Multi-Node XNAT Considerations (The Node/Task Framework)

If you are developing functionality that might be used an a multi-node environment, it is important to consider whether your code might have issues operating in a multi-node XNAT installation environment. Setup and configuration of a multi-node XNAT environment is described here.

Most code in XNAT and most code you're likely to develop will work fine as-is in a multi-node environment. Users will trigger the code by issuing URLs to a specific web application instance either via the XNAT UI or via script or command line. That instance will then execute the code and carry out its task. Other code, however, may run as a listener, waiting for things to happen in the system and responding or as a scheduled task that triggers at specific intervals. It is important to consider whether, in a multi-node setup, multiple instances will try to carry out these requests simultaneously and whether that could cause problems.

In an out-of-the box XNAT installation an example of this type of task is the *session xml rebuilder*, which builds new imaging sessions from recently arrived images in the prearchive. The Node/Task Framework is included in XNAT 1.7 to simplify coding such tasks and assist in making a decision whether or not a given node should perform the task.

Coding Your Task

It should be relatively simple to code your task in a way so that it is configurable and executes safely in a multi-node environment. We'll use the code for the Session XML Rebuilder as the code sample. You can find the full source code here. The relevant lines of code are included below:

SessionXMLRebuilder

```
@XnatTask(taskId = "SessionXMLRebuilder", description = "Session XML Rebuilder", defaultExecutionResolver =
"SingleNodeExecutionResolver", executionResolverConfigurable = true)
public class SessionXMLRebuilder extends AbstractXnatTask implements Runnable {

    @Override
    public void run() {

                // Other stuff not shown

       if (!shouldRunTask()) {
             logger.trace("Session XML rebuilder not configured to run on this node.  Skipping.");
             return;
        }
        logger.trace("Running prearc job as {}", user.getLogin());

        // Other stuff not shown (perform task)

    }
```

Key implementation details:

- Specify @XnatTask annotation
    - This allows XNAT to discover your task so that administrators can configure where your task executes in their environment.
    - Administrators will be able to configure which nodes should run your task under the "XNAT Task Settings" administration page. Your task will automatically show up as one of the XNAT tasks to be configured.
- Extend AbstractXnatTask
    - This provides implementation for the shouldRunTask() method that will use the configured execution resolver and the configuration supplied by the local site to determine whether the current node should run the task.
    - If you are unable to extend AbstractXnatTask, you can instead implement the XnatTaskI interface that the AbstractXnatTask class implements and supply implementation for the required methods.
- Call shouldRunTask() method to determine whether to continue performing task or return without performing the task.

Execution Resolvers

Execution resolvers do the work of deciding whether the current node should execute a task. In an out-of-the-box XNAT installation, there currently are two execution resolvers;

1. SingleNodeExecutionResolver - Administrators specify, via XNAT Task Settings configuration, a single node that should run the task.
2. SingleNodeExecutionResolverWithFailover - Administrators specify a list of nodes that can handle the task. If one is not reachable and has not executed the task within a configured span of time, the next node will run the task.

If these execution resolvers are not sufficient for your needs, you can write your own. We'll use the SingleNodeExecutionResolver as a code example. You can find the full source code here. The key lines of code are below:

SingleNodeExecutionResolver

```
@XnatTaskExecutionResolver(resolverId = "SingleNodeExecutionResolver",
        description = "Single Node Execution Resolver")
public class SingleNodeExecutionResolver implements XnatTaskExecutionResolverI {

        @Override
        public boolean shouldRunTask(final String taskId) {
                final String prefKey = "task-" + taskId + "-node";
                if (_preferences.getPreferenceKeys().contains(prefKey)) {
                        final Preference taskNodePref = _preferences.get(prefKey);
                        if (taskNodePref.getValue() != null && taskNodePref.getValue().trim().length()>0) {
                                return _xnatNode.getNodeId().equals(taskNodePref.getValue());
                        }
                }
                return true;
        }

        @Override
        public List<String> getConfigurationElementsYaml(final String taskId) {
                final List<String> eleList = Lists.newArrayList();
                final String ele =
                                "kind: panel.input.text\n" +
                                "label: " + taskId + " Execution Node\n" +
                                "name: task-" + taskId + "-node\n" +
                                "size: 45\n" +
                                "placeholder: nodeID\n" +
                                "description: Node on which to run this process. This configuration is not
required on a single-node XNAT installation, " +
                                        "however in a multi-node environment, where multiple XNAT instances
point to the same database, this configuration " +
                                        "should be in place in order to avoid conflicts when the task tries to
run simultaneously on multiple nodes. " +
                                        "Nodes are defined by setting a <em>node.id</em> property value in a
properies file called <em>node-conf.properties</em> located " +
                                        "in the same directory as your <em>xnat-conf.properties</em> file.\n";
                eleList.add(ele);
                return eleList;
        }

}
```

Key Implementation Details:

- Specify @XnatTaskExecutionResolver annotation.
    - This allows XNAT to discover your execution resolver.
    - It will automatically be one of the execution resolvers administrators can select from to configure their tasks to run.
- Implement XnatTaskExecutionResolverI interface.
    - You'll need to supply implementation for the shouldRunTask() and getConfigurationElementsYaml() methods.
    - The shouldRunTask() method is the main method where a determination is made as to whether or not the current node should process the task.
    - In the getConfigurationElementsYaml() method, you'll return the configuration UI elements required to configure your resolver. Presumably, the preference values users will set here will be referenced in the shouldRunTask() method to determine whether or not the current node should process the task.