IA91 SupportPac: Cache Nodes

IBM Information Bus V9 and V10

1. Introduction

There has been a need for some time from some to be able to store state in IBM's integration broker now called IBM Integration Bus or IIB for short. However, since IIB is aimed to be high performance, any state data needs to be cached in memory not written to disk via a database node. There are no IBM Primitive processing nodes in IIB that give this functionality. If data **is** put into memory, then the data is obviously transient and cannot withstand a broker restart.

This SupportPac now provides eight IIB nodes and six ESQL functions which provide the functionality of an in-memory cache. This cache nodes works by *put*ting data into the cache with an associated key and later *get*ting it back using that key. The putting and getting do not have to be in the same message flow nor even on the same broker, if configured properly.

REMEMBER: The in-memory cache is lost if the broker or execution group is restarted.

There are additional facilities that this SupportPac brings around this caching mechanism. Following is a list of these functions:

- Allow for different named memory caches to easier separate items to be saved
- Provide two mechanisms for putting data to and getting data from a memory cache:
    - Node based (CacheGet and CachePut nodes including new Named versions)
    - ESQL based (putIntoCache, getFromCache and createFromCache ESQL procedures including new Named versions)
- The ability to store data locally (e.g. within an execution group), within a broker (across execution groups) or within a domain (e.g. across one or more brokers).
    - WebSphere MQ Real-time transport is used to communicate between brokers for WMB V6 and V6.1
    - WebSphere MQ V7 Publish/Subscribe is used between brokers for WMB V7 and V8 and for WMB V6.1 brokers configured to use MQ V7 support instead of the real-time nodes
- A sample cache initialization program has been supplied which can read a database and load that information into a broker domain.
- The CacheConfig node and NamedCacheConfig node:
    - Allows data to be read from a database or property file to initialize a cache
    - Allows the Real-time transport to be configured
    - Allows for tracing of the Cache for problem resolution.
- The CacheReplay node and NamedCacheReplay node allow bulk data to be retrieved from a cache. This can be replayed using different sort categories (like key, data), sort orders (like alphabetical, reverse numerical), between two particular times, etc.

2. Notes for IIB V9 and beyond

With the advent of IBM Integration Bus (IIB) V9 comes the built-in functionality provided by the new Global Cache (or GlobalMap). If you are looking for new cache functionality, I would recommend you pursue that path rather than installing IA91. If you are a current IA91 user looking to upgrade to IIB, this is still available, but you may also want to look at converting to Global Cache.

Global Cache does have similar functionality to IA91:

- Ability to store items in memory
- Ability to share items across execution groups and/or brokers
- Ability to expire items from memory
- Ability to store items in separate namespaces

Global Cache is available in the Mapping Node or via Java code. Although the cache functionality isn't available "out of the box" in ESQL, it is very easy to write Java functions to access the cache, similar to the ESQL functions provided in IA91.

Global Cache is automatically available across execution groups (servers) within a broker (node). Global Cache items also remain in memory as long as the broker is running; IA91 caches are lost if the execution group restarts.

Global Cache can be configured to share across brokers. Global Cache does not require MQ to share items across execution groups or brokers; IA91 still requires a local queue manager for this ability.

Global Map does not provide certain functionality that IA91 provides:

- Nodes to access the cache
  - You must create your own JCNs if desired
  - Most applications would be simpler/faster using Java functions
- Ability to extract all items currently in the cache
  - Few applications need or use this capability in IA91
  - See Stack Overflow for a solution for Global Map
- Ability to pre-load cache from database/file
  - You must write your own cache loading code

3. Change Log

April 2017

New! Added "service:" URL support within CacheConfig node for using JDBC configurable services in V7 and beyond

New! Added support for using JVM properties ($propertyName) for the values used in Cache Config nodes for these fields: Trace file path, URL, Userid & Password fields.

New! Replace new installation process for Toolkit installation more compatible with V9 and V10

Ensure that CachePut nodes use a default value for Life of Data in case string is not numeric

Tested against IBM Information Bus V9 and V10

Added Prerequisite section to document required software and compile workaround

Clean up tracing outputs

Update this document to remove most older version specific information.

October 2011/January 2012

Create new installation process for Toolkit installation more compatible with V7 & V8

Added CacheNode-based ESQL routines for compatability with original V5 IA91 ESQL versions

Fixed ESQL routines to try to detect null keys/data fields better

Fixed trace output in several classes

Update instructions for InitCacheSample program

Ensure current ESQL functions work consistently across V6 and V7; create new V7 specific versions of ESQL get/create allowing NULL to be stored in Cache

April 2010

Fix NullPointerException bug in Publisher

Eliminate duplicate databases/file initialization operations on same source & cache

Update CacheConfig pre-loading sections

Remove source from package and source code install instructions from this document

March 2010

New! Add file: url support to CacheConfig and NamedCacheConfig nodes for reading properties files in addition to JDBC databases

Consolidate to single project source for V6, V6.1 and V7 versions

Consolidate installation zip files to one for all broker runtimes, and two for toolkits (V6 uses different directory structure than V6.1 and V7)

Simplify installation instructions

Auto-detect WMQ V7 and if found, determine if WMQ Pub/Sub is being used instead of Broker Real-Time nodes

Fix published messages to be non-persistent (Thanks to Neal Mulvenna for finding this)

Eliminated lag in updating local cache when publishing messages

Update cache initialization program to use Unix-style parameters (e.g. –v value) instead of positional parameters to be able to support both pub/sub engines

Remove V5 support

December 2009

Update doc to include:

- Status values returned by ESQL called routines
- Memory consideration for StackOverflowError (Thanks to Neal Mulvenna for finding this)

October 2009

Update for Message Broker V7:

- Remove Real-Time information from CacheConfig nodes (Real-Time node no longer supported by V7)
- Change publisher code to use MQ v7 publish/subscribe

March 2008

Update V61 toolkit install file to add missing icon files

January 2008

Update installation instructions to reflect CLASSPATH changes needed for broker runtime

Update installation instructions for needed rights to access the files in Unix

Package V61 versions of the installation files

Add installation instructions for Message Broker V6.1

Update node help toc.xml file to use correct linkage in V6

May/June 2007
Bug fix to CacheConfig set methods to output new values properly
Changes to CacheConfig nodes to:

- Connect to the real-time flow if node set for Broker Domain caching (in order to support ESQL and/or CacheReplayNodes receiving Broker Domain cached items)
- Update the local cache during database initialization runs even if unable to publish for Broker Domain caching
- Retry the database initialization if unable to publish or unable to access database, and to continue until it works as desired
- Cancel all retry tasks properly when node is deleted from EG

Rework tracing mechanism to avoid ArrayOutOfBounds issues by converting to use ThreadLocal variables and general updates to clarify where trace entries come from
Fixed exceptions thrown from CacheGet/Put nodes to refer to correct message file
Changed cacheToC.xml for V6 to have help files appear in local help
Update doc to reflect hardware used for performance tests
Update doc to attempt to clarify Broker-Domain caching in various places
Update doc to add section on how to read trace file output

May 2006
Bug fixes for ReplayNode to Environment and LocalEnvironment trees
Add cleanup checks to ESQL functions and ability to change cleanup frequency from ESQL

March 2006
Bug fixes for unlimited timeouts & cache config from database
Allow -1 for recache settings to avoid recaches if desired

January/February 2006
Add Named caches
Reorganize code to allow ESQL functions to work with V6 classloader changes
Replace manual testing with JUnit tests
Fix bugs introduced in prior changes
Allow for constant key names in CacheGet/Put nodes
Fix CacheConfig nodes to only recache as needed when a message comes into it
Standardize installs across V5 and V6

December 2005

Add ESQL functions to access cache
Add documentation under User Defined Nodes in Message Broker Help
Fixed minor bugs found during performance testing
Ensured cache tracing is off by default
Removed dependency for JMS-related jar files in CLASSPATH unless Broker-domain caching is used
Support for V6:

- For V6, removed some deprecated MessageAssembly functions in favor of new versions
- For V6, updated toolkit extensions to be compatible with V6

Update this documentation with some clarifications and performance testing under V5

March 2005

Get/Put Nodes changed to allow Environment and LocalEnvironment element paths in addition to Root.
All for non-expiring data items (-1 in Life time of CachePut)
Some error handling has been improved.
New packaging of components for easier installation.

4. Prerequisites

IA91 does require a local IBM MQ queue manager to be installed. This is only an issue for IIB V9 and V10, which no longer require a local queue manager unless certain nodes are used.

IBM Integration Broker V9 and V10 require a build-time work around in order for the nodes in IA91 to work correctly. See below for the work around. If you only use the ESQL functions and do not require CacheConfig nodes for any reason, you may not need to use the work around, but this has not been tested.

The following link explains the issue:

http://www-01.ibm.com/support/docview.wss?uid=swg1IC92824

However, when testing IA91 in V9 and V10, the fix above does not fix the issue, and using the workaround is required.

Workaround

Checking the "Compile and in-line resources" box on the Prepare tab of the BAR file, and rebuilding and redploying the changed BAR file should fix this problem.

5. Installation Instructions

In the version-specific installation instructions below, the apparent default values for the variables appear in Table 3.1 below. In the instructions, these variables will appear in **boldface.** Modify the instructions for your particular operating system as needed (e.g. change \ for Windows to / for Unix-based systems.)

Other Operating Systems – Broker only	
Install_dir	/opt/ibm/mqsi/9.0.0.*
workpath	/var/mqsi
IIB Version 9 for Unix	
Install_dir	/opt/IBM/mqsi/9.0.0.*
workpath	/var/mqsi
IIB Version 9 for Windows	
Install_dir	C:\Program Files\IBM\MQSI\9.0.0.*
workpath	C:\ProgramData\IBM\MQSI
IIB Version 10 for Linux / Unix	
Install_dir	There is no default installation directory anymore. The install can be local to a user or installed by root in a shared location. You must know where it is installed. The server components are under this directory in the **server** subdirectory.
workpath	The default global workpath is: /var/mqsi
IIB Version 10 for Windows	
Install_dir	C:\Program Files\IBM\MQSI\10.0.0.*
workpath	C:\ProgramData\IBM\MQSI

Table 5.1: Default variable values

Licensing

You **must agree** to the license included in the zip file (under the license sub-directory) before installing and using IA91.

Toolkit Installations

Unzip the ia91.wbimb.zip file and extract the CacheNodeInstallToolkit.zip file. Start the message broker toolkit. The remaining instructions are then version specific.

Individual Toolkit Installation – WMB V8, IIB V9 and IIB V10

The toolkit can be installed by each user using IA91 separately.

1. Go to the Help menu.
2. Select Install New Software…
3. Click Add…
4. Click Archive…
5. Type in or navigate to the location of the CacheNodeInstallToolkit.zip file and click Ok
6. Click Ok again (you can leave Name blank)
7. CacheNode should appear in the box below. Check the box next to it and click Next
8. Click Next again
9. Accept the license agreement and click Finish
10. When it prompts about installing unsigned content, click Ok.
11. It should prompt you to restart the toolkit. Click Restart Now.

Shared Toolkit Installation – WMB V8 and IIB V9

For shared toolkit installations, it can also be installed by an administrator, which makes the feature available to all users. Start the toolkit as the administrator and use the same instructions above.

Shared Toolkit Installation – IIB V10

For shared toolkit installations, it can also be installed by an administrator, which makes the feature available to all users. Use the instructions from the IIB documentation in the section titled “Installing plug-ins into the IBM Integration Toolkit for multiple users”.

Broker Installations

Unzip the ia91.wbimb.zip file and extract the CacheNodeInstallBroker.zip file. Unzip this into **install_dir**. This will install the runtime plugin (in **install_dir**\jplugin), the runtime error messages (in **install_dir**\messages), and the new shared class files (in **install_dir**\shared-classes).

Copy the CacheShared.jar file from **install_dir**\shared-classes to **workpath**\shared-classes for each broker[1]. If this is not done properly, the ESQL functions will not see the same cache as the Cache nodes.

Ensure that the broker has access to the files. For example, on Unix systems, you may want make these files group- and world-readable (via a chmod 755 command).

Finally, check to ensure that the following WMQ jar files are added to the CLASSPATH variable for the user starting the broker (this should be done automatically):

- **wmq_install_dir**\java\lib\com.ibm.mqjms.jar
- **wmq_install_dir**\java\lib\jms.jar

Domain Caching Note

If you plan to use Broker Domain caching, please see the note at the end of section 9 to complete the installation.

[1] This is preferable to adding this library to the CLASSPATH environment variable in V6. If you prefer, you can add this file to CLASSPATH. You may also modify mqsiprofile to add this file to CLASSPATH just before starting the broker.

6. Using the CachePut Node

Basic Section

The CachePut node puts data into the cache. The scope of the cache, by default, is the execution group, which is a local in-memory cache; the scope can be changed to a broker domain cache, which is discussed in detail in section 9. This data is stored and later retrieved using a key that uniquely identifies the data within the cache. The CachePut node needs both the key and data to be specified as elements within your message tree. Only Root, Environment and LocalEnvironment are valid starting element names. For the *key field only*, anything not starting with Root, Environment or LocalEnvironment is treated as a constant. Input to the node's in terminal is not altered but passed unchanged to the out terminal.

For example, "Root.XML.go.key" could be the path to where actual key value is stored, and "Root.XML.go.data" could be the path to where the data is stored. An example of such an XML message would look like Figure 6.1.

```
<go>
      <key>TheKey</key>
      <data>The data to store</data>
</go>
```

Figure 6.1: Example message containing a key and some data

The life span of cache data can also be supplied. This shows how long, in milliseconds, to keep the data before it becomes stale. The default time is 60 000 milliseconds, or 1 minute. If the data becomes stale, then it cannot be retrieved and it will (eventually) be removed from the cache to keep the total memory used for the cache as small as possible. A value of **-1** indicates that the item never expires. See Figure 6.2 for an example of these properties.

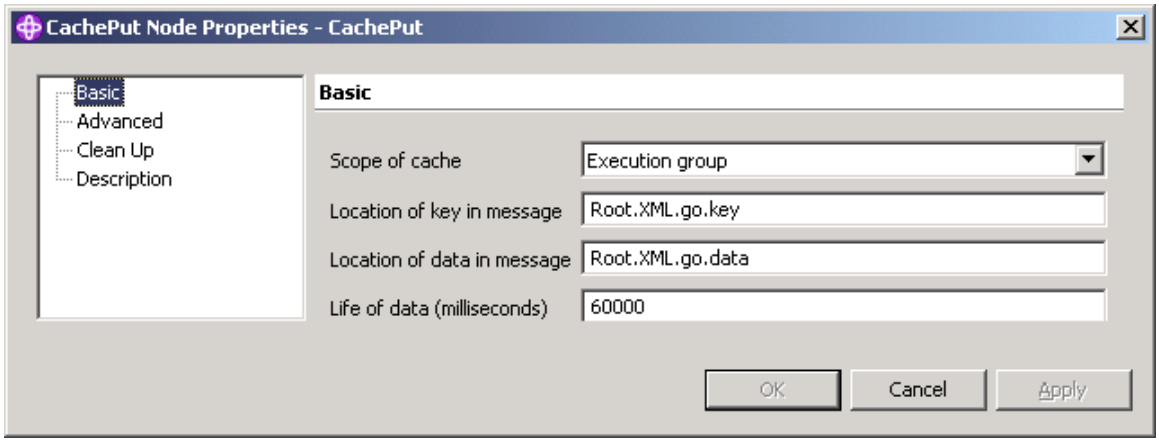

Figure 6.2: Example CachePut Basic properties

Advanced Section

By default, if either the key or data path cannot be found in the message tree, the CachePut node will throw an exception and will be handled the same as any other exception by Message Broker (e.g. propagating back through the nodes until a Catch or Failure node is wired). Checkboxes in this section allow for this behavior to be modified allowing execution of the flow to continue. If the key is NULL, then a null string will be used for the key; if the data is NULL, then NULL will be used for the data. See Figure 6.3 for an example of these properties:

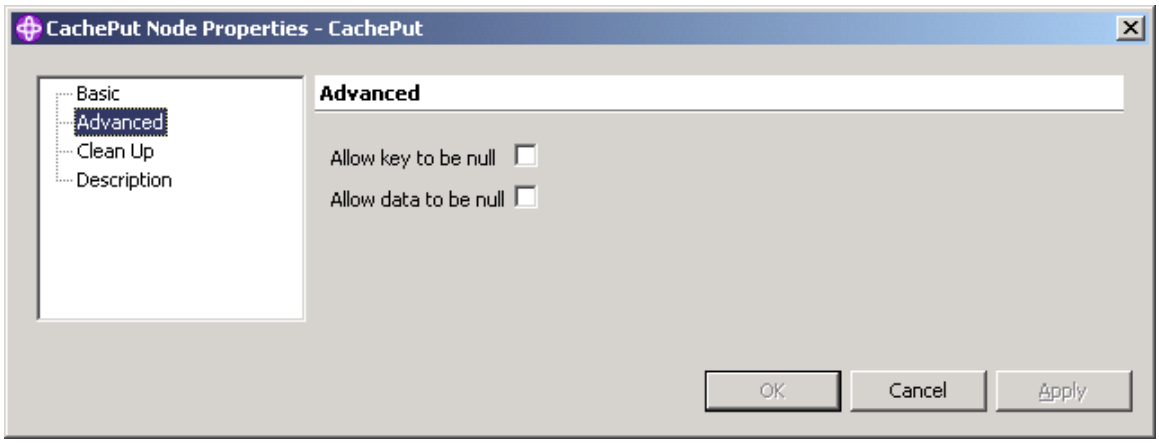

Figure 6.3: Example CachePut Advanced properties

Clean Up Section

By default, the cache will be checked no more than every 60 000 milliseconds, or every 1 minute, or every 1000 times any CacheGet or CachePut node is invoked, in order to delete stale cache data. See Figure 6.4 for an example of this property. Note that the cache **must** be accessed via a CacheGet or CachePut node to have the cleanup occur.

A value of **-1** in either field means not to use that field as criteria to cleanup (e.g. do not cleanup based on time if Timed frequency is -1, but only clean up based on invocations). Very small values will cause cleanup to run very often, which may negatively impact flow performance. A value of 0 will cause cleanup to run for every message. Cleanup also happens before any other processing, meaning it will run even if the node takes an exception.

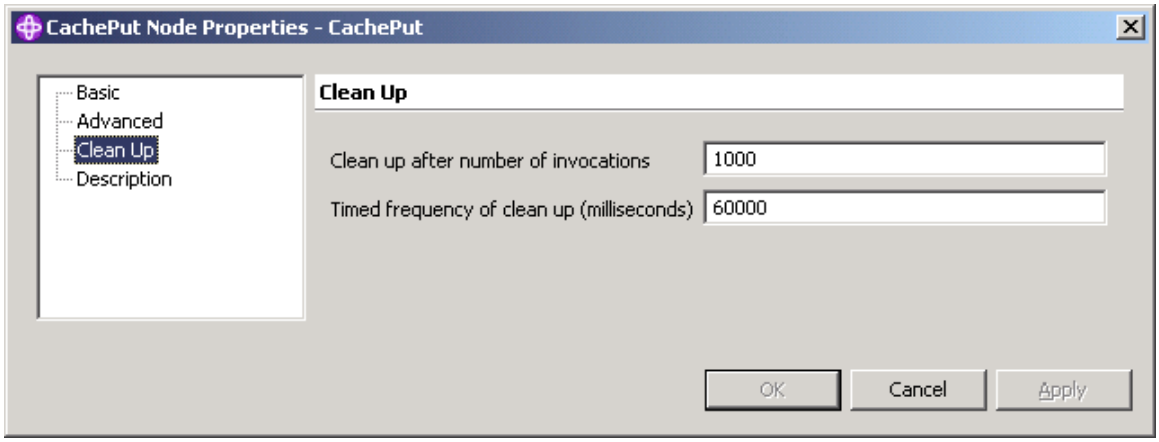

Figure 6.4: Example CachePut Clean Up properties

7. Using the CacheGet Node

Basic Section

The CacheGet node retrieves data from the cache. The scope of the cache, by default, is the execution group, which is a local in-memory cache; the scope can be changed to a broker domain cache, which is discussed in detail in section 9. This data is retrieved using a key that uniquely identifies the data within the cache. The CacheGet node needs both the key and data to be specified as elements within your message tree. Only Root, Environment and LocalEnvironment are valid starting element names. For the *key field only*, anything not starting with Root, Environment or LocalEnvironment is treated as a constant.

For example, "Root.XML.go.key" could be the path to where actual key value is stored, and "Root.XML.go.data" could be the path to where the data is to be stored. An example of such an XML message would look like Figure 7.1. Other than the new output value, input to the node's in terminal is not altered but passed unchanged to the out terminal.

```
<go>
      <key>TheKey</key>
      <data>The data replaces this string</data>
</go>
```

Figure 7.1: Example message containing a key and location for data

The retrieval from the cache can be non-destructive, where the data is left in the cache after retrieval, or destructive, where the data is removed after retrieval. Note: The data retrieved will **always** replace any value already in the data element specified in the message tree, and if the data element does not exist, it (and the path to it) will be created. See Figure 7.2 for an example of these properties.

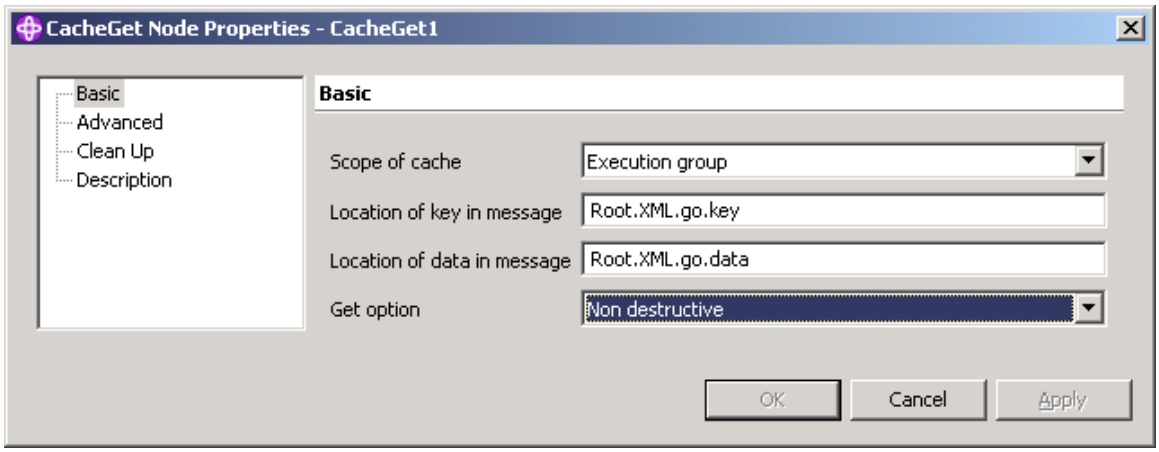

Figure 7.2: Example CacheGet Basic properties

Advanced Section

By default, if the key cannot be found in the message tree, the CacheGet node will throw an exception and will be handled the same as any other exception by Message Broker (e.g. propagating back through the nodes until a Catch or Failure node is wired). A checkbox in this section allows for this behavior to be modified allowing execution of the flow to continue. In this case, if the key is NULL, then a null string will be used for the key.

Also by default, if the key is not found within the cache (i.e. the data went stale), the data element will simply be set to a null string and execution will continue. Another checkbox in this section can be unchecked to cause the node to throw an exception if the key cannot be found. See Figure 7.3 for an example of these properties.

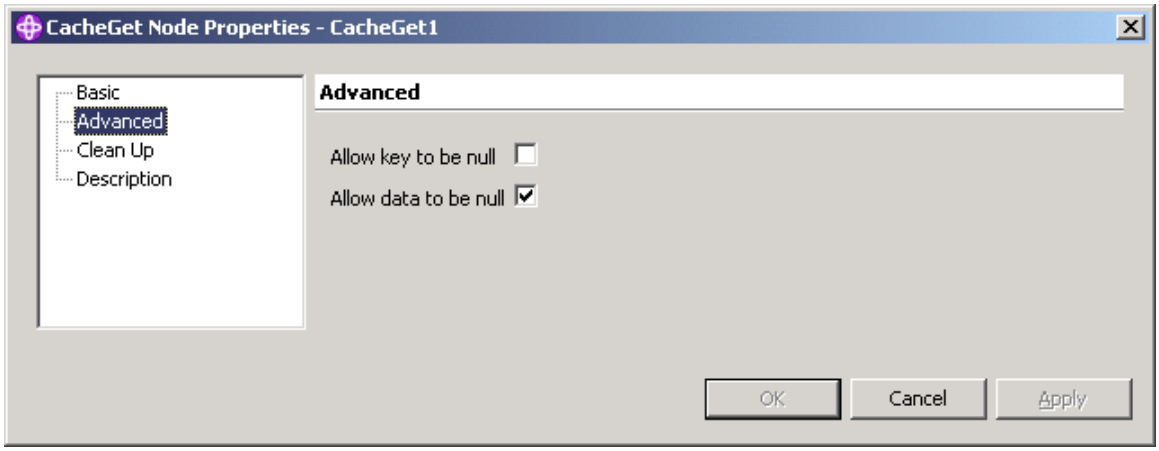

Figure 7.3: Example CacheGet Advanced properties

Clean Up Section

This is the same as the CachePut Clean Up section.

Note: Each CacheGet or CachePut node has its own "Clean Up" section, and each time either cache node is entered, it compares its clean up properties with the cache's last clean up values. This means that if different settings are used within different nodes in the same execution group, or even the same flow, cleanup can happen more or less frequently than desired.

8. Using Named Caches

It is now possible to name a cache to get more separation between cached values and perhaps even better performance when working with large amounts of cached data. For example, you could have an 'item' cache just for shipping weight data, with the key being the SKU, then have a 'sale' cache containing new prices just for items on sale, with the SKU also being the key.

Before this change, you could use a naming convention to achieve the same results, with a little difficulty. For example, you could have used keys "item_xxx" and "sale_xxx" where xxx is the SKU. However, if you have thousands of items, even looking up the sale item price in this situation could be expensive. With the named cache, the sale lookup should be much faster since that cache would, in theory, be much smaller.

By default, the CacheGet and CachePut nodes work on a cache named "default". The NamedCacheGet and NamedCachePut nodes have a Cache Name field added to the Defaults section, allowing you to specify the name of the cache to use within this node. This is treated as a string value; it cannot be in a message tree like they key or data fields using these nodes. The name defaults to "default", meaning that a NamedCache node works on the same cache as a regular Cache node unless changed. The name is case-sensitive, so "mydata", "MyData" and "MYDATA" are each a separate cache.

Except for storing or retrieving data using the specified cache name, the NamedCache nodes work exactly like the regular cache nodes. See the chapter for the corresponding Cache node for details. Clean-up is done only on the cache named.

Throughout this documentation, terms such as "CacheGet Node" or "CachePut Node" generally mean either the original or the named version of the node may be used at your discretion.

Internally, each Cache, including default, is a separate object, which is stored in a master Cache by its name. There is no chance for the same key across caches to result in the same cached item (unless the user has actually stored the same value in both places). The cache name is also used for broker domain level caching as discussed in section 9.

9. Broker Domain Cache

It was noticed that a serious limitation of the first version of the Cache nodes (developed for WMQIv2.1) was that the data in memory is tied down to a specific data flow engine (i.e. Execution Group).

IIB uses the high performance pub/sub engine included in IBM MQ. This is used by the Cache nodes to send data (or remove data) between data flow engines in a broker and to any interconnected queue managers, thus implementing a Broker Domain level cache.

The following setup items must be done to enable Broker Domain caching:

- Ensure IBM MQ Java jar files are available to IIB
- Ensure proper IBM MQ settings
- Alter Cache Node settings
- Optional: Share across brokers
  - Use IBM MQ connection capabilities
- Recommended: Add CacheConfig node to support CacheReplay or ESQL

Required set-up: IIB MQ Java jar files

The use of the Broker Domain cache requires that particular IBM MQ Java jar files are specified in the CLASSPATH environment variable when the broker is started. Apparently, some JVMs have limits on the length of this variable, which can cause the cache nodes to take exceptions at run-time. The minimum jar files needed are listed below. MQ_JAVA_LIB_PATH refers to the java\lib directory under your WebSphere MQ installation. Reminder: If you have to change CLASSPATH, you must restart the broker to pick up the change.

IBM MQ files needed:

MQ_JAVA_LIB_PATH\com.ibm.mqjms.jar
MQ_JAVA_LIB_PATH\jms.jar

Ensure proper IBM MQ settings

The IBM MQ queue manager must be setup for publish/subscribe to work properly. This should be the default value since MQ V7.5. IIB will also check this upon starting the broker and alter the setting as needed.

- Displaying the current value via runmqsc:
  dis qmgr psmode
- Enabling the IBM MQ pub/sub engine via runmqsc:
  alter qmgr psmode(enabled)

See IIB and IBM MQ documentation for complete details.

Required set-up: Cache Node settings

The final setup step is to ensure that your Cache node has its Scope set to "Broker-Domain". This will connect the execution group to the real-time flow above and allow it to receive data from other execution groups.

Only the CacheGet, CachePut and CacheConfig nodes (and their Named counterparts) have this Scope setting. Using this setting may also affect the execution of these nodes however:

- Using Broker-Domain on a CachePut node will cause the same key/value to be sent to all other execution groups (in the same broker domain).
- Using Broker-Domain on a CacheGet node which is doing a Destructive get will cause the same key/value to be deleted in all other execution groups; using this with a normal get has no effect.
- Using Broker-Domain on a CacheConfig node will cause any key/value read for cache initialization or refresh from a database to be sent to all other execution groups

Optional set-up: Share across brokers

Use IBM MQ connection capabilities

To share cache data across brokers, you simply need to interconnect the queue managers to allow the IBM MQ pub/sub engine to send messages between queue managers. See the IBM MQ Pub/Sub guide for more details. There is no need to setup topics.

Recommended set-up: Add CacheConfig Nodes

If you are using a CacheGet node to simply read a value from a Broker-Domain cache, you can set the Scope as indicated above and the node will receive items from other execution groups. However, the CacheReplay nodes (see section 10) do not have this setting.

Get and Put calls from ESQL (see section 12) do have this setting, and if used, will cause the execution group to connect to the real-time flow if it is not already connected. However, this usually happens too late – values that should have been received were not because this execution group was not connected to the real-time flow to receive them.

The simplest way to ensure that ***any*** execution group will receive data from other execution groups is to add a fresh CacheConfig node with its Scope changed to Broker-Domain to your flows using caching; this node does not even have to be connected into the flow in this instance. This technique should be used to ensure that CacheReplay nodes or ESQL calls properly receive data from other execution groups.

Data types and Broker Domain caches

Although local caches can, in theory at least, hold any type of data, Broker Domain caching requires that the data can be represented as a simple string. The messages between brokers are currently sent via JMS text messages, so the data must be sent as a string. So if the local cache is working fine, but the Broker Domain version returns values like "B@77e1b346", then the data cannot be represented as a simple string. For example, the ESQL function ASBITSTREAM does not return a CHAR, it returns a BLOB, which can (apparently) be stored locally with no problems but cannot be sent to another broker at this time. The solution is to CAST the value to CHAR before storing it into the cache.

Final notes on Broker Domain caches

All caches are still essentially local to the execution group, and any Get happens only against this in-memory cache. The sharing mechanism simply allows any Put operation to also publish the item out to any execution group or broker interested in receiving it. A Destructive Get is still a Destructive Get against the local cache, followed by a published Delete operation to the other execution groups or brokers.

It does mean there will be a slight lag for other execution groups and brokers to receive and process these messages, meaning values may not be instantly visible or instantly removed as happens where the cache operation is initiated. This lag can be exacerbated by network issues or other processing, and now with MQ V7 pub/sub, even things like slow channel starts and transmission queue depths can be a factor.

If you are using Broker Domain caching and are not getting the results you expect, it is typically because the execution group is not receiving the published items. Enabling tracing in the CacheConfig node will usually assist in debugging this situation.

Caches are also not refreshed automatically at startup from other broker domain caches; therefore, if an execution group is restarted, its cache is cleared and must be reloaded, either via a CacheConfig node or other custom startup mechanism.

10. Using the CacheReplay Node

The replay function provides a mechanism to extract any or all data that is currently in the cache and added to a message. This is a very powerful function which allows data to be replayed using any or all of the following criteria:

- Items created between a start and end time
- Keys starting with particular characters
- Sorting the extracted data by key, data, create time, using either ascending or desending order, and treating the data as alphabetic or numeric
- Limiting the number of items in one output message (“chunk size”)
- How the output is to appear in the message

The above options can be used with each other hence giving a very power replay capability. The options can also be changed easily at run-time using an input message to the node.

Basic Section

The only value that must be specified is a path within a message tree. This path is used for two purposes: 1) The starting path within the message tree to be used by the over-rides section; and 2) The starting path within the message tree where ultimately the results will be inserted. Paths starting with Root, Environment and LocalEnvironment are valid message trees. Figure 10.1 shows the Basic configuration properties of the CacheReplay node.

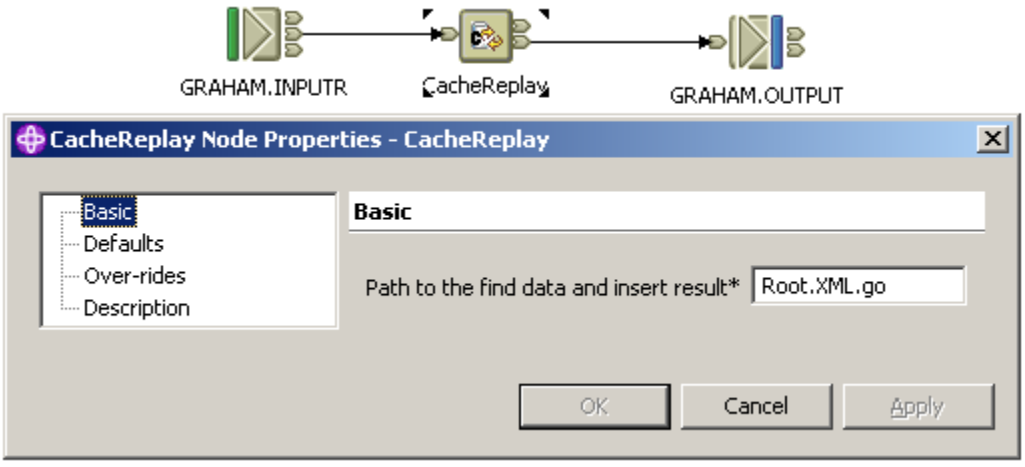

Figure 10.1: Sample screen for setting the Basic properties of a cache replay node

Defaults Section

This section specifies the default values used to replay the data if the Over-rides section has not been set, or if the override keys specified there are not found in the input message:

- **Start Time**: Earliest record (by put time)
- **End Time**: Latest record (by put time)
- **Key start**: The characters the key has to start with; if blank, all keys will be used
- **Sort category** What should be sorted (no sort, key, data or put time)
- **Sort order** How should the resultant set be ordered
- **Chunk size** Number of records between message propagations (this does not apply when replaying to Environment or LocalEnvironment)
- **Format style** How to output the records

Figure 10.2 shows the Defaults configuration properties of the CacheReply node:

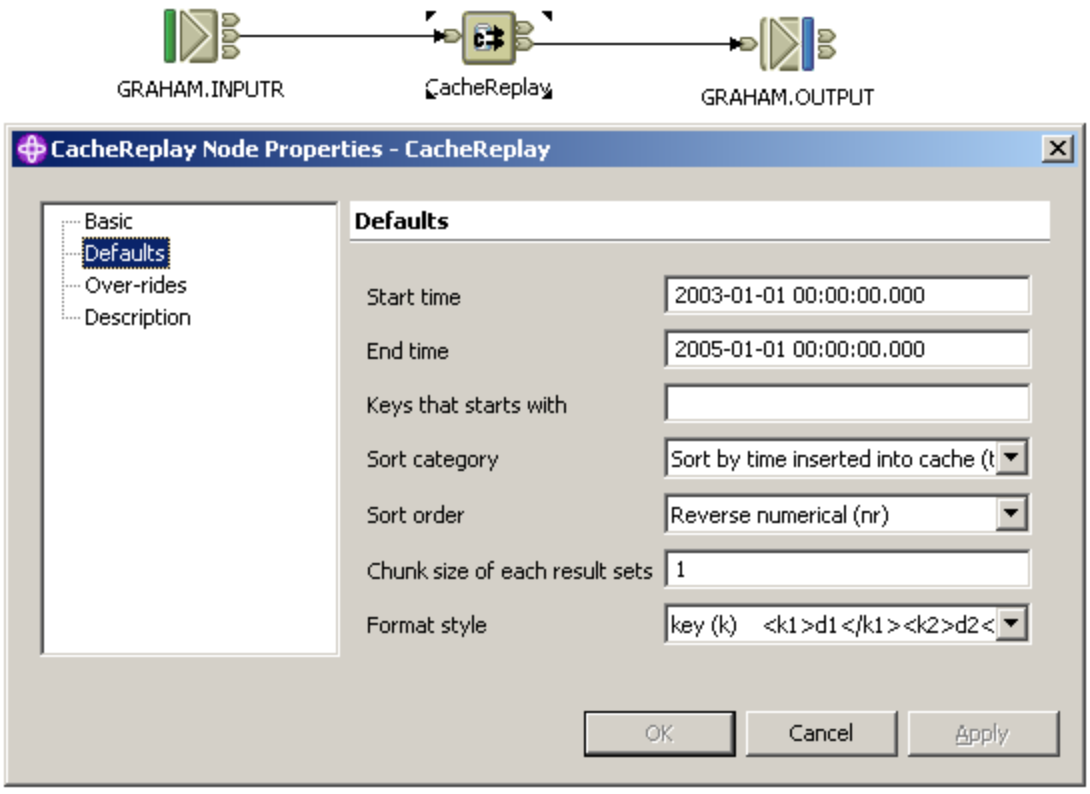

Figure 10.2: Sample screen for setting the Defaults properties of a cache replay node

Message output format

The message tree specified will be augmented to include the results of the replay. These will appear under a "results" element under the path specified in the Basics section. This will be created if it does not exist. Any existing results element will be appended with the replay data. (This may work best under an XML or XMLNS message). Figure 10.3 shows the general format of the output, including the various output format options.

```
<go>
      ...{existing items}...
      <results>
            <!-- This is key data format  -->
            <key>xxx1</key><data>yyyy1</data>
            <key>xxx2</key><data>yyyy2</data>
            <start>1</start>
            <end>2</end>
            <size>2</size>
      </results>
 </go>

<go>
      ...{existing items}...
      <results>
            <!-- This is key format  -->
            <xxx1>yyyy1</xxx1>
            <xxx2>yyyy2</xxx2>
            <start>1</start>
            <end>2</end>
            <size>2</size>
      </results>
 </go>
<go>
      ...{existing items}...
      <results>
            <!-- This is key index format  -->
            <key1>xxx1</key1><data1>yyyy1</data1>
            <key2>xxx2</key2><data2>yyyy2</data2>
            <start>1</start>
            <end>5</end>
            <size>5</size>
      </results>
 </go>
```

Figure 10.3: Example message containing replay output

Over-rides Section

Overrides allow a single replay node to be used to replay the data many different ways. Override element values are specified as field paths and names under the path specified in the Basic section. Use the Defaults section to determine what values to be set in the over-ride message (e.g. “key” or “k” for key format).

If the override field does not exist, or is blank or invalid, the default value from the Defaults section will be used.

Figure 10.4 shows the Over-rides configuration properties of the CacheReply node with some sample element values. Figure 10.5 shows an example of what the override message could be based on the element names specified in Figure 10.4, and the “Path” on the Basic tab being set to “Root.XML.go”.

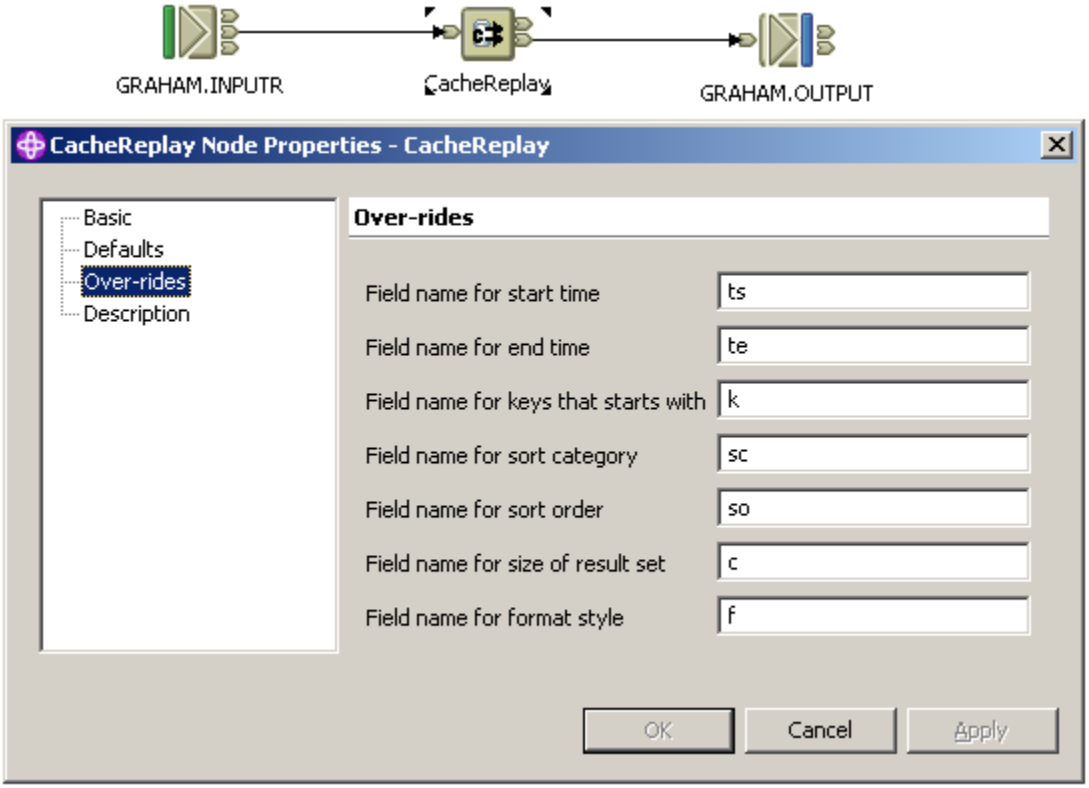

Figure 10.4: Sample screen for setting the over-ride properties of a cache replay node

```
<go>
      <ts>01/01/03 00:00:00.000</st>
      <k>S</k>
      <sc>key</sc>
      <so>alphabetical</so>
      <c>5</c>
      <f>ki</f>
 </go>
```

Figure 10.5: Example message containing replay information

The message in Figure 10.5 will replay messages that were inserted after Jan 1st 2003, with a key that starts with "S", in key order alphabetically, 5 at a time, in the output format of key with index. If there are more than 5 cache records, then a second message will be created and propagated down the message flow. If there are more than 10 cache records then a third message will be created and propagated, and so on. Note that missing override values mean that the values specified in the Defaults section apply (e.g. element te does not appear in the override message, so the default end time applies).

11. Using the CacheConfig Node

Much of the time, you can simply add the CacheGet and CachePut nodes to your flows and begin using the cache. However, you will need the CacheConfig node (or its Named equivalent) if:

- You desire to pre-load the cache from a database
  New! Database info can now be specified via a JDBC configurable service
- **New!** You desire to pre-load the cache from a properties file
- You need to change the host/port defaults used by real-time broker flow
- You are asked to trace the Cache execution for debugging

Basic Tab: Pre-loading a cache from a source database or file

This section of the CacheConfig node controls whether or not the cache is initialized from a database or file when the broker is started, and if so, whether the data is refreshed automatically at some interval. Figure 11.1 shows an example JDBC configuration.

If initialization of the cache is required, then the following values will be used regardless of source:

- **Cache Initialized**: yes/no, use the below parameters to init cache?
- **Period**: How often to re-read the source to refresh the cache?
- **Scope**: Scope of the cache (same as CacheGet/CachePut nodes)

To read values from the source only one time, set the Period value to **-1**. This will turn off the refresh operation from this particular CacheConfig node only. The read operation will happen only one time at startup or during the first message to pass through the node.

Note that there is no “Life of data field” as there is on a CachePut node. Data read from the source is put into the cache with UNLIMITED life; it can only be removed by a destructive Get. Updating the value of an item in the source will cause the cache to be updated on the **next** refresh, but deleting an item from the database will not cause the corresponding item to be removed from the cache. Updates to the cache value are never propagated back to the database.

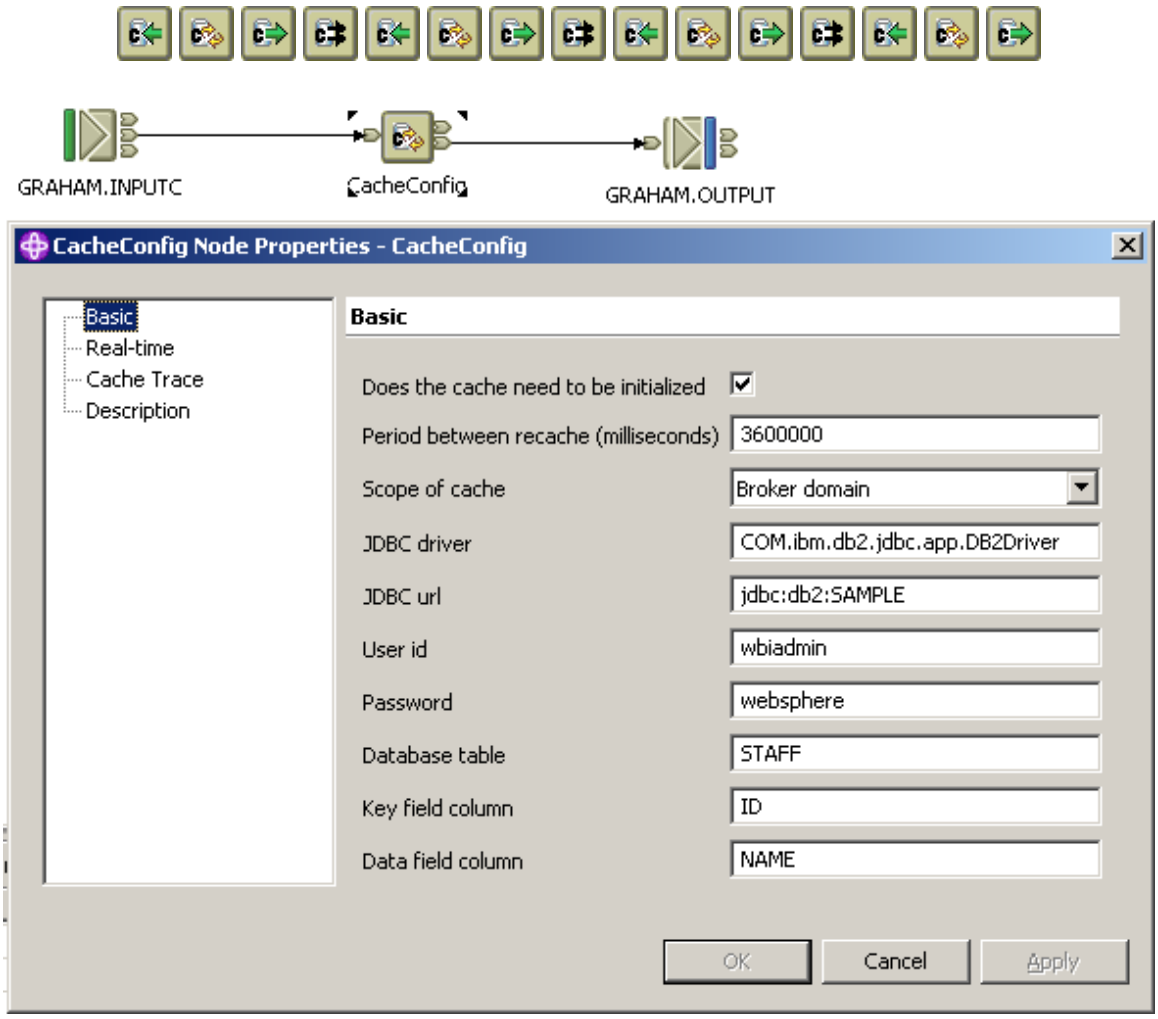

Figure 11.1: Example configuration of Basic section of CacheConfig node

Note that multiple CacheConfig nodes may be used to read data from several databases or files. If this is needed, you must ensure all of the remaining values on the other tabs are the SAME for ALL CacheConfig or NamedCacheConfig nodes within an execution group.

Source configurations

There are 4 possible source configurations.

Direct JDBC
JDBC Provider Service
Java Properties File
JVM system property

See below for each configuration type.

Direct JDBC

This allows for direct configuration of the JDBC information in the node. The following fields are used:

- **JDBC Url**: JDBC url, starting with "jdbc:"
- **JDBC Driver**: Which JDBC database driver to use
- **Userid**: Database userid
- **Password**: Database password in clear text
- **Table**: Table to SELECT from
- **Key**: Which is the key field to use in the table?
- **Data:** Which field should be used for the data?

The query executed is basically: SELECT {Key}, {Data} FROM {Table}. This allows for any number of key/data pairs to be read from the database into the cache (up to broker memory constraints).

JDBC configurable service

Beginning in WMB V7, it is possible to configure a JDBC Configurable Service to specify the database connection information, userid and password. This version of IA91 now supports this method. The following fields are used:

- **JDBC Url**: Must be formatted as: service:*jdbcservicename*
- **Table**: Table to SELECT from
- **Key**: Which is the key field to use in the table?
- **Data:** Which field should be used for the data?

The JDBC configurable service named (jdbcservicename above) will be used to obtain the connection to the database. This is a much more secure manner to obtain the connection because this connection data is now set externally from the message flow, and therefore no longer needs to be changed here as a flow moves from development, to test and to prod.

NOTE: Due to broker limits, the service database can only be referenced when a message is passing through the CacheConfig node. See "Ensuring pre-loading has occurred prior to any Get" below for more details.

Java properties file

This allows for configuration of the file information in the node. The following fields are used:

- **JDBC Url**: File URL, starting with "file:"

The URL must be a valid Java file URL (i.e. "file:///path/to/file"). The file referenced must be a properties file readable by the Java Properties class.

The file can be a plain text file of properties, or XML files can be used – if XML is used, the file name **must end** in ".xml" (case insensitive).

Valid file URLs have the format file://host/path/file If the file is on the local file system, host can be omitted, but the slashes must be present (e.g. file:///path/file). Path should be an absolute path. For Windows, replace backslashes with forward slashes (e.g. C:\PATH\FILE becomes file:///C:/PATH/FILE).

JVM system property

This allows for configuration of the URL externally:

- **JDBC Url**: Valid system property name, starting with "$"

The value after the $ will be interpreted as a JVM System Property name. The value of the system property, if non-blank, must be one of the above valid URL types (e.g. jdbc:, file: or service:). If the system property is not found, the URL is considered invalid and the pre-load & refresh are disabled.

To configure a JVM variable, use the mqsichangeproperties command with the ComIbmJVMManager object:

```
mqsireportproperties broker -e executiongroup -o ComIbmJVMManager
-n jvmSystemProperty

mqsichangeproperties broker -e executiongroup -o ComIbmJVMManager
-n jvmSystemProperty -v "-DMyTest=0 -DMyTest2=Test2"
```

Unrecognized URL handling

If the effective JDBC URL does not begin with "jdbc", "file:", or "service:", the CacheConfig node will simply not attempt to load any data and will set the recache interval so that no further attempts will be tried. This also applies to using the "service:" URL in a version of the broker which does not support configurable services.

A valid URL which fails, however, will continue to try. For example, if the properties file does not exist, it's possible that something or someone will simply create it after the broker is running, or if the database isn't available, it may come back.

Ensuring pre-loading has occurred prior to any Get

If you are initializing the cache from a database or file, the CacheConfig node must be connected into your flow so that the message flows through it to ensure that the cache is loaded before you attempt any Cache Get (via node or ESQL). A CacheConfig node will not alter the message as it comes through, but it will use the message's arrival to check to see if the pre-load has occurred, and if it has not, to attempt to perform the pre-load prior to sending the message to the out terminal. For databases accessed via the configurable service option, this is the ONLY way the database is accessed.

This is required because the current node initialization process does not signal that all properties have been set, so there is simply a 5 second delay after the node is created to allow all properties to be set before attempting to read from the database or file. This delay can allow messages to be processed through a flow before the pre-load occurs.

Pre-load failure retries

If the pre-load operation fails (e.g. database unavailable), the initialization routine will automatically retry every 5 seconds until the pre-load succeeds.

This retry mechanism will also be used if the scope is set to Broker Domain and the pub/sub mechanism fails (e.g. if the real-time flow is not yet running at broker startup). In this case, the local cache will be updated once to allow local applications to continue. The database or file will not be re-read during subsequent retries until the pub/sub mechanism is available (in order to cut down on database traffic or file I/O.)

If a message passes through the CacheConfig node during this retry process, the message will still be processed; this may trigger an additional data load retry attempt.

New! Within the default cache, the last successful recache time will now be stored for each CacheConfig node. The name of the item is `SYSTEM.GTO.RECACHE.{nodename}`, where spaces in the node name are replaced by underscores ("_"). The value is the Java time in milliseconds. This is also published if the scope is "Broker Domain".

Eliminating redundant loads

As of April 2010, work has been done to eliminate redundant cache load operations.

As noted in bold above, if you are pre-loading a cache, you should wire in CacheConfig nodes to ensure the pre-load is done prior to processing. If you have many flows using the same cache, this can result in many CacheConfig nodes with exactly the same database or file information. This is also common when using a common sub-flow to access a cache.

Prior to this change, this setup would cause the database or file to be read one time per CacheConfig node. At broker startup, for example, this meant that a database could be hit multiple times for the exact same data.

The CacheConfig nodes will now determine if the **same** database or file is being used multiple times for the **same** cache in the **same** execution group. When this situation is identified, the redundant reads are suppressed.

Most of the information from the Basic tab is used to try to uniquely identify this scenario. The comparisons for the fields are case-sensitive, except for table name, key field name, and data field name. Note: This does include the fields ignored when files are being used instead of databases.

Two fields are not used for this purpose: Scope and Period. However, these can affect the load operations if different values are used on otherwise duplicate CacheConfig nodes.

The net effect of having different Scope values in CacheConfig nodes for the same unique database & cache will be that Broker Domain will in effect override the Execution Group setting. This may still result in multiple reads at startup.

The net effect of having different Period values in CacheConfig nodes for the same unique database & cache will be that only the shortest recache interval will be utilized.

Example #1: Two CacheConfig nodes are configured exactly the same to read database DB except that one has URL “jdbc:db2:OTHER” while the other has URL “jdbc:db2:other”. They will be treated as unique databases resulting in multiple reads.

Example #2: Two CacheConfig nodes are configured exactly the same to read database DB except that one is set to recache every 15 minutes and the other is set to recache every 60 minutes. After startup, per the first node, the database is recached every 15 minutes as configured. At the 60 minute marks, the second node will attempt to run and will now determine that the database was read 15 minutes ago and will not recache.

Example #3.1: Two CacheConfig nodes are configured exactly the same to read file PROP except that one is set to Execution Group while the other is set to Broker Domain. At startup, the first node reads the file and does not publish the values. The second node then reads the file again and does publish the values (because they were not already published).

Example #3.2: The same two CacheConfig nodes as above. This time, at startup, the nodes are initialized in a different order so that the second node reads the file first and does publish the values. The first node now determines that a 2nd reading of the file is not needed (since the values were already published).

Example #4: Two CacheConfig nodes are configured exactly the same to read database DB except that one is set to recache every 15 minutes with Execution Group scope and the other is set to never recache (Period = -1) with Broker Domain scope. At startup, depending on order, the database is read either 1 or 2 times and the values are published. After startup, per the first node, the database is recached every 15 minutes as configured, except because of the 2nd node, the recached data is always published as well.

Real-time Tab – V6 and V6.1: Changing the host/port defaults

For specifying only tracing or real-time flow defaults, a CacheConfig node simply needs to appear in your flow. **** This section is obsolete in IIB. ****

This section of the CacheConfig node controls the configuration needed to use the Real-TimeOptimizedFlow discussed in section 5. Figure 11.2 shows an example configuration.

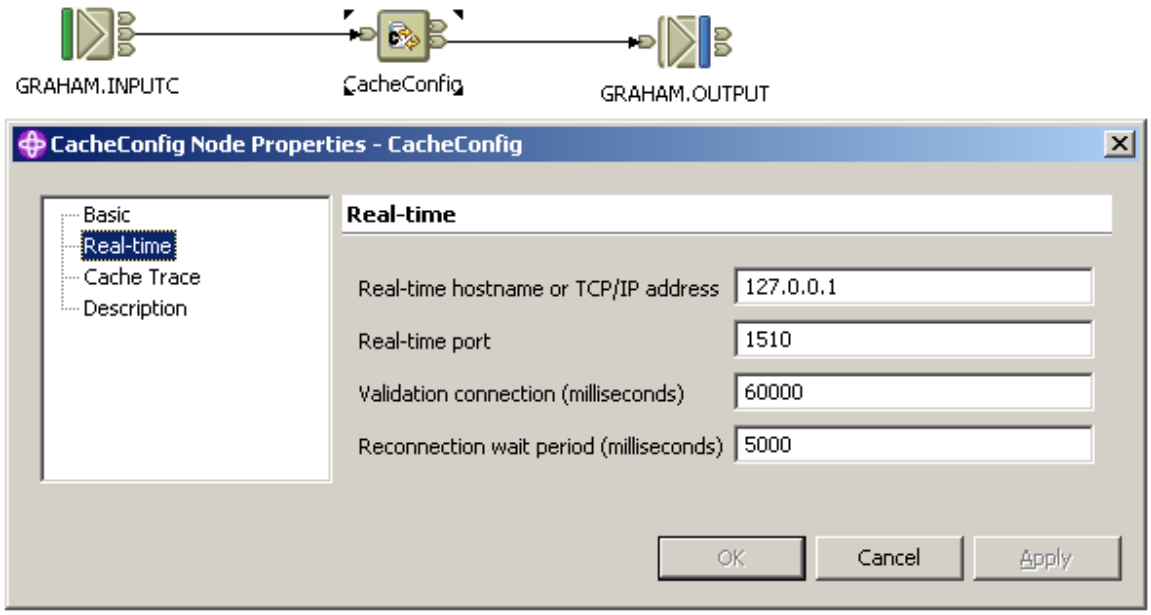

Figure 11.2: Example of the Real-time section of the CacheConfig node

The following configuration information is used to connect to the publisher broker:

- **Host**: Hostname or host address, defaults to 127.0.0.1 (localhost)
- **Port**: Port to use, typically **1506** however in this example showing 1510
- **Validate** How often should the connection best test to ensure it's ok
- **Reconnect** If the connection has broken how often to try to re-establish the connection?

Note: For this to work properly, ensure that EVERY CacheConfig and NamedCacheConfig node in an execution group uses the same values. There is only one set of these values per execution group. Using multiple nodes with differing values will result in essentially random behavior.

Note: For brokers using WMQ V7 pub/sub, the cache simply connects to the broker's queue manager. In this case, the hostname & port are ignored.

Cache Trace Tab – Tracing the Cache execution for debugging

For specifying only tracing or real-time flow defaults, a CacheConfig node simply needs to appear in your flow.

This section of the CacheConfig node controls the configuration needed to use to obtain a trace of the Cache nodes. Figure 11.3 shows an example configuration.

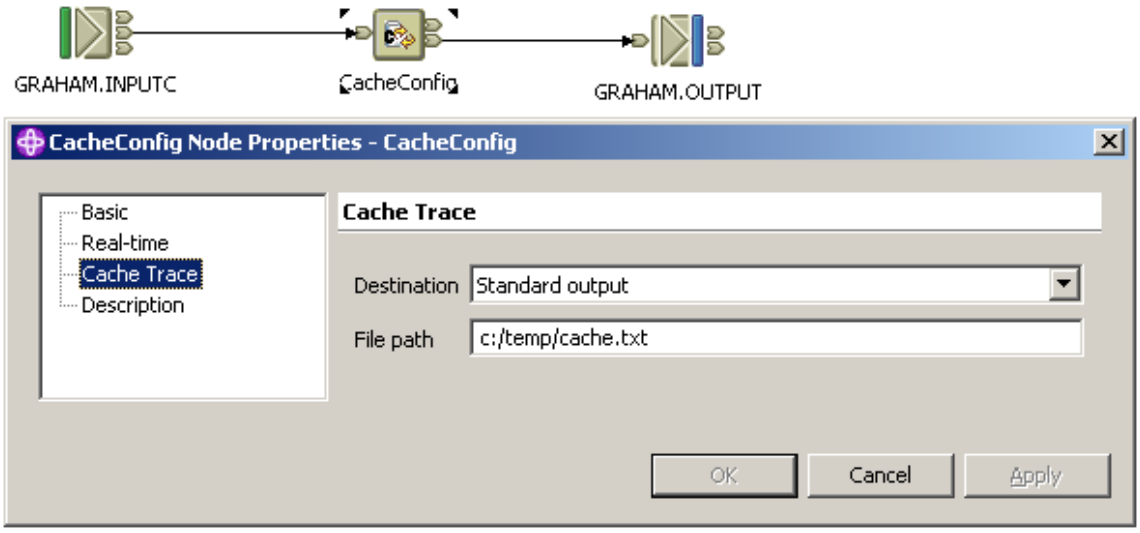

Figure 11.3: Example of the Cache Trace section of the CacheConfig node

Configuration information used for tracing:

- **Destination**: Where should the trace output go?
- **File path**: If destination is File, then which file.
  (Note: I have only stress tested File and None with v5 and v6.)

If the destination is File and the file path starts with $, the remaining value will be interpreted as a JVM System Property name. The value of the system property, if non-blank, must then be a valid file URL. If the system property is not found, file name is not changed.

To configure a JVM variable, use the mqsichangeproperties command with the ComIbmJVMManager object:

```
mqsireportproperties broker -e executiongroup -o ComIbmJVMManager
-n jvmSystemProperty

mqsichangeproperties broker -e executiongroup -o ComIbmJVMManager
-n jvmSystemProperty -v "-DMyTest=0 -DMyTest2=Test2"
```

Note: For tracing to work properly, ensure that EVERY CacheConfig and NamedCacheConfig node in an execution group uses the same values. There is only one set of these values per execution group. Using multiple nodes with differing values will result in essentially random behavior.

Caveat: Tracing can be "chatty" and can quickly fill up a file system, so use it with care.

12. Using the Cache from ESQL

A new feature of Message Broker, starting with Version 5 CSD04, is the ability to call a Java function from ESQL. Six functions now allow ESQL to access the Cache in addition to using the Get and Put nodes.

Using the new functions is simple. Included in the ia91.wbimb.zip file is a file named CacheJavaFunctions.esql. The most flexible way is to create a new Message Flow project and import this file into it. Add a reference this new project from your message flow project (right-click and select Properties – Project References). You may also import this file directly into any project.

Caveat: The Broker Domain caching mechanism should be non-transactional. However, testing shows that Broker Domain cached values may not be available until the in-flight transaction, if any, is committed.

putIntoCache and putIntoNamedCache

These functions allow values to be stored into the Cache, and are the equivalent to the CachePut and NamedCachePut nodes respectively. Figure 12.1 shows sample calls.

```
DECLARE Status CHAR;
SET Status = putIntoCache(OutputLocalEnvironment.Variables.Key1,
    OutputLocalEnvironment.Variables.Value1, 60000, FALSE);

SET Status = putIntoNamedCache('alternate',
    OutputLocalEnvironment.Variables.Key1,
    OutputLocalEnvironment.Variables.Value1, 60000, FALSE);
```

Figure 12.1: Sample code calling putIntoCache and putIntoNamedCache

The parameters to putIntoCache and putIntoNamedCache are:

- CHAR Cache name (putIntoNamedCache only)
- ESQL Reference to the key to store the value under
- ESQL Reference to the value to be stored
- ESQL Integer representing the Life of Data value from the CachePut node, in milliseconds; use -1 to never expire a value
- ESQL Boolean representing the Scope of the cache from the CachePut node, where FALSE means Execution Group, and TRUE means Broker Domain

Key and value must both be non-null ESQL References, meaning you cannot pass scalar variables (e.g. `DECLARE Key CHAR;`) and the elements must exist. However, any InputRoot, OutputRoot, Environment, InputLocalEnvironment or OutputLocalEnvironment reference, or a reference variable to someplace under one of these will work.

The Status value returned will be one of the following strings:

- Ok
- Exception {java exception message}
- Null Name not allowed
- Null Key not allowed

getFromCache and getFromNamedCache

These functions allow values to be retrieved from the Cache into an existing element in a message tree, and are the equivalent to the CacheGet and NamedCacheGet nodes respectively. Figure 12.2 shows sample calls.

```
DECLARE Status CHAR;
SET Status = getFromCache(Environment.Variables.Key,
    Environment.Variables.Value, FALSE, FALSE);

SET Status = getFromCache('alternate', Environment.Variables.Key,
    Environment.Variables.Value, FALSE, FALSE);
```

Figure 12.2: Sample code calling getFromCache and getFromNamedCache

The parameters to getFromCache and getFromNamedCache are:

- CHAR Cache name (getFromNamedCache only)
- ESQL Reference to the key to retrieve the value from
- ESQL Reference to where the value is to be stored
- ESQL Boolean representing whether the get is destructive; FALSE is not destructive, and TRUE means delete the key and value from the cache after the get operation
- ESQL Boolean representing the Scope of the cache from the CachePut node, where FALSE means Execution Group, and TRUE means Broker Domain. This is ignored if the destructive value above is FALSE, since non-destructive gets do not need to be published.

Key and value must both be non-null ESQL References, meaning you cannot pass scalar variables (e.g. `DECLARE Key CHAR;`) and the elements must exist. However, any InputRoot, OutputRoot, Environment, InputLocalEnvironment or OutputLocalEnvironment reference, or a reference variable to someplace under one of these will work. Due to the way ESQL References work with Java routines, the value reference must already exist; it will not be created. To create the value in the tree if it does not exist, use createFromCache or createFromNamedCache instead.

The Status value returned will be one of the following strings:

- Ok
- Exception {java exception message}
- Error Output value is in a Read Only Message (e.g. InputRoot)
- Null Name not allowed
- Null Key not allowed
- Null Value not allowed

createFromCache and createFromNamedCache

These functions allow values to be retrieved from the Cache into a new or existing element in a message tree, and are the equivalent to the CacheGet and NamedCacheGet nodes respectively. Figure 12.3 shows sample calls.

```
DECLARE Status CHAR;
SET Status =
   createFromCache(OutputLocalEnvironment.Variables.Key,
   OutputLocalEnvironment, 'Variables.LocalValue', FALSE, FALSE);

SET Status = createFromNamedCache('alternate',
   OutputLocalEnvironment.Variables.Key, OutputLocalEnvironment,
   'Variables.LocalValue', FALSE, FALSE);
```

Figure 12.3: Sample code calling createFromCache and createFromNamedCache

The parameters to createFromCache and createFromNamedCache are:

- CHAR Cache name (getFromNamedCache only)
- ESQL Reference to the key to retrieve the value from
- ESQL Reference to the base of the tree where the value is to be stored
- ESQL String representing the path to where the value is to be stored under the base of the tree
- ESQL Boolean representing whether the get is destructive; FALSE is not destructive, and TRUE means delete the key and value from the cache after the get operation
- ESQL Boolean representing the Scope of the cache from the CachePut node, where FALSE means Execution Group, and TRUE means Broker Domain. This is ignored if the destructive value above is FALSE, since non-destructive gets do not need to be published.

Key and base must both be non-null ESQL References, meaning you cannot pass scalar variables (e.g. `DECLARE Key CHAR;`) and the elements must exist. However, any InputRoot, OutputRoot, Environment, InputLocalEnvironment or OutputLocalEnvironment reference, or a reference variable to someplace under one of these will work. The path to the new value, represented by the string, will be created if it does not exist. The base and path can be specified in several ways to achieve the same update. For example, both statements in Figure 12.4 are equivalent.

```
SET Status =
   createFromCache(OutputLocalEnvironment.Variables.Key,
   OutputLocalEnvironment, 'Variables.LocalValue', FALSE, FALSE);

SET Status =
   createFromCache(OutputLocalEnvironment.Variables.Key,
   OutputLocalEnvironment.Variables, 'LocalValue', FALSE, FALSE);
```

Figure 12.4: Equivalent createFromCache calls

The Status value returned will be one of the following strings:

- Ok
- Exception {java exception message}
- Error Output value is in a Read Only Message (e.g. InputRoot)
- Null Name not allowed
- Null Key not allowed
- Null Root not allowed
- Empty value path not allowed

Cache Cleanup from ESQL

Effective May 2006, cache cleanup will occur automatically during ESQL calls. The defaults are to cleanup the cache accessed every 10 000 ESQL calls or every 5 minutes, whichever occurs first. This is intentionally different than the node cleanup frequency in order to not adversely affect ESQL performance. Four new ESQL functions allow these parameters to be adjusted as desired. See below for details.

The different frequency values may mean that the cache is cleaned more or less often than desired, depending upon whether ESQL or nodes are used.

The functions to determine or adjust the cleanup parameters are shown in Figure 12.5.

```
DECLARE oldFreq     INTEGER;
DECLARE oldTimeInMs INTEGER;

SET oldFreq = getCleanupFrequency();
SET oldTimeInMs = getCleanupTimeInMs();

CALL setCleanupFrequency(100000);
CALL setCleanupTimeInMs(100000);
```

Figure 12.5: Sample code changing cleanup parameters

The parameter to setCleanupFrequency is:

- INTEGER Number of ESQL operations between cleanup checks

The parameter to setCleanupTimeInMs is:

- INTEGER Number of milliseconds between cleanup checks

For both set calls, values below 0 are treated as -1, meaning that the particular parameter will not be used to determine cleanup frequency.

Caching Null values from ESQL (V7 and later only)

In WMB V6, it is not possible to store or access Null values in caches via ESQL; this is due to the way WMB V6 handles nulls and not really a cache limitation.

In WMB V7 and later, it is now possible to store and access Null values via ESQL. In order not to break existing code, however, the behavior for handling nulls needs to stay the same. Therefore, new functions have been created allowing Null values to be handled if needed in WMB V7 and later.

In the supplied CacheNodeFunctions.esql definitions, there are new xxxxV7 versions of each of the cache access functions; for example, getFromCache and getFromCacheV7. The new functions have a new parameter, named AllowNulls, allowing the handling of Null values to be controlled via parameters.

If the new parameter value is FALSE, Nulls will be handled as is done today in WMB V6 – that is, null values will throw exceptions and return Status other than "Ok" instead of being returned or stored.

If the new parameter value is TRUE, Nulls can be stored and accessed and the Status value returned will be "Ok".

```
DECLARE Status CHAR;
SET Status = getFromCacheV7(Environment.Variables.Key,
    Environment.Variables.Value, FALSE, FALSE, TRUE);
```

Figure 12.6: Sample code calling getFromCacheV7

13. Cache Initialization Program

The same cache initialization process used by the CacheConfig nodes can also be used externally from the broker to set up the in-memory cache for a broker domain cache. It requires the use of the Broker Domain cache mechanism. This can be run from a command prompt or simple command script. This script should:

- Set up the classpath to include CacheShared.jar, the WMQ java libraries, and the database java libraries required
- Specify the following broker info to publish the data:
  - hostname and port for the WMB V6.x RealTime flow
  - hostname and port and queue manager for WMQ V7, if WMQ pub/sub is not being used (queue manager name needed to verify WMQ pub/sub is not used)
  - queue manager only for WMQ V7 if WMQ pub/sub is used
- Specify the following for database initialization:
  - JDBC driver for the database to read from,
  - JDBC URL for the database,
  - table, key column, and data (info) column
  - user id, password (if needed)
- Specify the following for properties file initialization
  - Complete path and file name (must end in .xml for XML properties files)
    - Note: Can also be passed in JDBC URL if a file: URL
- Specify what named cache (if any) to store the data into
- Call the InitCacheSample program using the syntax from Figure 13.1

```
java com.goakes.Cache.InitCacheSample options

      -v             : Verbose
      -h | --host    : publisher host (V6.x) (required for MQ V6
                       pub sub)
      -p | --port    : publisher port (V6.x) (defaults to 1506)
      -m | --qmgr    : queue manager (required for MQ V7)
      -f | --file    : File name
      -j | --jdbc    : JDBC URL (required)
      -d | --driver  : JDBC Driver (required)
      -u | --user    : JDBC Userid (password required if
                       specified)
      -w | --pass    : JDBC Password (user required if specified)
      -t | --table   : Database Table (required)
      -k | --key     : Database key column (required)
      -i | --info    : Database data (info) column (required)
      -n | --name    : Cache name (defaults to "default")
```

Figure 13.1: Syntax for calling the InitCacheSample program

The example in Figure 13.2 shows reading data from the STAFF table using ID as the key and NAME as the data. STAFF is in a DB2 database named SAMPLE. The data from STAFF is published using the RealTime flow running at 127.0.0.1, port 1506.

```
java com.goakes.Cache.InitCacheSample -h 127.0.0.1 -p 1506 -d
"COM.ibm.db2.jdbc.app.DB2Driver" -j "jdbc:db2:SAMPLE" -u wbiadmin
-w websphere -t STAFF -k ID -i NAME
```

Figure 13.2: Example for calling the InitCacheSample program

14. Broker memory considerations

Each named Cache uses memory within the JVM of the broker's execution group. Very large caches can cause the execution group to run out of heap space. This can occur many ways, including, but not limited to:

- Using very long or infinite "Life of Data" values during CachePut operations
- Using very large "Timed frequency of clean-up" values during CacheGet or CachePut operations
- Using very large "Clean up after number of invocations" values during CacheGet or CachePut operations
- Storing large amounts of data, then not using CacheGet or CachePut Nodes to clean up the cache

If you do see java.lang.OutOfMemory errors, one option is to increase the heap size of the JVM. The syntax of the commands to do this is shown in Figure 14.1.

```
mqsireportproperties broker -e executiongroup -o ComIbmJVMManager
-n jvmMaxHeapSize

mqsichangeproperties broker -e executiongroup -o ComIbmJVMManager
-n jvmMaxHeapSize -v newsize
```

Figure 14.1: Syntax of the command to display and change the JVM size for an execution group

Other memory considerations

Another related memory issue may occur in very large complex flows with many Cache nodes, other Java-based nodes, or XML Transformation nodes. In these cases, the execution group may terminate due to a `java.lang.StackOverflowError`.

In these cases, the Java operating system stack needs to be larger. This can happen on V6.1 or later brokers due to a change in the JVM used by broker.

The IBM recommended solution is to increase the operating system stack from the default 256KB to 2MB. This is done by setting an environment variable and restarting the broker to pick it up; an example is shown in Figure 12.2.

```
export IBM_JAVA_OPTIONS=-Xmso2m
```

Figure 14.2: Setting the operating system stack to 2MB under AIX

15. Performance Comparison of Options

There are other ways to store data needed by message flows. For example, a database or the MQGet Node could be used. The following table will help to determine which of these several state mechanisms would be the best for you.

	Database	MQGet	Custom Java Code	Shared Variable	Cache nodes
M/F Cross Plex	Yes	Yes	Maybe	No	Broker Domain
No Administration	No	No	Maybe	Yes	Yes
IBM Supported	Yes	Yes	No	Yes	No
Portable to M/F	Yes	Required	Maybe	Yes	Yes
Performance			Fastest	Fastest	Fastest
Persistent	Yes	Optional	Maybe	No	No
Search flexibility	High	Medium	Depends	High	Low
Use in compute node	Yes	No	Maybe	Yes	Yes

Table 15.1: Comparison of different ways of storing state in message flows

16. Known Limitations / Testing

- The nodes have only been tested with string data in normal elements within the XML and XMLNS message trees.
  - ESQL Functions that return BLOB must be CAST to CHAR for Broker Domain caching
- Cleaning up of memory can only occur when the cache node is entered.
- There is an operating system limit to the size of memory a process can use, so there is a limit on the amount of data that can be stored in memory.
- In my experience, some versions/platforms may not automatically pick up the shared-classes JAR files. These will need to be added to the CLASSPATH environment variable in order for the ESQL functions to see the same cache as the Cache nodes.

17. Comparison of Performance of Cache Node vs. Cache ESQL vs. DB2 vs. Shared Variables

To see the relative difference in performance, a test flow was constructed to carry out a series of puts followed by a series of gets. Various technologies were used: IA91 cache via node, IA91 cache via ESQL, a DB2 database, and WMB shared variables. The message flow shown in Figure 17.1 was used to obtain the results and then Table 17.1 shows the amount of time taken for the different tests.

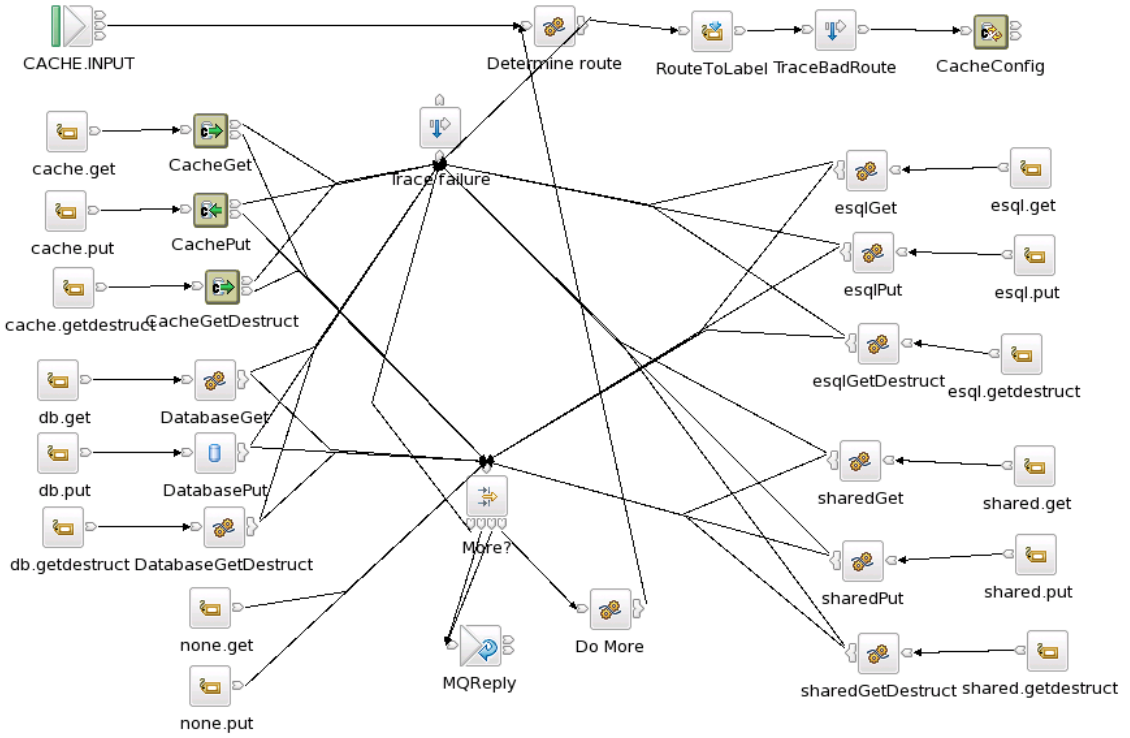

Figure 17.1: Message flow used to do performance test

Test Type	Size	Average time (seconds)		
		Put time	Get time	Get Destructive
None	100	0.00024	0.00018	0.00024
	1000	0.00024	0.00017	0.00022
	3000	0.00068	0.00016	0.00022
Cache Node	100	0.00112	0.00096	0.00152
	1000	0.00245	0.00183	0.00178
	3000	0.00139	0.00135	0.00139
Cache ESQL	100	0.00107	0.00099	0.00137
	1000	0.00129	0.00156	0.00101
	3000	0.00100	0.00114	0.00089
Shared Variable	100	0.00281	0.00396	0.00210
	1000	0.00254	0.00351	0.00215
	3000	0.00248	0.00356	0.00215
Database	100	0.00112	0.00075	0.00122
	1000	0.00120	0.00088	0.00134
	3000	0.00131	0.00079	0.00111

Table 17.1: Results comparing sending 5000 messages using WMB V6.

Test Methodology
The testing procedure was as follows:

1. The following "types" were run:
    a. None – To test flow overhead
    b. Database – A DB2 table with two columns (key/value) with a unique index on the key
    c. Cache Node – Default IA91 cache, accessed only via the CacheGet and CachePut nodes. Only execution group caching is used.
    d. Cache ESQL – Default IA91 cache, accessed only via the ESQL createFromCache and putIntoCache functions. Only execution group caching is used.
    e. Shared Variable – A shared ROW type variable, containing a tree of values. The keys are not sorted.
2. Three different test runs are run, where different length data values are stored:
    a. 100 byte values
    b. 1000 byte values
    c. 3000 byte values

3. A single test run for a given size and type is defined as follows:
    a. Clearing of any existing data
    b. 100 puts (priming, not timed)
    c. 5000 puts
    d. 100 gets (priming, not timed)
    e. 5000 gets
    f. 100 get destructs (priming, not timed)
    g. 5000 get destructs
4. Key values are a constant length and range from KEY00000 to KEY05000. However, to ensure that sequential access doesn't help, the order of the keys used for each operation is randomized
5. Tracing was disabled
6. The priming steps ensure that the code needed is in memory instead of paged out by Linux
7. Database: Put is a ESQL Insert; if that fails because the key already exists, then an ESQL Update is executed on the key to replace the value stored (not tested). Get is an ESQL Select. The Get Destruct is an ESQL Select followed by an ESQL Delete. (Passthru function is not used).
8. Shared Variable: Put is a loop over the row values; if not found, a new row is added at the end (so that the list is never ordered); if found, the value is replaced and the loop is terminated. Get is an ESQL Select. Get Destruct is a loop over the row values; when found, the entry is removed via a DETACH statement.
9. The time is computed after the get/put operation by subtracting the MQPUT message PutTime field from WMB's CURRENT_TIMESTAMP in a node after the operation; this should then exclude any test driver overhead. Times are then averaged.
10. The tests were run on a ThinkPad T400 with3GB RAM running Windows XP SP3. WMB V6 CSD 10 was installed on a VMWare VM running CentOS 5.5 with 652MB RAM.

Additional notes on performance testing

To be fair, I did not spend a lot of time trying to improve the Shared Variable performance; I was simply minimizing my coding effort. New values are simply stored at the end of the tree and searching was done via an ESQL Select statement or a FOR loop. This means that, on average, ½ of the values must be checked to find a match.

If I were really trying to use a Shared Variable in this manner, I would have spent some time determining how to improve performance. For example, organizing the keys better could have meant using a binary search vs. For Loops and Select statements.

18. Interpretation of Cache Trace files

This section describes some of the messages that appear regularly in the trace file (if trace is enabled), along with interpretation of the message. This should help in determining where a problem may exist with relation to caching.

Here is a typical line from the trace file:

```
2007-04-12 08:40:51.344 T1    NamedCachePu NamedCachePut   ] Constructor
^          ^            ^     ^            ^               ^ ^
|          |            |     |            |               | |
Date       Time         |     Class        Method          | Comment
                        Thread                              Level
```

Date and time are self-explanatory. Thread identifies which thread is doing the tracing. The possible thread values are: "PublT" for the thread receiving real-time updates; "ConnT" for the thread keeping the PublT thread connected to the real-time flow; "C9999" for CacheConfig nodes; "G9999" for CacheGet nodes; "P9999" for CachePut nodes; "R9999" for CacheReplay nodes; and "D9999" for threads used to refresh data from a database from a CacheConfig node. The "9999" will be replaced by a number indicating unique threads, however, since the number of threads actually created by broker can be large, this number simply rolls back to 1 if it exceeds 9999.

The class, method, and comment fields can be used to go into the source to locate the related code. Typically, the comment gives a useful piece of information for debugging. Finally, the level field gives a visual indicator of when a piece of code starts or ends; "{" marks the start of a method call, "|" marks a trace from the middle of a method call, "}" marks the end of the method call, and "]" indicates that only a single trace call was needed from the method (rather than 3). Level and comment can also be indented to show nested calls from within the cache code.

Typical trace points that appear are: creation of a node within the flow and all initial settings from the flow (e.g. from broker startup or a deploy); deletion of a node (e.g. from broker shutdown or a deploy); execution of the node body (Cache Get/Put as a message passes through the node); heartbeat and other messages from the pub/sub caching mechanisms; and period tasks such as refreshing the cache from the database and staying connected to the real-time flow.

Exceptions can and do appear in the trace file. Some typical ones are:

`com.ibm.mq.jms.JMSWrappedException: MQJMS6115` – This means that the real-time flow is not running and some node is setup for Broker Domain caching. This typically appears at start-up a few times until the flow starts.

`javax.jms.IllegalStateException: MQJMS6116` – This means that the real-time flow was running but has since stopped. This typically appears at shut-down.

`java.lang.ClassNotFoundException` – This typically means that either an MQ jar file or database jar file is not available in the CLASSPATH, or that the cache nodes are not properly installed.

Various database exceptions – These are too varied to fully document here and are dependent upon the database being used. These are only from the CacheConfig nodes where database initialization is being done. Typical types experienced are "not authorized" (userid/password invalid), "connection refused" (check jdbc url host & port), and "database or table not found" (check ODBC entries and/or jdbc url – not the odbc.ini used by broker however).

Various file IO or property exceptions – These are too varied to fully document here and are dependent upon the file being used. These are only from the CacheConfig nodes where property file initialization is being done. Typical types experienced are "not authorized" (file permissions invalid), and "file not found" (check file url).