handler

```

...:
...:
873:' -----
874:' E2K_StoreDBSink
875:' -----
876:Sub E2K_StoreDBSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
...:
882:     intRC = WriteToFile (objLogFileName, _
883:         "(E2K_StoreDBSink_OnObjectReady) Start Sink.")
884:
885:     For intIndice = 1 To Ubound(strDNStoreDB)
886:         If UCase(objWbemObject.TargetInstance.Name) = UCase(strDBPath (intIndice)) Then
887:
888:             If intStoreDismountSize (intIndice) < objWbemObject.TargetInstance.FileSize Then
889:                 strHTML = cStartREDHTML & "WARNING:" & _
890:                     cBacktoNormalHTML & "<BR>Dismounting store " & _
891:                     strStoreName(intIndice) & _
892:                     "' (" & strStoreDBClass(intIndice) & ") in " & _
893:                     cWaitBeforeDBDismount / 1000 & "s. on " & _
894:                     strComputerName & " (E2K_StoreDBInfo)"
895:
896:                 strHTML = cStartHTMLTableTitle & strHTML & cEndHTMLTableTitle
897:                 strHTML = strHTML & cStartHTMLTableData
898:                 strHTML = strHTML & FormatHTML ("", _
899:                     "<B>Current State</B>", _

```

```

900:                                     "<B>Previous State</B>")
901:     strHTML = strHTML & FormatHTML ("Name: ", _
902:                                     objWbemObject.TargetInstance.Name, _
903:                                     objWbemObject.PreviousInstance.Name)
904:     strHTML = strHTML & _
905:         FormatHTML ("Filesize: ", _
906:                     objWbemObject.TargetInstance.FileSize / 1024, _
907:                     objWbemObject.PreviousInstance.FileSize / 1024)
908:     strHTML = strHTML & FormatHTML ("Encrypted: ", _
909:                                     objWbemObject.TargetInstance.Encrypted, _
910:                                     objWbemObject.PreviousInstance.Encrypted)
911:     strHTML = strHTML & FormatHTML ("FileType: ", _
912:                                     objWbemObject.TargetInstance.FileType, _
913:                                     objWbemObject.PreviousInstance.FileType)
914:     strHTML = strHTML & FormatHTML ("LastAccessed: ", _
915:                                     objWbemObject.TargetInstance.LastAccessed, _
916:                                     objWbemObject.PreviousInstance.LastAccessed)
917:     strHTML = strHTML & FormatHTML ("LastModified: ", _
918:                                     objWbemObject.TargetInstance.LastModified, _
919:                                     objWbemObject.PreviousInstance.LastModified)
920:     strHTML = strHTML & cEndHTMLTableData
921:
922:     AlertHandler objLogFileName, _
923:         "(E2K_StoreDBSink)", _
924:         "Information Store '" & strStoreName(intIndice) & _
925:         "' (Size=" & objWbemObject.TargetInstance.FileSize / 1024 & _
926:         " Kbytes) will be dismounted in " & _
927:         cWaitBeforeDBDismount / 1000 & ". ", _
928:         strHTML, _
929:         cMailAlert, _
930:         cEventLogAlert, _
931:         cCommandAlert, _
932:         cPopupAlert
933:
934:     AlertHandler objLogFileName, _
935:         "(E2K_StoreDBSink)", _
936:         "Pausing " & cWaitBeforeDBDismount / 1000 & "s. before action.", _
937:         strHTML, _
938:         False, _
939:         False, _
940:         False, _
941:         False
942:
943:     WScript.Sleep (cWaitBeforeDBDismount)
944:
945:     AlertHandler objLogFileName, _
946:         "(E2K_StoreDBSink)", _
947:         "Dismounting store '" & strStoreName(intIndice) & "'.", _
948:         strHTML, _
949:         False, _
950:         False, _
951:         False, _
952:         False
953:
954:     DismountStoreDB (strDNStoreDB (intIndice))
955:
956:     AlertHandler objLogFileName, _
957:         "(E2K_StoreDBSink)", _
958:         "Store '" & strStoreName(intIndice) & "' dismounted.", _
959:         strHTML, _
960:         cMailAlert, _
961:         cEventLogAlert, _
962:         cCommandAlert, _
963:         cPopupAlert
964: Else
965:     E2K_StoreDBInfo objLogFileName, _
966:         objWbemObject.TargetInstance, _
967:         objWbemObject.PreviousInstance
968: End If
969:
970: End If
971: Next
972:
973: intRC = WriteToFile (objLogFileName, "(E2K_StoreDBSink_OnObjectReady) End Sink.")
974: intRC = WriteToFile (objLogFileName, _
975:     "-----")

```

```

976:
977:End Sub
...:
...:

```

WMI *NTProcess* class

The *NTProcess* event handler uses the same logic as the *Win32_Processor* event handler (See Sample 15 on page 25) to retrieve the matches based on the content of the configuration file. The modifications reside in the possibility to use a wildcard. The string manipulation logic is the same as Sample 18 on page 27 for the *Win32_NTLogEvent* class.

Next, if the process names match (Sample 20 below, lines 1151 and 1152) for an associated maximum threshold (lines 1153 and 1154), the function *Win32_NTProcessInfo()* is invoked to format the related data (lines 1155 to 1157) and send a mail.

Sample 20 The WMI *NTProcess* class asynchronous event handler

```

....:
....:
1131:      ' -----
1132:      ' NTProcessSink
1133:      ' -----
1134:      Sub NTProcessSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
1135:
1136:          Dim strTempProcessName
1137:
1138:          On Error Resume Next
1139:
1140:          intRC = WriteToFile (objLogFileName, _
1141:                              "(NTProcessSink_OnObjectReady) Start Sink.")
1142:
1143:          For intIndice = 1 To Ubound(strProcessName)
1144:
1145:              If strProcessName(intIndice) = "*" Then
1146:                  strTempProcessName = objWbemObject.TargetInstance.Process
1147:              Else
1148:                  strTempProcessName = strProcessName(intIndice)
1149:              End If
1150:
1151:              If Ucase(strTempProcessName) = _
1152:                  Ucase(objWbemObject.TargetInstance.Process) And _
1153:                  CLng(intProcessCPUPercentageMax(intIndice)) <= _
1154:                  CLng(objWbemObject.TargetInstance.PercentageProcessTime) Then
1155:                  Win32_NTProcessInfo objLogFileName, _
1156:                                      objWbemObject.TargetInstance, _
1157:                                      objWbemObject.PreviousInstance
1158:              Exit For
1159:              End If
1160:          Next
1161:
1162:          intRC = WriteToFile (objLogFileName, _
1163:                              "(NTProcessSink_OnObjectReady) End Sink.")
1164:          intRC = WriteToFile (objLogFileName, _
1165:                              "-----")
....:
....:

```

WMI *ExchangeServerState* class

The *ExchangeServerState* event handler logic is not different than the one of other WMI event handlers. The registration is made to trigger a WMI event for any change to an instance of this class (See Sample 21 below). Once the loop (lines 990 to 997) has found a match (lines 991 and 992) for the triggered event and the configuration file data, the function *E2K_ServerStateInfo()* is called (lines 993 to 995) to format the data and send a mail.

Sample 21 The WMI *ExchangeServerState* class asynchronous event handler

```

...:
...:
980:' -----

```

```

981:' E2K_ServerStateSink
982:' -----
983:Sub E2K_ServerStateSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
984:
985:    On Error Resume Next
986:
987:    intRC = WriteToFile (objLogFileName, _
988:                        "(E2K_ServerStateSink_OnObjectReady) Start Sink.")
989:
990:    For intIndice = 1 To Ubound(strWMIExchangeServerName)
991:        If Ucase(strWMIExchangeServerName(intIndice)) = _
992:            Ucase(objWbemObject.TargetInstance.Name) Then
993:            E2K_ServerStateInfo objLogFileName, _
994:                                objWbemObject.TargetInstance, _
995:                                objWbemObject.PreviousInstance
996:        End If
997:    Next
998:
999:    intRC = WriteToFile (objLogFileName, _
1000:                       "(E2K_ServerStateSink_OnObjectReady) End Sink.")
1001:    intRC = WriteToFile (objLogFileName, _
1002:                       "-----")
1003:
1004:End Sub
....:
....:

```

WMI *ExchangeConnectorState* class

The event handler for the *ExchangeConnectorState* uses exactly the same logic as the *ExchangeServerState*. See the paragraph “WMI *ExchangeServerState* class” on page 30 for any remarks and information.

Sample 22 The WMI *ExchangeConnectorState* class asynchronous event handler

```

....:
....:
1006:    ' -----
1007:    ' E2K_ConnectorStateSink
1008:    ' -----
1009:    Sub E2K_ConnectorStateSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
1010:
1011:        On Error Resume Next
1012:
1013:        intRC = WriteToFile (objLogFileName, _
1014:                            "(E2K_ConnectorStateSink_OnObjectReady) Start Sink.")
1015:
1016:        For intIndice = 1 To Ubound(strWMIExchangeConnector)
1017:            If Ucase(strWMIExchangeConnector(intIndice)) = _
1018:                Ucase(objWbemObject.TargetInstance.Name) Then
1019:                E2K_ConnectorStateInfo objLogFileName, _
1020:                                        objWbemObject.TargetInstance, _
1021:                                        objWbemObject.PreviousInstance
1022:            End If
1023:        Next
1024:
1025:        intRC = WriteToFile (objLogFileName, _
1026:                            "(E2K_ConnectorStateSink_OnObjectReady) End Sink.")
1027:        intRC = WriteToFile (objLogFileName, _
1028:                            "-----")
1029:
1030:    End Sub
....:
....:

```

WMI *ExchangeLink* class

The *ExchangeLink* class event handler allows the usage of a wildcard for the link name. It uses the same string manipulation logic as Sample 18 on page 27 or Sample 20 on page 30. The *E2K_LinkInfo()* function (See Sample 23 on page 32) is called:

- If the link name matches a link name given in the configuration file (See Figure 1 on page 16, line 115) and

- If the *increasingTime* value reaches the threshold given in the configuration file.

Sample 23 The WMI *ExchangeLink* class asynchronous event handler

```

.....:
.....:
1032:      ' -----
1033:      ' E2K_LinkSink
1034:      ' -----
1035:      Sub E2K_LinkSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
1036:
1037:          Dim strTempWMILinkName
1038:
1039:          On Error Resume Next
1040:
1041:          intRC = WriteToFile (objLogFileName, _
1042:                              "(E2K_LinkSink_OnObjectReady) Start Sink.")
1043:
1044:          For intIndice = 1 To Ubound(strWMIExchangeLink)
1045:              If strWMIExchangeLink(intIndice) = "*" Then
1046:                  strTempWMILinkName = objWbemObject.TargetInstance.LinkName
1047:              Else
1048:                  strTempWMILinkName = strWMIExchangeLink(intIndice)
1049:              End If
1050:
1051:              If UCase(strTempWMILinkName) = _
1052:                  UCase (objWbemObject.TargetInstance.LinkName) And _
1053:                  CLng(intWMIExchangeLinkIncreasingTime(intIndice)) <= _
1054:                  CLng(objWbemObject.TargetInstance.IncreasingTime) Then
1055:                  E2K_LinkInfo objLogFileName, _
1056:                  objWbemObject.TargetInstance, _
1057:                  objWbemObject.PreviousInstance
1058:              End If
1059:          Next
1060:
1061:          intRC = WriteToFile (objLogFileName, _
1062:                              "(E2K_LinkSink_OnObjectReady) End Sink.")
1063:          intRC = WriteToFile (objLogFileName, _
1064:                              "-----")
1065:
1066:      End Sub
.....:
.....:

```

WMI *ExchangeQueue* class

The event handler for the *ExchangeQueue* uses exactly the same logic as the *ExchangeLink* class. See the paragraph “WMI *ExchangeLink* class” on page 31 for any remarks and information.

Sample 24 The WMI *ExchangeQueue* class asynchronous event handler

```

.....:
.....:
1068:      ' -----
1069:      ' E2K_QueueSink
1070:      ' -----
1071:      Sub E2K_QueueSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
1072:
1073:          Dim strTempWMIQueueName
1074:
1075:          On Error Resume Next
1076:
1077:          intRC = WriteToFile (objLogFileName, _
1078:                              "(E2K_QueueSink_OnObjectReady) Start Sink.")
1079:
1080:          For intIndice = 1 To Ubound(strWMIExchangeQueue)
1081:              If strWMIExchangeLink(intIndice) = "*" Then
1082:                  strTempWMIQueueName = objWbemObject.TargetInstance.QueueName
1083:              Else
1084:                  strTempWMIQueueName = strWMIExchangeQueue(intIndice)
1085:              End If
1086:
1087:              If UCase(strTempWMIQueueName) = _
1088:                  UCase (objWbemObject.TargetInstance.QueueName) And _
1089:                  CLng(intWMIExchangeQueueIncreasingTime(intIndice)) <= _

```

```

1090:                CLng(objWbemObject.TargetInstance.IncreasingTime) Then
1091:                    E2K_QueueInfo objLogFileName, _
1092:                        objWbemObject.TargetInstance, _
1093:                        objWbemObject.PreviousInstance
1094:                End If
1095:            Next
1096:
1097:            intRC = WriteToFile (objLogFileName, _
1098:                                "(E2K_QueueSink_OnObjectReady) End Sink.")
1099:            intRC = WriteToFile (objLogFileName, _
1100:                                "-----")
1101:
1102:        End Sub
1103:
1104:
1105:

```

WMI ExchangeClusterResource class

The event handler for the *ExchangeClusterResource* uses exactly the same logic as the *ExchangeServerState*. See the paragraph “WMI ExchangeServerState class” on page 30 for any remarks and information. It is important to note that this event is relevant only if you run the script with an Exchange 2000 cluster.

Sample 25 The WMI ExchangeClusterResource class asynchronous event handler

```

1104:
1105:
1106:
1107:        Sub E2K_ClusterResourceSink_OnObjectReady (objWbemObject, objWbemAsyncContext)
1108:
1109:            On Error Resume Next
1110:
1111:            intRC = WriteToFile (objLogFileName, _
1112:                                "(E2K_ClusterResourceSink_OnObjectReady) Start Sink.")
1113:
1114:            For intIndice = 1 To Ubound(strWMIExchangeClusterVName)
1115:                If Ucase(strWMIExchangeClusterVName(intIndice)) = _
1116:                    Ucase(objWbemObject.TargetInstance.VirtualMachine) Then
1117:                    E2K_ClusterResourceInfo objLogFileName, _
1118:                        objWbemObject.TargetInstance, _
1119:                        objWbemObject.PreviousInstance
1120:                End If
1121:            Next
1122:
1123:            intRC = WriteToFile (objLogFileName, _
1124:                                "(E2K_ClusterResourceSink_OnObjectReady) End Sink.")
1125:            intRC = WriteToFile (objLogFileName, _
1126:                                "-----")
1127:
1128:        End Sub
1129:
1130:
1131:

```

Added-functions not directly related to the Exchange 2000 Management

Many other functions are included in the “E2KWatch” script. These functions are only a set of helper functions to make this script usable in a production environment.

The script includes some helper functions such as:

- The Log file Creation: The *CreateLogFile()* function creates a local log file containing a trace activity of the script. The log filename is a default log file specified in the script as a constant.

- The HTML formatter: The *FormatHTML()* function is used to make string manipulations to insert miscellaneous HTML tags. The purpose is to build a HTML body with the WMI data. This HTML body will be sent by the *SendMessage()* function.
- The Alert Handler: The *AlertHandler()* function is a function mainly called by the *ClassNameInfo()* functions to send a mail. This function can also generate popup window or invoke a default .CMD file “ALERT.CMD” when an alert occurs. See constants definitions in the script header for more information.
- The Error Handling: The *ErrorHandler()* has the same structure as the *AlertHandler()* function. The difference resides in the usage of the function because it is present to resume any errors that can occur during the script execution. This helps differentiate alerts and errors.
- The configuration file: The *ReadConfigurationFile()* function is used to read, parse and make some basic data validation on the definitions included in the configuration file. The configuration file can be specified on the command line or when the script prompts the user for a configuration file. (If no command line parameter is given)
- The Dismount Store: The *DismountStore()* function is used by the *CIM_DATAFile* WMI class event handler (See Sample 19 on page 28, line 954). This function uses the CDOEXM method to dismount a store (See Part 1, Sample 9 for a Mailbox Store or Sample 16 for a Public Store).
- The class of Store: The function *GetStoreDBClass()* is a helper function for the *DismountStore()* function to determine, based on the *distinguishedName* of the store given in the configuration file, if it is a Mailbox Store or Public Store.
- The path of the EDB file: The *GetStoreMDBPaths()* function is a helper function to determine the path of the EDB file based on the *distinguishedName* of the store given in the configuration file.
- The path of the STM file: The *GetStoreSTMPaths()* function is a helper function to determine the path of the STM file based on the *distinguishedName* of the store given in the configuration file. This function has the same logic as the *GetStoreMDBPaths()* function.
- Sending messages: This function is a helper function for the *ClassNameInfo()* function to send the formatted HTML data by mail. This function uses the information explained in Part 1, Sample 23 (line 40 for the *HTMLBody* and line 76 for the *Send* method).

The possible enhancements

The “E2KWatch” script is not perfect. Many enhancements can be done to the script. Of course, the adaptations can be infinite and will probably never perfectly match the needs of everyone. Nevertheless, here are some general functions that can be added to the script:

- The script sends a mail if a CPU usage (Process or Processor) reach a certain threshold. A nice enhancement is to send an alert if the CPU usage threshold is maintained for a certain period of time. In a real world environment, it is normal to have some CPU peak usages and this does not represent a problem. On the other hand, having a CPU usage of 90% for 15 minutes or more can indicate a serious performance problem. In such a situation, the script, as it is today, will react but it will send a lot of alerts (one for each peak detected)
- If a service monitored by the script is changed from a manual or automatic startup mode to a disabled startup mode, the script will try to start the service even if the service is disabled. This will result in an error because the service manager does not allow a disabled service to start. An enhancement is to take this situation in consideration.
- The script saves all the activity in a log file. In its current version, this log file is never purged. A nice enhancement is to use the WMI event timer to trigger an event every day to purge or archive the log.
- The configuration file holds all the parameters for the script. When a change is made to this file, is it necessary to restart the script to take the new changes in consideration. A possible enhancement is to program an asynchronous event with the `CIM_DATAFile` class to monitor the configuration file. If a change is made to the file, the script will cancel all WMI asynchronous events registered, refresh data from the updated configuration file and reprogram all the events.
- The script monitors the links with the *ExchangeLink* class based on the *increasingTime* value. If the link is a dial-up link, the script does not take this particularity in consideration. It means that the script will send an alert because the number of mail in the queue increases regularly. But in the case of a scheduled link, it is a normal situation because the link waits for the next scheduled time to establish the connection.

Here is, as samples, a screen snapshot showing what you can expect to have when running the “E2KWatch” script.

DC01-CPQ - ALERT:(E2K_LinkInfo) - Tuesday, 23 May, 2000 at 18:29 - '_fd15f7faede3d7409e18634a77ac5340_D' Exchange link has an increasing time of 10422 ms. - Me...

File Edit View Insert Format Tools Actions Help

Reply Reply to All Forward

From: E2KWatch@E2KWatch.Wsf Sent: Tue 23-05-2000 18:30
 To: LIS50IR Alain
 Cc:
 Subject: DC01-CPQ - ALERT:(E2K_LinkInfo) - Tuesday, 23 May, 2000 at 18:29 - '_fd15f7faede3d7409e18634a77ac5340_D' Exchange link has an increasing time of 10422 ms.

DC01-CPQ (E2K_LinkInfo)

	Previous State	Current State
LinkName:	_fd15f7faede3d7409e18634a77ac5340_D	_fd15f7faede3d7409e18634a77ac5340_D
ProtocolName:	SMTP	SMTP
VirtualServerName:	1	1
VirtualMachine:	DC01-CPQ	DC01-CPQ
Version:	4	4
NumberOfMessages:	9	9
NextScheduledConnection:	20000524014810.000750+***	20000524014810.000750+***
OldestMessage:	20000524010235.000565+***	20000524010235.000565+***
SizeOfQueue:	170910	170910
LinkDN:	fa715fd-e3ed-40d7-9e18-634a77ac5340	fa715fd-e3ed-40d7-9e18-634a77ac5340
ExtendedStateInfo:	The remote server did not respond to a connection attempt.	The remote server did not respond to a connection attempt.
IncreasingTime:	922	10422
StateFlags:	260	260
StateActive:	False	False
StateReady:	False	False

Figure 2: A sample "E2KWatch" mail alert for the *increasingTime* value

Conclusion

Exchange 2000 and Windows 2000 come with a lot of COM objects usable from scripting languages. By using the combined forces of ADSI, CDOEX, CDOEXM and WMI, an administrator can perform most of the management tasks related to a Windows 2000/Exchange 2000 Enterprise network.

The two advanced samples given in this document illustrate a small part of the possibilities offered. With a good logic and some scripting knowledge, it is possible for an Administrator to build his own tools for his specific management needs. For instance, moving Exchange mailboxes based on simple query is a feature that many Exchange 5.5 administrators were waiting for. With Exchange 2000, the combination of ADSI and CDOEXM with a small VBScript is enough to provide the expected answer.

Learning one scripting language combined with the COM object model of ADSI, CDOEX, CDOEXM and WMI is a good investment for the future. This knowledge will enable administrators to get a better control and a better understanding of their own environment.