OpenOPC Notes

December, 2008

1 Introduction

Most DCS (Distributed Control Systems) and many new devices (like analyzers) support OPC (OLE for Process Control). OPC is one option offered by data historians as a way of retrieving data. Making OPC connections from a third party software product on a separate computer can be a frustrating exercise. OPC uses the Microsoft protocols COM (local access) and DCOM (remote access) for the communication which makes it difficult to access OPC Servers from other operating systems like Linux. Getting remote OPC connections working often require changes to the DCOM security settings. Matrikon has documents on their web site explaining how to configure DCOM to make remote OPC connections work. In reality it can be quite complicated to get remote OPC connections working. This is certainly true for Windows XP SP2 where additional security settings further complicates the configuration.

An Open Source software project called OpenOPC tries to change this. With OpenOpc you can access OPC servers from other platforms. OpenOpc also gives you the possibility to bypass DCOM security by installing a service that allows remote TCP/IP connections requesting OPC data. The Gateway service will typically be installed on the same machine where the OPC server is installed, making it much simpler to ensure the Gateway service has the correct permissions allowing it to connect to the OPC server. The Gateway service is also used by non-Windows operating systems that do not support DCOM. The software is freely available here:

`http://openopc.sourceforge.net`

This document is written from a user's perspective. The document tries to use examples as much as possible to show how the OpenOPC software works[1].

[1]This document is based upon OpenOPC version 1.1.5

2 Installation on Test Machine

It is always a good idea to test software on a non-production machine before rolling it out to the real world. The OPC software mentioned in this chapter has been installed several times on computers running various versions of Windows (Windows NT 4.0, Windows 2000, Windows XP). The software appear to work well and do not appear to cause any problems for other applications. This section will cover the installation of on an OPC server and the OpenOPC gateway service that allows remote clients (even non-windows) to retrieve OPC data from computer. In later sections we will refer to this computer as the *server*.

2.1 Install OPC server

An OPC server is needed to test the OpenOPC software. Matrikon has a freely available test server than can be downloaded from their web-site:

```
http://www.matrikon.com
```

Another place to get the Matrikon OPC Server (without having to give contact information) is from Kinetica:

```
http://www.kineticart.co.uk/KineticaRTOPC.asp
```

There are many other free OPC servers available for download some are listed here:

```
http://www.opcconnect.com/freestuf.php
```

The OpenOPC tools have been heavily tested against the Matrikon OPC Simulation Server, and the default settings for the OpenOPC tools is for connection to the Matrikon OPC Server. Thus this document will mostly show how to make connections to the Matrikon OPC server.

Two different OPC servers are likely to behave slightly different. Not all functions available in one server may be available in another server. All OPC servers generally support read and write, but browsing (using OPC server to look up available tags in Server) is often implemented differently or not supported by all OPC servers. There may (for obvious reason) be additional security related to writing to an OPC server connected to a DCS. These kind of security settings is from what I understand not part of the OPC standard, at least not in the form you see this implemented by DCS vendors.

2.2 Install OpenOPC software

To install the OpenOPC software, simply run the installler. When the installer asks which components to install select:

Command Line Client: This installs the command line client (opc.exe) which is a powerful command line tool for retrieving OPC data.

OPC Gateway Service: This is the service that allows non-windows machines to access OPC data. Basically this service accesses the OPC data and passes it on to the client asking for the data. This service is also what allows you to eliminate DCOM configuration issues as remote hosts can get the data using the service and not through DCOM.

Development Library: This library is needed when developing your own OPC applications. Examples of this will be shown in section 4.

Source Code: This option installs the source code for the opc client and the gateway server. Note that the installer will complain if Python is not installed on the machine, but the files will still be placed in the *SRC* directory for inspection. This document does not discuss the source code. The source code takes very little space.

A reboot is not required, but if you install the software on NT 4.0 it may be a good idea. The OpenOPC software is typically installed in C:\OpenOPC. If you install the software on a machine running NT 4.0, the installer will place msvcr71.dll (Microsoft C Runtime Library) in the C:\WINNT\SYSTEM32 directory. This DLL is generally always present on machines running Windows XP. The DLL is not present on a machine with Windows 2000 freshly installed, but may have been installed by other applications.

2.3 Quick test of software on local machine

Several default setting for the program opc.exe has been set using environment variables. You can see these settings by typing "SET" from the command prompt and looking for lines that start with OPC:

```
OPC_CLASS=Matrikon.OPC.Automation;Graybox.OPC.DAWrapper;HSCOPC.Automation;
OPC_CLIENT=OpenOPC
OPC_GATE_HOST=localhost
OPC_GATE_PORT=7766
OPC_HOST=localhost
```

```
OPC_MODE=dcom
OPC_SERVER=Hci.TPNServer;Matrikon.OPC.Simulation;Prosys.OPC.Simulation
```

To read one tag from the Matrikon OPC server using the default settings simply type:

```
opc Random.Int4
```

You can get some information about the default OPC server like this:

```
opc -i
```

You can list the OPC servers on the local machine like this[2]:

```
opc -q
```

If the OPC server supports browsing (the Matrikon OPC Server does) you can search for tags matching certain patterns using wild cards. The following command list all tags having the extension Real8.

```
opc -f *.Real8
```

Which will produce a list of tags from the Matrikon Simulation Server like this:

```
Bucket Brigade.Real8
Random.Real8
Read Error.Real8
Saw-toothed Waves.Real8
Square Waves.Real8
Triangle Waves.Real8
Write Error.Real8
Write Only.Real8
```

The program opc.exe has good support for using pipes. Thus it is possible to read the values of these tags in a command like this:

```
opc -f *.Real8 | opc -
```

Which will display the following output:

[2] On Windows NT 4.0 it is sometimes necessary to specify the hostname instead of the default (localhost) to get a list of OPC servers: `opc -h testbox -q` (assuming host name of computer is *testbox*).

```
Bucket Brigade.Real8        0.0000       Good   10/26/07 22:41:16
Random.Real8                16499.4963   Good   10/26/07 22:41:16
Read Error.Real8            -1073479676  Good   10/26/07 22:41:16
Saw-toothed Waves.Real8     3.1416       Good   10/26/07 22:41:16
Square Waves.Real8          0.0000       Good   10/26/07 22:41:16
Triangle Waves.Real8        3.1416       Good   10/26/07 22:41:16
Write Error.Real8           0.0000       Good   10/26/07 22:41:16
Write Only.Real8            0.0000       Bad    10/26/07 22:40:41
```

2.4 Verifying Installation of Gateway Service

The gateway server will be used by remote clients to get OPC data. This sub-section explains how to verify the gateway server is working properly before we try to access it from a remote client computer.

First look in Task-Manager to see if the process "openopcservice" is running. If not, it can be started like this[3]:

```
NET START zzzopenopcservice
```

If errors like: *"the service name is invalid"* are encountered, the service was likely not installed correctly. Verify that "OpenOPCService.exe" is located in C:\OpenOPC\bin and install the service like this:

```
openopcservice -install -auto
NET START zzzopenopcservice
```

Try to list the current connections to the gateway server - there should be none:

```
opc -S
```

Try to read one tag using the gateway server[4]:

```
opc -m open Random.Int4
```

You can easily convince yourself that the gateway server was used for retrieving the data. Simply stop the gateway server and try to read the tag again. You will recieve an error message:

[3]In OpenOPC version 1.1.5 the name of the windows service was changed from openopcservice to zzzopenopcservice to make sure the OpenOPC Gateway Service would start after all other services was started.

[4]If this does not work, the SYSTEM account may not have the correct permissions. One thing I have found that works is to change the startup of the service to use an administrator account.

```
NET STOP zzzopenopcservice
opc -m open Random.Int4
```

Remember to start the gateway service again before proceeding with the next section.

3 Manual installation on client machine

This section covers the manual installation of the OpenOPC software on a separate machine - *the client*. The purpose of explaining the manual installation is to give the user a more detailed understanding of how the software works, The data will be retrieved through the gateway service on the remote machine, this eliminates DCOM configuration issues. No OPC server will be installed on the client machine. The install procedure involves nothing more than copying "opc.exe" to a new machine. Preferably you place "opc.exe" in the PATH (like a tools directory), but this is entirely up to you. If you are using NT 4.0 you may need to place a copy of msvcr71.dll (Microsoft C Runtime Library) in the C:\WINNT\SYSTEM32 directory if it is not there.

3.1 Reading a Value from the remote host using gateway

The environment variables will not be set since we did not run the installer on the client machine. This means we will have to specify more options when making the call. Assume that the machine (the server) we installed the Matrikon OPC Simulation and OpenOPC software on before is called *"TESTBOX"*.

To quickly test that opc.exe can talk to the gateway service on the server, execute the following command on the client machine:

```
opc -S -H TESTBOX
```

This command will show the remote connections to the gateway service - there should be no remote connections. If errors are encountered, you need to ensure that the gateway service is running on the remote computer as described in section 2.4. If name resolution is not working properly, the IP-address of the remote computer name can be used in the command:

`opc -S -H 10.36.16.100` (replace with IP address of TESTBOX)

Try reading a value from the Matrikon OPC Server on *TESTBOX* by typing (note the case sensitivity of the options) the following on one line at the command prompt:

```
opc -m open -H TESTBOX -h TESTBOX -s Matrikon.OPC.Simulation -r Random.Int4
```

The "-H" option means we want to use the gateway service on *TESTBOX*. The "-h" option specifies the host name of computer the running the OPC server. The "-s" option specified the name of the OPC server. The "-r" option means read and is optional (i.e. not needed).

You can set the following environment variables to avoid having to type the long command line. To temporary set these environment variables, simply type the following from the command line:

```
SET OPC_GATE_HOST=TESTBOX
SET OPC_HOST=TESTBOX
SET OPC_MODE=open
SET OPC_SERVER=Matrikon.OPC.Simulation
```

These environment variables should probably be set permanently once you have the configuration working properly. With the environment variables set, the command for retrieving data remotely becomes much simpler:

`opc -r Random.Int4` (Note the "-r" is optional)

3.2 Gateway Port

The gateway service uses port 7766 by default for TCP/IP communication. If you change this port number (set by environment variable `OPC_GATE_PORT`) you will need to do so on both the server machine and the client machine. Firewalls on either the remote or client computer can cause connection problem. Make sure that the gateway port is open on both the server and the client.

To verify that the gateway port is being used try the following command from the client machine:

```
opc -r Random.Int4 -L1
```

This command assumes that the environment variables, shown in section 3.1, are set such that there is no need to specify hostname and opc server name in the command. The "-L1" option tells the opc.exe program to retrieve the data every 1 second continuously. Now from a second command prompt on the remote machine type "netstat" to see TCP/IP connections. You will clearly see that port 7766 is being used for TCP/IP communication between client machine (laptop) and the server running the gateway service (testbox):

```
Proto  Local Address  Foreign Address  State
TCP    laptop:1089    testbox:7766     TIME_WAIT
TCP    laptop:1090    testbox:7766     ESTABLISHED
```

The gateway service on the server can be queried to show open connections. Execute the following command on the client computer:

```
opc -S -H TESTBOX
```

The output will look similar to this if the continuous data request is still running at another command prompt on the client computer:

```
Remote Client          Start Time          Last Transaction
10.36.16.58 (labtop)   06/27/08 14:53:01   06/27/08 14:53:07
```

4 Developing OPC applications with Python

The real power of OpenOPC is the ability to use it within Python. Python offers easy connectivity to many databases and makes it very easy to develop web applications. Python runs on many platforms including Windows, Linux and possibly your PDA. In addition to Python, the following Python add-on packages are required (see *readme.txt* included with OpenOPC package):

Python Remote Objects (Pyro)
Python for Windows Extensions (pywin32)

This document is not a comprehensive tutorial on using Python with OpenOPC. The main purpose this section is to make the user aware that it is possible to develop powerful applications accessing OPC data in Python.

4.1 Simple Example

Below is a simple listing of Python code for reading two OPC values from the remote gateway server on *TESTBOX*.

```
import OpenOPC

gateway='testbox'
opchost='testbox'
```

```
opcserv='Matrikon.OPC.Simulation'
taglist =['Random.Int4','Random.Real4']

print 'Connecting to Gateway Server on: ',gateway
opc = OpenOPC.open_client(gateway)
opc.connect(opcserv,opchost)
v = opc.read(taglist)
opc.close()

for i in range(len(v)):
    (name, val, qual, time) = v[i]
    print '%-15s %-15s %-15s  %-15s'%(name,val,qual,time)
```

The Python program above produces the following output:

```
Connecting to Gateway Server on:  testbox
Random.Int4     9961.0   Good   10/29/07 04:23:52
Random.Real4    491.0    Good   10/29/07 04:23:52
```

4.2 Writing OPC data to database

Below is another simple example. OPC data is retrieved and written to a *SQLite* database. The Python program is made simple on purpose without proper error checking. A real program should perform proper error checking for failed OPC connections etc. The purpose is to illustrate the power of OpenOPC and Python. It would be very simple to develop a web application that displayed the values stored in the database.

```
# OpenOPC and SQLite Python example
import sqlite3
import os
import time
import OpenOPC

dbfile='test.db'

remotehost='testbox'
opcserver='Matrikon.OPC.Simulation'
tagname='Random.Int4'
sampletime = 2

dbexist = os.path.exists(dbfile)
db = sqlite3.connect(dbfile)
```

```
cur = db.cursor()

# create table first time new database is accessed
if not dbexist:
    sqlstring = 'CREATE TABLE opcdata \
    (id INTEGER PRIMARY KEY, tag TEXT, \
    val TEXT, stat TEXT, datetime TEXT, atime TEXT)'
    cur.execute(sqlstring)
    db.commit()

while 1>0: # infinite loop kill with Ctrl-C
    opc=OpenOPC.client()
    opc.connect(opcserver,remotehost)
    (opcvalue,opcstat,opctime) = opc.read(tagname)
    opc.close()

    atime = time.asctime() # current computer time as float
    sqlstring = 'INSERT INTO opcdata (id, tag, val, stat, datetime, atime) \
    VALUES(NULL, "%s", "%f", "%s", "%s", "%s")'\
    %(tagname,opcvalue,opcstat,opctime,atime)

    cur.execute(sqlstring)
    db.commit()
    cur.execute('SELECT * FROM opcdata')
    dbvalues = cur.fetchall()
    dblen = len(dbvalues)
    print dbvalues[dblen-1]  # print newest value in database

    time.sleep(sampletime)
# end of while loop
```

5 Using OPC client in DCOM mode

If the OpenOPC command line client *opc.exe* is used to read values locally (COM) things works well. However, if the client *opc.exe* are used to read values remotely (DCOM) things gets more complicated. First DCOM security needs to be configured, which can be quite complicated[5]. A trick that often works is do be logged in to the two computers that need to talk with the same account that preferably should have administrative rights on both computers. Once a connection is made, accounts with provileges less than Administrator can be created.

[5]DCOM security is outside the scope of this writeup.

To illustrate some of the problems, first install the Prosys OPC server (see section 6.1) on a clean client machine (laptop). This will install the following OPC foundation files in the *SYSTEM32* directory needed to create OPC connections:

```
05-11-05 11:44a 61,440 opccomn_ps.dll
05-11-05 11:44a 98,304 OpcEnum.exe
05-11-05 11:44a 102,400 opcproxy.dll
```

Try to list servers on the local machine *laptop*:

```
opc -h laptop -q
```

This should display the Prosys OPC Server: `Prosys.OPC.Simulation`. Try to list servers on the remote machine (assume name is *TESTTBOX* and that the Matrikon Simulation Server is installed):

```
opc -h TESTBOX -q
```

Reading values from the local machine (laptop) works fine:

```
opc -m dcom -s Prosys.OPC.Simulation -r Random.PsFloat1
```

But reading a value from the Matrikon OPC Server on the remote host (*TESTBOX*) fails:

```
opc -m dcom -h TESTBOX -s Matrikon.OPC.Simulation.1 -r Random.Int4
```

This may be a DCOM configuration issue not yet understood. A workaround that appear to work is to install the Matrikon Simulation Server on the local machine (laptop). OpenOPC will then use the OPC Class (Matrikon.OPC.Automation) provided by the Matrikon OPC Server for making remote OPC connections because the automation class Matrikon.OPC.Automation is listed before the OPC Class provided by Graybox (Graybox.OPC.DAWrapper) in the enviroment variable *OPC_CLASS*. The following command now works:

```
opc -m dcom -h TESTBOX -s Matrikon.OPC.Simulation.1 -r Random.Int4
```

Forcing *opc.exe* to use the Automation Class provided by the Gray Box DLL also works after the installation of the Matrikon Simulation Server. Thus it appear that the Matrikon Simulation Server changes some DCOM setting that makes this work.

```
opc -m dcom -C Graybox.OPC.DAWrapper -h TESTBOX -s Matrikon.OPC.Simulation.1
-r Random.Int4
```

Summary:
We have seen that getting remote OPC connections working with OpenOPC can be complicated. This is one of the main reasons for developing the gateway service which eliminates the need for remote OPC connections. Basically this means that the easiest installation of OpenOPC occurs when OpenOPC is installed on the host where the OPC server is running. However, if it is necessary to create remote OPC connections with OpenOPC, then a workaround for the problems encountered is to install the Matrikon OPC Simulation Server on the same machine.

6 Specific OPC Servers

This section will cover the use of OpenOPC together with various OPC servers. It is the hope that this section can be expanded by the user community over time.

6.1 Prosys OPC Server

Prosys has a well working OPC test server available for free at their web site:

```
http://www.prosys.fi/downloads.html
```

Assume this OPC server is installed on the *TESTBOX* server where the Matrikon OPC Server is also installed (see section 2). The OPC servers can be listed like this:

```
opc -h TESTBOX -q
```

If the Matrikon SImulation Server is installed, then *opc.exe* will use the OPC class *Matrikon.OPC.Automation* (check with `opc -i`) and the above command will only list the Matrikon OPC server. It is however still possible to read data from the Prosys OPC Server:

```
opc -h TESTBOX -s Prosys.OPC.Simulation -r Random.PsFloat1
```

If the Automation class from the freeware Graybox DLL is used (located in C:\OpenOPC\lib), then the `opc -q` command will show both OPC servers:

```
opc -C Graybox.OPC.DAWrapper -h TESTBOX -q
```

Wild card browsing works well with the Prosys OPC Server:

```
opc -h TESTBOX -s Prosys.OPC.Simulation -f *.PsFloat1
```

6.1.1 Using Prosys OPC Server to Monitor OPC Calls

One of the nice features in the Prosys OPC Server is the ability to see the OPC calls made against the server. When connections to the Prosys Server is made, a small popup window will allow you to monitor the OPC connections. This is a very useful faeture when developing your own OPC client.

Note that the Prosys OPC Server may not always run as a visible program. To force the Prosys OPC Server to always be visible use *dcomcnfg.exe* to force the Prosys OPC Server to run as *"The interactive user"*.

Using the Prosys OPC Server it is easy to see that when *opc.exe* is forced to use *"sync"* mode that the OPC function *"IOPCSync.Read"* is used:

```
opc -F sync -s Prosys.OPC.Simulation -r Random.PsFloat1 -L5
```

Similarily it is easy to see that when *opc.exe* is forced to use *"async"* mode that the OPC function *"IOPCAsync02.Refresh"* is used:

```
opc -F async -s Prosys.OPC.Simulation -r Random.PsFloat1 -L5
```

It is also seen that the OPC updaterate is set to 5000 ms (5 sec) in *async* mode.

6.2 Honeywell TDC3000 OPC Server

The Honeywell TDC3000 is a distributed control system (DCS) in wide use in 2008. It is possible to get OPC data from the Honeywell TDC3000 through an OPC server supplied by Honeywell. The OPC server runs on an Windows computer connected to the LCN. Honeywell sells this computer and it is typically called an APP node. The OPC server is typically called the TPN server. The TPN OPC Server is likely the slowest of the OPC servers discussed in this section. It is easy to temporarily overwhelm the TPN Server. Never versions of the TPN server support browsing, but be careful not to submit large wildcard requests if critical applications are getting data through the TPN server.

6.2.1 Installing OpenOPC on an APP node

It is possible to install OpenOPC on an Honeywell APP node. The name of the OPC server is typically *Hci.TPNServer*. The default OPC server should be changed:

```
OPC_SERVER=Hci.TPNServer
```

Test the connection by reading a single tag from the DCS (replace host name and tag name with names valid for system):

```
opc -h APPNODE -s Hci.TPNServer 01FI100.PV
```

If this does not work, consider testing if the account used for logging on to the APP has permission to read OPC data. This can be done using one of many free OPC clients. See section 7 for details.

Verify that data can be read through the gateway service (replace host name and tag name with names valid for system):

```
opc -m open -H APPNODE -h APPNODE -s Hci.TPNServer 01FI100.PV
```

If this does not work, veryfy that gateway service is running. The gateway service runs as SYSTEM by default. It may be necessary to have it run as a user with permission to access OPC data.

6.2.2 Installing OpenOPC on a GUS node

It is also possible to install the OpenOPC software on a Honeywell GUS or any other machine in the same Domain. In this case the OpenOPC gateway service on the GUS will read the OPC data on the APP node using DCOM. A requirement is that the OPC Foundation files are present (look for files starting with *opc* in C:\Windows\System32). As we saw in section 5, it can be complicated getting remote OPC calls working. A workaround is to install the Matrikon Simulation Server on the GUS which solves the problem.

The following environment variables need to be set on the GUS:

```
OPC_MODE=dcom
OPC_HOST=APPNODE   (replace name with correct host name)
OPC_CLASS=Matrikon.OPC.Automation
OPC_SERVER=Hci.TPNServer
```

With the above enviroment variables set, it is simple to read a single tag from the OPC server on the APP node (replace host name and tag name with names valid for your system):

```
opc -h APPNODE -s Hci.TPNServer 01FI100.PV
```

It is important that the account used for login to the GUS has permission to read OPC data from the APP node.

6.2.3 Installing OpenOPC Gateway Service on GUS node

It is possible to have the Gateway service running on the GUS reading values from the APP node. In this slightly complicated configuration data flows from the APP to the GUS in DCOM mode. Other machines on the network gets OPC data from the GUS using the Gateway Service.

First verify that data from the OPC server on the APP can be read through the gateway service on the GUS (replace host name and tag name with names valid for system):

```
opc -m open -H GUSNAME -h APPNODE -s Hci.TPNServer 01FI100.PV
```

This may not work because the SYSTEM account used for running the OpenOPCService on the GUS may not have permission to read OPC data on the APP node. If this is the case, try to configure the OpenOPCService to use an account with permission to access OPC data on the APP.

6.2.4 Building the cache

The Honeywell TPN server uses a cache to speed up data requests. First time a tag it accessed, it can take a long time to get a response. After a tag has been requested once, the following requests are much faster. It is possible to read all tags in a long list by piping the contents of a file to opc.exe:

`opc - < list.txt` **(Do not do this first time!)**

Instead of trying to read all new tags at once, it is possible to read a few tags at the time in a long list with a small wait between reads:

```
opc -g 3 -z 500 - < list.txt
```

The ”-g” option specifies the group size. The ”-z” option specifies the time to wait between reads in milliseconds. The ”-” option indicates input will come from a pipe. By reading a long list this way the first time, there is less of a change of overwhelming the TPN server which can cause problems for other applications accessing the TPN server at the same time. The output can be captured to a file. Having the output in a file allows for easy inspection of OPC read errors which could indicate misspelled tag names.

```
opc -g 3 -z 500 - < list.txt > output.txt
```

Note that if the the tag list is long (say 2000 tags), it will take a fair amount of time (minimum 2000*500/3 = 333333 milliseconds = 5.5 minutes) before the data request is complete in the example shown above. Once the tag list has been validated and misspelled or missing tags have been corrected there is no need to use the ”-z” option and the group size ”-g” option can be quite large (1500 to 2000):

```
opc -g 1500 - < list.txt > output.txt
```

Once tags have been read at least once, reading a long list is very fast (100-200 milliseconds or better).

6.3 Honeywell PKS Experion OPC Server

The Honeywell PKS Experion has a very fast OPC server. The OPC server appear to work very well for native PKS tags. If the PKS OPC server is asked for tags from the LCN, the OPC server does not appear to always be relaible. The number of OPC connections to the PKS OPC server is, from what I understand, licensed. Thus if you have problems connecting to the PKS OPC server, check the number of licenses purchased.

The OpenOPC software has been tested on a PKS system and appear to work well. The PKS OPC server is typically named *Hsc.OPCServer*. The OpenOPC environment variables should be changed to reflect that:

```
OPC_SERVER=Hsc.OPCServer
```

OpenOPC installs the freeware Graybox DLL (located in C:\OpenOPC\lib) which contains the class: *Graybox.OPC.DAWrapper*. By default OpenOPC tries to use the first Automation DLL it finds in the following order (specified in environment variable *OPC_CLASS*):

1. Matrikon.OPC.Automation
2. Graybox.OPC.DAWrapper
3. HSCOPC.Automation
4. RSI.OPCAutomation
5. OPC.Automation

To verify the Automation class being used type:

```
opc -i
```

It is possible to force OpenOPC use the Automation DLL shipped with the PKS OPC server by changing the environment variable:

```
OPC_CLASS=HSCOPC.Automation
```

7 OPC Clients

When troubleshooting OPC connection issues it is convenient to have different OPC clients to ensure the problem is not with the client. The Matrikon OPC Server ships with a OPC test client. Thus if you install the Simulation server, you will also have a well working client. The Matrikon OPC Server installer can also be used to just install a OPC client.

Another well working OPC client (very simple install) is the Prosys OPC client.

```
http://www.prosys.fi/downloads.html
```

8 Gateway Service Special Tags

One of the powerful features of the Gateway Service is the ability to monitor itself and other processes on the server computer. In the typical setup the OpenOPC Gateway service is installed on the machine hosting the OPC server we want to read data from. The Gateway service can monitor several system paramters including it's own memory consumption. The special tagnames available are:

```
@SineWave
@SawWave
@MemFree
@MemUsed
@MemPercent
@DiskFree
@CpuUsage
@TaskMem(name)
@TaskExists(name)
```

To read the current CPU usage on the machine running the Gateway (*TESTBOX*) service simply type:

```
opc -H TESTBOX -m open @CpuUsage
```

To read the current memory consumption of the OpenOPC Gateway Service on *TESTBOX* type:

```
opc -H TESTBOX -m open @TaskMem(openopcservice.exe)
```