# The Cache process node


## *Introduction*

It has been a requirement for some time to be able to store state in WMQI, however since WMQI is aimed to be high performance the state data should be put in memory. However if this is put into memory then it is transient data which can not withstand a broker restart. This process node is a memory cache node when the broker is stopped and restarted the cache data is lost. This cache process node works by putting data into the cache with a key and later getting it back later when require. The putting and getting do not have to be in the same message flow however they do have to be in the same execution group.

## *Installation Instructions*

### Control Center Installation

There are two file types that need to be copied into the WMQI installation directories as follows. <WMQI> is the location of where the product is installed

| | |
|---|---|
| Cache*.gif: | Copy these to <WMQI>/Tool/images |
| Cache.properties | Copy this to <WMQI>/Tool/com/goakes/mqsiv21 |

Now start the control center and import the Cache.xml into it. This can be achieved as follows:

- Clicking on the file button, import to workspace,
- This starts the “Import resources window”, ensure that only “Message flows” is checked,
- Press the “Browse” button and navigate to the “Cache.xml” file
- Press “Import”, this should then give you a confirmation that 1 resource has been imported.

### Broker Installation

Here’s the easy bit and should not take to long. Simply follow the instructions below to get the cache process node into the run time broker system.

- Stop all the brokers on the server machine,
- Place the supplied jar file into the <WMQI>/jplugin directory,
- Restart the brokers.

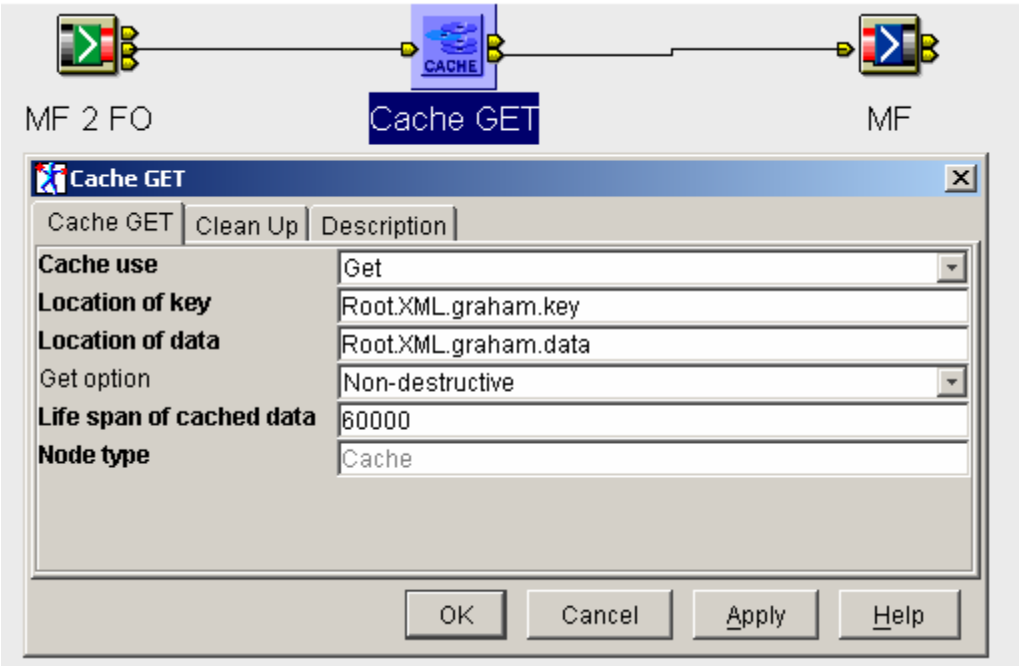
## *Usage Instructions*

The one cache node can be used to either put data into memory or get data back. When getting or putting, a key has to be used. You will never supply the data or key you want to store in the properties, but instead a pointer to them via the normal dot notation.

On a put the data must be supplied via the location of data property. The life span of cache data should also be supplied. This shows how long in milliseconds (default is 60000, hence 1 minute) the cache is good before the data is stale. If the data goes stale then it can not be retrieved and it will be cleaned up (removed) from the cache to keep the memory as small as possible.

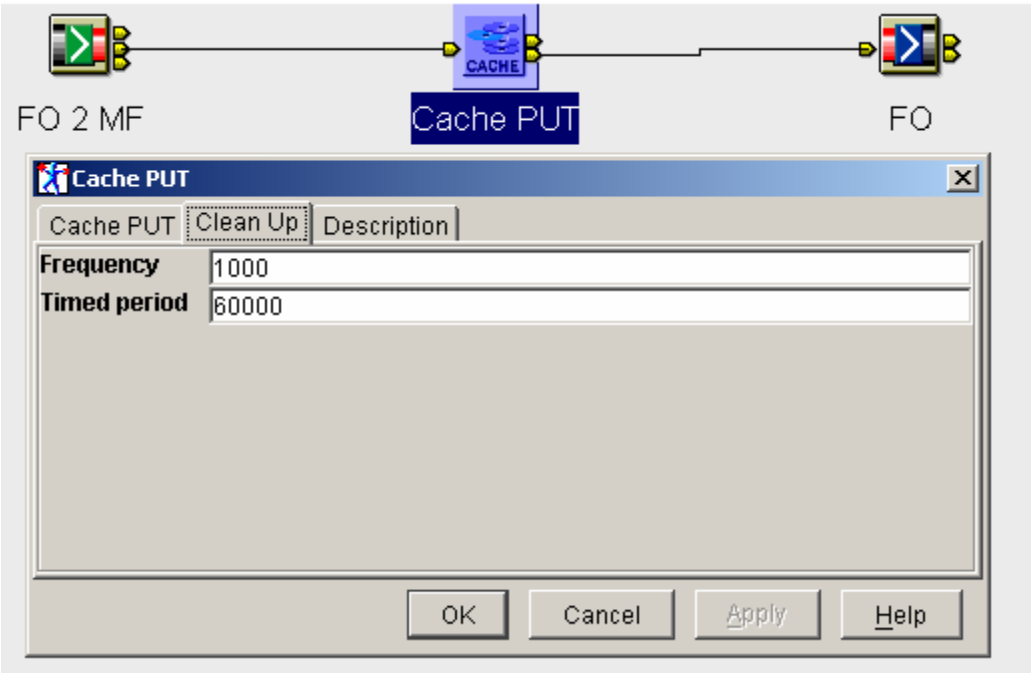
On the get the data will be retrieved and put into the location of data property. Also, when getting, if the destructive get option is used then the data will be removed from the cache, hence it's only good for one get.

Below is an image showing the setting of the properties in the cache process node.




If data times out by getting to the end of its life as defined by "Life span of cached data" then it becomes stale. Part of the function of the cache process node is to clean up stale data. There is a tab to set properties and define how this is to be performed. The clean up process can be set to take place within a set timed period using property "Timed period" which is measured in milliseconds (by default every minute). Clean up can also be set to run dependant on the number of times the cache process nodes are entered. This is shown by the property called "Frequency". The default is 1000 which means enter the cache process node a 1000 times before cleanup takes place.

It's also important to realize that each cache process node has its own "Clean Up" tab and that internally the last cleanup time and frequency count is stored for all cache process nodes. Each time cache process node is entered it compares the cleanup properties with the global last cleanup time and frequency count.



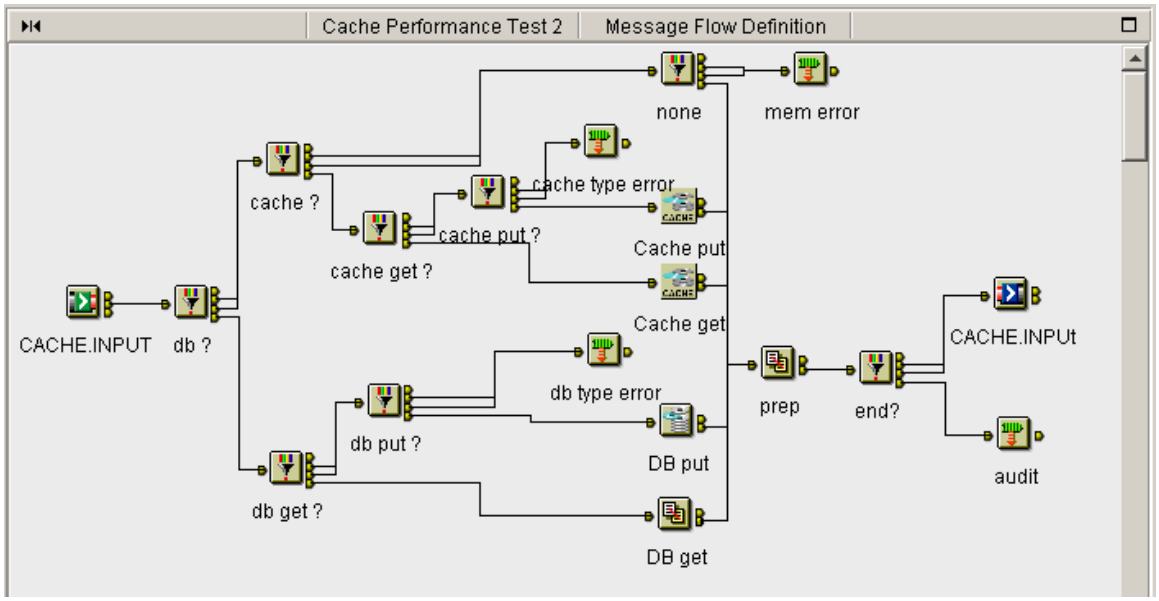

## *Comparison*

There are other ways to store data. For example a database could be used, or probably the MQGet supportpac. The following table will help to determine which state mechanism would be the best for you.

| | DB2 | MQGet | Java Node |
|---|---|---|---|
| **Mainframe Cross Plex** | Yes | Yes | No |
| **Admin required** | Yes | Yes | No |
| **IBM Supported** | Yes | No | No |
| **Portable to mainframe** | Yes | Required | Yes |
| **Performance** | | | Fastest |
| **Persistent** | Yes | Optional | No |
| **Search flexibility** | High | Medium | Low |
| **Use in compute node** | Yes | No | No |

## *Performance of Cache node v DB2.*

To see the difference in performance a simple example was put together to carry out an inserts into either a cache or database, then read it back. Since the database would be using the file system it should be noted that the file system cache on NT is left on to help the database. The following message flow was used to obtain the results and then the following table shows the amount of time taken to process 10000 non persistent gets and puts.




| Size in Bytes | None MM:SS.m | Java Node | DB2 |
|---|---|---|---|
| 100 | 34.4 | 1:41.3 | 2:44.0 |
| 1000 | 27.3 | **1:20.7** | **12:56.5** |
| 3000 | 35.6 | **1:51.7** | **36:06.0** |

## *Limitations*

- The cache data is only available in execution group it was created.
- The process node has been only been tested with string data
- Cleaning up of memory can only occur when the cache node is entered.
- There is a limit on the size of memory a process can grow to so there is a limit on the amount of data that can be stored in memory.