



## xTier™ Cluster Service

The xTier™ Cluster Service provides the developer with a robust API for building cluster enabled applications.

### Key Benefits:

- Open API for development of fault tolerant and load balanced applications
- Simple to use API for determining the characteristics of a given cluster node
- "Cluster Groups" provide the facility to create virtual clusters within the physical cluster
- Cluster Topology Version allows to optimistically check for topology changes

## Introduction

A "cluster" is a loosely coupled group of computers (called nodes) that cohesively perform certain functionality. Clustering is usually needed in large scale systems to sustain big loads, achieve better fault tolerance, and provide effective load balancing. In some cases, however, clustering may increase response latencies and complicate overall management of the system. That is why the number of nodes for the application cluster should be carefully picked to tailor underlying business requirements. The following paragraphs will examine some of the features common among all application clusters, briefly cover clustering in J2EE, and then proceed to a discussion of the xTier™ Cluster Service.

## Fault-Tolerance, Fail-over, and Availability

Having more than one node in the cluster allows for a system to appear always online even in case of a node failure. The underlying clustering capabilities of the application should transparently remove the failed node from the cluster and allow the application to continue functioning normally. Such *fault-tolerant* behavior eliminates single point failure from cluster-enabled applications.

Often the application will be able to fail-over the task of the failed node to another node in the cluster. Many applications implement extra handling to ensure normal process flow in the event of a node failure. For example, if Node A was caching session data of some user and Node B was the backup node for this data, then in the case of the failure of Node A, the user session handling should be failed over to node B. Node B in its turn should choose another node to become the backup node for user session data. Note that fault-tolerance should not be confused with fail-over capabilities. Fault-tolerance ensures that cluster successfully acknowledges a failure of any node in the system and makes necessary adjustments, which should generally be considered more as middleware functionality. Fail-over logic usually sits on top of the basic cluster functionality – a system (such as a J2EE container or any user defined application) must have special logic to make sure that all business tasks are successfully failed-over to another node in case of a node failure.

Fault-tolerance and fail-over combined constitute the overall availability of a system. Availability measures the degree to which the system is available for use by its clients. If the probability of a single application (node) being available is  $1/m$ , then the probability of it being unavailable is  $(1 - 1/m)$ . For the cluster of  $N$  nodes, the probability of the whole cluster being unavailable is  $(1 - 1/m)^N$ . This number gets smaller as the number of nodes in the cluster increases, which means that more nodes in the cluster provide for better overall availability of a system and that a cluster of more than one node will always be more available than a single server.

## Load Balancing

Having more than one node in the cluster introduces the need for an application to evenly distribute the overall load of the system across every node in the cluster. Load balancers serve to accommodate this purpose. A load balancer is a piece of hardware or software that sits in front of the cluster and forwards incoming requests to different nodes in accordance with some load balancing algorithm. Such an algorithm could be a plain round-robin implementation or it could be based on the CPU utilization of participating nodes. There are many different algorithms and the simplest of which (and is often very effective) would be selecting a node for processing a request at random.

## J2EE Clustering

Clustering is not part of the J2EE specification, however, the vast majority of J2EE application servers have embedded clustering capabilities. Large scale J2EE applications must be able to handle significant loads and be highly available, thus clustering of J2EE applications plays a very important role. The clustering logic of J2EE applications may reside at different levels. For example, a container could choose to load-balance requests to acquire an **InitialContext** within the JNDI driver; or load-balancing and fail-over logic could reside within the remote stubs to ensure that requests are safely forwarded to an active node without disrupting the overall user process flow in case of peak loads or crashes.

Although J2EE containers come with very advanced clustering capabilities, the clustering functionality is usually deeply embedded into the container logic and is often absolutely hidden from the developer. For example, the developer does not have an obvious way to determine which node is designated to process a request or even how many nodes are there in the cluster or further, what the physical characteristic of the nodes are. Since clustering APIs are not a part of the J2EE specification, they are rarely openly exposed by J2EE containers or they are exposed in a very limited fashion. Often users have to implement their own simple clustering functionality if application logic must be based on various physical characteristics of every node or needs to determine which node should be handling a certain task.

## The xTier™ Cluster Service

The xTier™ cluster service provides a clear and well-defined API for determining various characteristics of every node in the cluster, listening to cluster events such as nodes joining and leaving the cluster and node failures, viewing different collections of cluster nodes based on some user-defined criteria, cluster group memberships and more. The xTier™ cluster does not have a notion of an administrator node, nor does it require the startup of any node before other nodes can be started. All nodes in the xTier™ cluster are absolutely identical and can be started or stopped concurrently.

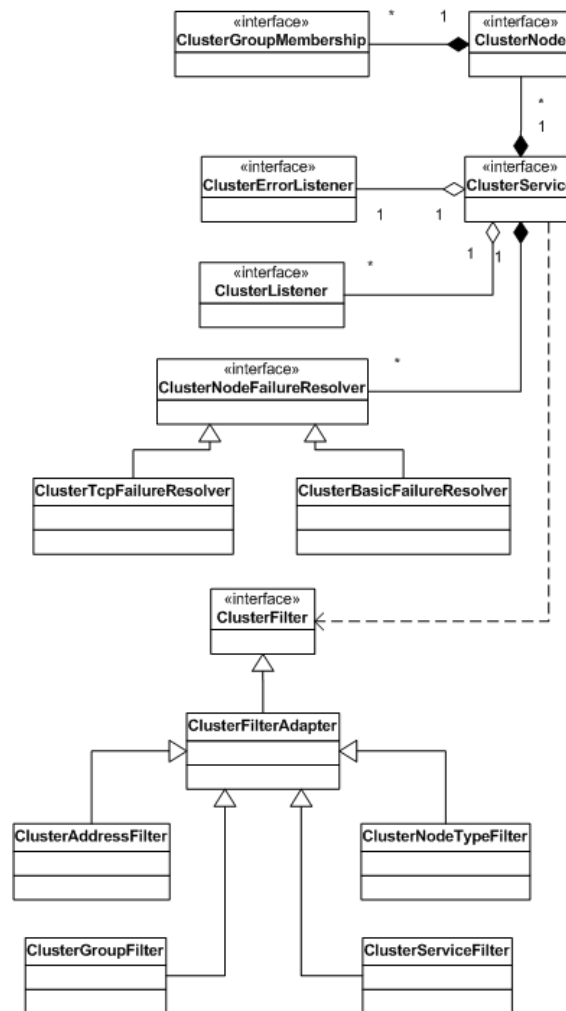
In addition to providing a useful API to the developer, the xTier™ cluster service is used by several other xTier™ services. For example, the Grid service utilizes the cluster API to enable on-demand resource provisioning. The Cache service would normally make use of the Cluster service API to determine which group of nodes (a.k.a. cache topology) should

participate in a cache transaction. These use cases serve as a clear example of how the xTier™ Cluster service could be utilized within a user application.

One of the main advantages of the xTier™ Cluster service is that, although xTier™ makes use of it for other services, it provides an open API that can be used freely as with any other xTier™ service. Unlike J2EE clustering, or the clustering capabilities of many other vendors, the purpose of the xTier™ cluster service is not to accommodate other distributed functionality within the system, but to give the developer a simple and powerful cluster service API to cluster-enable any user application without any restrictions.

### Class Diagram

The diagram below illustrates basic relationship between interfaces and classes within cluster service.



## Cluster Node

The developer view on every cluster node is accommodated via the **ClusterNode** interface. Different collections of cluster nodes can be retrieved using various cluster node filters, which will be described in detail in the following paragraphs. There are also methods provided to obtain the collection of all nodes in the cluster or to get a handle to only the local cluster node.

The cluster service distinguishes between three types of cluster nodes:

- Local node – node this instance of xTier™ is running on.
- Localhost node – node running on the same physical IP address as local node but not the local node.
- Remote node – node running on other, remote IP address. Remote nodes are nodes that are not local or localhost nodes.

The **ClusterNode** API provides access to the following information about the cluster node:

- IP address and port
- number of CPUs and overall CPU utilization
- operation system information
- node type and ID
- group memberships to which this node belongs.

Additionally, there is a method **checkFailureStatus()** which provides the ability to check the failure status of the node at any given point of time.

## Cluster Node Failure Resolver

The xTier™ Cluster service comes with pluggable node failure resolution. All nodes in the xTier™ cluster must exchange heartbeats at a predefined time interval. When a certain number of heartbeats are missed, a node is suspected of a failure. In this case the **ClusterNodeFailureResolver** implementation provided by the end user is used to determine the life status of the node suspected of failure. If the node has indeed failed, then it is removed from the cluster and appropriate event notifications are triggered.

The xTier™ Cluster Service provides two predefined node failure resolvers out of the box:

- **ClusterBasicFailureResolver**, based on its initial configuration, will detect a node as always failed or always alive. Due to its simplistic nature, the basic failure resolver is usually not suited for production and its anticipated use is primarily for debugging and testing.
- **ClusterTcpNodeFailureResolver** uses a TCP/IP connection to check the life status of remote node. It assumes that if a connection to remote node could be established and the node was not restarted then the node is alive. This failure resolver should be sufficient for most enterprise applications.

## Cluster Filters

Cluster filters are used to select which nodes should be returned to the user. Essentially, for every node in the cluster, the implementation will check if the node is accepted by the filter, and if it is, it will be added to the returned collection. The user is free to implement any custom cluster filter.

The xTier™ Cluster service provides the following cluster filters out of the box:

- **ClusterAddressFilter** - accepts all nodes for a specified IP address.
- **ClusterGroupFilter** - accepts all nodes that belong to the given group with the given group properties.
- **ClusterNodeTypeInfoFilter** - allows for selecting between local node, localhost nodes, and remote nodes.
- **ClusterServiceFilter** - accepts all nodes that have the specified xTier™ service running.

All predefined cluster filters can take a nested cluster filter as a parameter, so it is possible to construct any possible combination of cluster filters.

## Cluster Groups

The notion of a cluster group is a powerful part of the xTier™ Cluster Service. Through cluster group memberships a node can be attached to any user-defined cluster group, although xTier™ does not require that a node become a member of any cluster group. Every group membership can have different group attributes. The developer specifies if a node should belong to any cluster group and what attributes should be associated with such membership through the XML configuration files. Specifying different cluster groups for different sets of nodes enables the developer to define many virtual clusters within the physical xTier cluster. Nodes belonging to different cluster groups with different attributes can be retrieved using the cluster group filter (**ClusterGroupFilter**) shipped with xTier™.

The following is a sample of cluster group membership configuration. Note that the group names and properties below are simply provided for demonstration purposes. xTier™ does not require for any node to become a member of any cluster group.

```
...
<!-- Cluster groups this node belongs to. -->
<memberships>
  <group name="cache"/>

  <group name="grid">
    <prop name="algorithm" value="bfs"/>
  </group>
</memberships>
...
```

## Optimistic Topology Version Control

Cluster topology is dynamic. Nodes can join or leave the cluster at any given point of time. When working with cluster nodes it is often convenient to know whether the cluster node topology has changed within a certain period of time. To

accommodate such a need, the xTier™ Cluster Service provides topology version number which is guaranteed to change every time cluster topology changes. This enables the developer to check for cluster topology changes optimistically. Prior to performing operations on cluster nodes, the developer can obtain the current version number. Then at any point afterwards, the developer can check whether the topology version number has changed and if it has, take appropriate action.

The example bellow demonstrates how to optimistically check cluster topology version.

```
// Get version of cluster topology..
int version = cluster.getTopologyVersion();

// Sleep for a short period of time.
Thread.sleep(100);

// Implement logic that handles topology change.
if (version != cluster.getTopologyVersion()) {
    // Take appropriate action.
    ...
}
```

## Usage

The following code snippet demonstrates a simple example using the xTier™ Cluster service API.

```
// Get the instance of xTier kernel.
XtierKernel xtier = XtierKernel.getInstance();

// Get the instance of 'cluster' service.
ClusterService cluster = xtier.cluster();

// 1.
// Get local node.
ClusterNode localNode = cluster.getLocalNode();

// 2.
// Get remote nodes ('false' for local node, 'false'
// for localhost nodes, 'true' for remote nodes).
Set remoteNodes = cluster.getNodes(new ClusterNodeTypeFilter(false, false, true));

// 3.
// Get remote nodes that run 'cache' service.
Set cacheRemoteNodes = cluster.getNodes(new ClusterServiceFilter("cache", new ClusterNodeTypeFilter(false, false, true)));

// 4.
// Get group memberships for the local node.
Map grpMshps = localNode.getGroupMemberships();
```

```
// Stop the xTier kernel.  
xtier.stop();
```

Let us briefly examine some parts of the code snippet above.

```
// Get local node.  
ClusterNode localNode = cluster.getLocalNode();
```

This line obtains a handle on the local cluster node.

```
// Get remote nodes ('false' for local node, 'false'  
// for localhost nodes, 'true' for remote nodes).  
Set remoteNodes = cluster.getNodes(new ClusterNodeTypeFilter(false, false, true));
```

This snippet demonstrates how to use a cluster node type filter to retrieve the set of remote nodes.

```
// Get remote nodes that run 'cache' service.  
Set cacheRemoteNodes = cluster.getNodes(new ClusterServiceFilter("cache", new  
ClusterNodeTypeFilter(false, false, true));
```

This section demonstrates the use of a combination of **ClusterServiceFilter** with **ClusterNodeTypeFilter** in order to retrieve all remote nodes that have the 'cache' service running.

```
// Get group memberships for the local node.  
Map grpMshps = localNode.getGroupMemberships();
```

This line obtains a map of all group memberships the local node belongs to.

## Conclusion

xTier™ provides the developer with rich yet simple functionality for creating clustered applications. Unlike J2EE, the cluster API is not hidden deep within the implementation, but is provided for the developer. The xTier™ cluster service is robust and extensible and is, in fact, utilized by xTier™ internally. The xTier™ Cluster Service is ideally suited for application development requiring a cluster without imposing a complicated API, or the necessity to write Clustering functionality in-house.

### Fitech Laboratories Inc.

#### Corporate Headquarters

Century Plaza, Suite 405  
1065 East Hillsdale Boulevard  
Foster City, CA 94404  
USA

Phone: 1-650-578-0808  
Fax: 1-650-578-0805

#### East Coast Sales Office

330 Madison Ave., 9th floor  
New York, NY 10017  
USA

Phone: 1-646-495-5076

#### Fitech Laboratories Japan

Toranomon40 MT Bldg. 3F  
5-13-1 Toranomon Minato-ku  
Tokyo 105-0001, Japan

Phone: +81-3-5402-7711  
Web: [www.fitechlabs.co.jp](http://www.fitechlabs.co.jp)