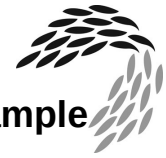
OPCFailover in Action



Explanation through Example

Overview

OPC server monitoring should be simple and at times can appear to be. It is easy for this simplicity to lead us into a false sense of security and can lead to being caught out. Usually problems happen when we're not there. You're lucky if you've never had an overnight run fail on you.

Infrequent events can be particularly challenging especially when running experiments that may not yield a result until much later on. Connecting an OPC client to a server and logging the data being output is fairly straightforward and such a set-up is usually adequate. Occasionally however such a configuration fails to capture the result because of OPC server failure. There can be many reasons for such a failure but it can be as simple as the server being switched off.

The alternative usually requires the time of Administrators or other Engineers to set up proxies for redundancy or further OPC servers for data storage (which could loose connection also). But is this really necessary for just some experimentation work? Also if redundancy is already in place then a set-up should be possible such that if one OPC server is unavailable another one can be used instead without having to go to great pains.

Jemmac Software Ltd took an approach with their product, OPCFailover, which is very straightforward. It allows an OPC client to connect directly to a server, but if the server fails switches the client to communicate with one or more backup OPC servers. The solution does not use proxies of any kind so is very easy to set up.

The purpose of this article is to see how OPCFailover works in practice through experimentation. The reader should feel comfortable setting up OPCFailover so they can evaluate for themselves. Jemmac Software Ltd actively encourages potential users to download the trial version of their software before purchase and test in a real environment. All testing is done using commercially available software preferably trial version where possible.

OPCFailover in Action

The Set-Up

After visiting the Jemmac Software Ltd[1] website and reading the product documentation for OPCFailover the trial version of the software was downloaded. The process was very straightforward, requiring an email address to receive the link for the download. Upon installation I read through the help file to gain further knowledge on the product to determine what sort of test set-up to choose.

A typical OPC set-up would usually involve multiple OPC Servers accessing a single DCS. Reading a data item in such a manner would not show through which path the data had travelled so is not useful for a testing environment.

After reading through the documentation I decided that it would be best to set up a test that would show clearly the mechanics of OPCFailover by making the data source obvious. My test suite would not be connected to a DCS and would require dummy OPC Servers, which would appear to be the same, but give different values depending on their source. Figure 1 shows the how 3 PCs will be configured for testing purposes.

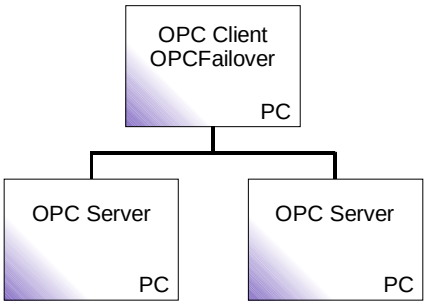


Figure 1 - Test Configuration

Two PCs will be installed with an OPC Server and the same data item set configured[2]. The value of this data item will be different on the two PCs. When we read from an OPC Client[3] on a third PC, with OPCFailover[4] also installed, it should be obvious which OPC Server the data has come from. Trial versions of software will be used so that these experiments can be repeated easily.

The PCs will be named as follows:

- OPC Client / OPCFailover PC – Test Master
- OPC Server PC #1 – Test Node 2
- OPC Server PC #2 – Test Node 3

On *Test Node 2* I will create an OPC data item with the value of "2". Similarly on *Test Node 3* the value will be "3" for the same named OPC data item. This means that if we read from the OPC Client and the value is "2" then we know that the source of the data was *Test Node 2*.

---

[1] Jemmac Software Ltd – www.jemmac.com
[2] MatrikonOPC Simulation Server used for the OPC servers – www.matrikonopc.com
[3] SapphireTrend was used as OPC client and data visualisation – www.sapphiretrend.com
[4] OPCFailover available to trial and purchase from www.jemmac.com

Matrikon Simulation Server is going to be used for the OPC Servers, so the name of the OPC Item ID will be *Bucket.Brigade.UInt4*. The configuration of the OPC Servers can be seen in Figure 2.

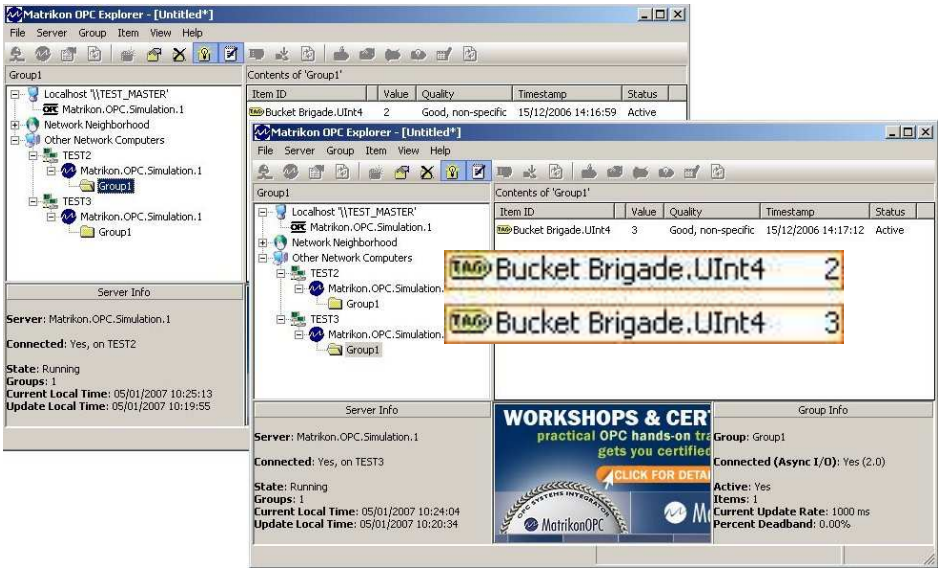


Figure 2 - OPC Server data

To think about how to configure *Test Master*, some explanation of terminology used by OPCFailover is necessary. OPCFailover makes reference to *Failover Groups*, but what does this mean? It is simply a number of OPC Servers that provide redundancy and OPCFailover is to manage. Note that OPCFailover can manage several OPCFailover Groups. The name of the Failover Group is important, as it is the name that the OPC Client will connect to.

To communicate with an OPC Server, its *ProgId* is looked up and its *CLSID* is retrieved. It is through the *CLSID* that the OPC Client then actually communicates. The common analogy for this is looking up someone's phone number with that person's name, where the *ProgId* is thought of as the person's name and the *CLSID* as the phone number. If an error occurs with communication the OPC Client should reconnect by looking up the *CLSID* again from the *ProgId*.

OPCFailover works by creating a new *ProgId* with the name specified by the Failover Group and assigning the *CLSID* to be the same as the required OPC Server's *CLSID*. It then monitors the health of the currently active OPC Server and if there is an error, rewires the *CLSID* of the Failover Group. When an OPC Client tries to reconnect to the OPC Server it will now use the *CLSID* of one of the other OPC Servers in the Failover Group.

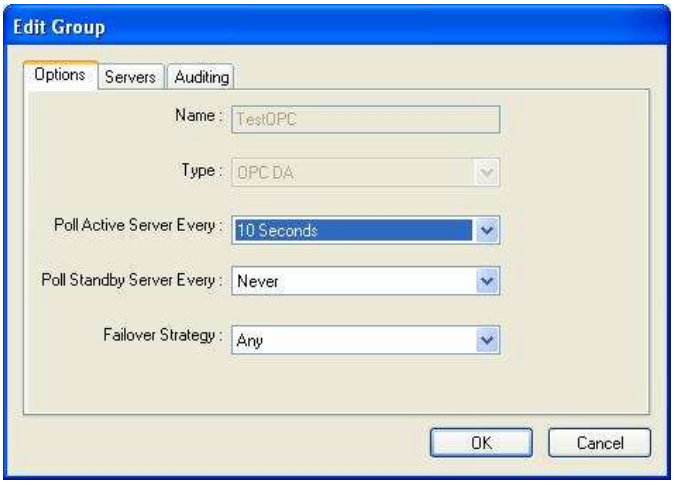


Figure 3 – Group Configuration

The way in which OPCFailover monitors the health of the OPC Servers in the Failover Groups can be adjusted in the OPCFailover Manager. The options are shown in Figure 3 above. The number of threads that are used to monitor OPC Servers can be modified, as well as the polling period for health of the OPC Servers and the strategy to use for selecting the next OPC Server upon failure of the active OPC Server. Note that the number of monitor threads is per instance of the OPCFailover Service and not per OPC Failover Group.

The OPCFailover strategy dictates the manner in which the next OPC Server is to be picked if the current OPC Server fails. The following options are available:

- *Any* - picks the server which responds the quickest with a success. This should have a random result.
- *First Available* will always provide precedence to the servers nearest to the top of the list, which is useful in a Primary, Secondary, Tertiary situation.
- *None* - makes no modification to the Active Server - whether set or not.
- *Ordered* – sets highest ranked server as Active Server whether the current Active Server has a failure or not.
- *Round Robin* - processes the OPC servers in the order that they appear on the screen. This is useful if failure is common and you want it to move through the list and let it settle on a reliable one.

The following table shows the effect on the Active Server when switching on and off OPC Servers when there are 3 in an OPCFailover Group. Note that it is assumed that in all cases the current Active Server is Server 1 (with *Any* this may not be true):

Step	Any	First Available	None	Ordered	Round Robin	Comment
1	1	1	-	1	1	-
2	2 3	2	-	2	2	Stop 1
3	2 3	2	-	1	2	Start 1
4	1 3	1	-	1	3	Stop 2

Table 1 - OPCFailover Strategy Comparison

As can be seen in Table 1, only *Ordered* changes the Active Server in Step 3. This is because it will not only change the server assignment on a failure, but also when a higher ranked server becomes available again.

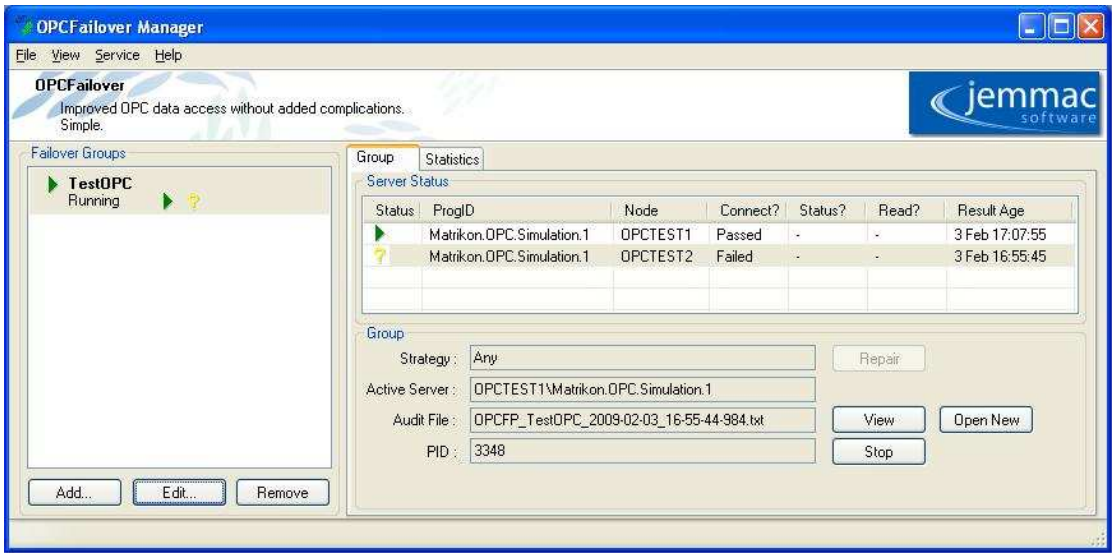


Figure 4 - OPCFailover Manager Configuration

Figure 3 and Figure 4 show OPCFailover Manager as the test will be configured:

- Failover Group named *TestOPC* created containing the OPC Servers from *Test Node 2* and *Test Node 3*.
- Poll Active Server Every – default (10 seconds). Although detection can be set quicker, it is often not necessary to do so.
- Poll Standby Servers Every – default (Never). As *Ordered* is not being used and standby server health monitoring is not necessary, there is no need to poll standby servers.
- Failover Strategy – default (Any). When there are only 2 OPC Servers the choice of failover strategy is fairly irrelevant, although *Any* may be slightly more responsive.

Test Node 2 and 3 will need to be configured so that the DCOM security allows remote launch and access to the user account that the OPCFailover Service is running under (account named OPCFailover by default). For simplicity of the test I will simply be opening up DCOM security so that EVERYONE can remotely communicate. This is not recommended in configuring real OPC Servers, as it is a security risk.

SapphireTrend will be used as the OPC Client on Test Master with plots for Test Node 2, Test Node 3 and TestOPC Failover Group.

At this point I have 3 PCs ready and waiting to start my test. I will keep the test itself fairly simple, performing and expecting to see the following if OPCFailover is working:

- All nodes will be powered up
- SapphireTrend plotting will be started
    - Test Node 2 plot will show value 2
    - Test Node 3 plot will show value 3
    - Test Master plot will show value 2 - indicating a connection to Test Node 2
- Test Node 2 will be powered down:
    - SapphireTrend plots for Test Master and Test Node 2 will stop
    - After a period of time > 60s Test Master plot will show value of 3 – indicating a connection to Test Node 3 and successful OPCFailover
- Test Node 2 will be powered up:
    - Test Node 2 plot will show value 2
    - Test Master plot will remain at 3
- Test Node 3 will be powered down:
    - SapphireTrend plots for Test Master and Test Node 3 will stop
    - After a period of time > 60s Test Master plot will show value of 2 – indicating a connection to Test Node 2 and successful OPCFailover

If OPCFailover does not work I will expect Test Master and Test Node 2 plots to be the same.

Running the Test

The test was carried out as described in the following section and the plots were observed as shown in Figure 5. From the plots it can be seen that:

- Test Node 2 was disabled between 17:09:30 and 17:11:30
- Test Node 3 was disabled between 17:13:00 and 17:15:30
- Test Master switched to Test Node 3 after 90s of no connection when Test Node 2 was disabled
- Test Master switched back to Test Node 2 after approximately 110s of no connection when Test Node 3 was disabled

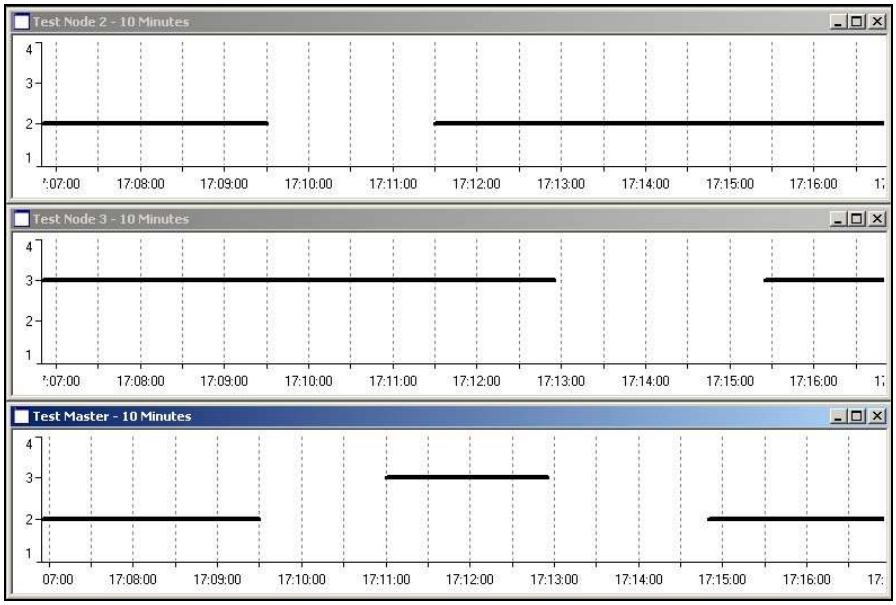


Figure 5 - Test Results

During the period of the test, Test Master was not touched in any way except to take a screenshot after completion. The only interaction with Test Node 2 and Test Node 3 was to disconnect them from the network through Windows Network Configuration.

Observations

It can be seen that the test was successful. As expected the Test Master plot changed value of 2 to 3 and then back again whenever the actual OPC Server that it is connected to fails. Also as expected there was a period of no activity while DCOM/OPC was trying to communicate with the failed OPC Server before giving up. This is outside of the scope of OPCFailover however as it is built-in to the RPC (Remote Procedure Call) mechanism that DCOM and OPC are built on top of and related to the OPC Client communication.

Although this was a simple example it showed very clearly how easy it was to put in place OPCFailover.

One thing to note is that some OPC clients do not try to reconnect with failed servers in a standard manner and so cannot be used with OPCFailover this only occurs if the client fails to perform a *ProgID* to *CLSID* lookup on reconnect however. Going back to the phonebook analogy - if you try and ring a second time starting by looking up the phone number first there will be no problem, if however you remember the number and redial you will not get through because the number has changed.

There are also some OPC Clients that give-up if a communication fault has been found. With these OPC Clients OPCFailover cannot be used, but they are rare to find. It is useful to use the trial version of the software if there is any doubt about compatibility. In fact reproducing the test performed here is a very good way of proving if an OPC Client is suitable for use with OPCFailover, so it can be seen that SapphireTrend is capable.

One thing that an article such as this does not convey is the speed at which it takes to set up OPCFailover. The creation of OPCFailover Groups can be done in minutes and are as simple to create as configuring an OPC client to connect to an OPC server.

In most cases when using OPC Servers that have been written in a robust manner there will be no problem with connection to OPCFailover.

With OPCFailover in place there is now a way to confidently set up ad-hoc configurations with OPC with confidence. In fact with such an easy to use solution there really is no longer an excuse for failure when running short-term trials and experiments over OPC.