Callable Example

```
public static void testThread() {
    try {
            HashMap<String, String> states = new HashMap<String, String>();
        View allStates = Factory.getSession(SessionType.CURRENT).getCurrentDatabase().getView("AllStates");
        for (Document doc : allStates.getAllDocuments()) {
            states.put(doc.getItemValueString("Key"), doc.getItemValueString("Name"));
        }

        // Create a callable for each State
        List<Future<String>> results = new ArrayList<Future<String>>();
        for (String key : states.keySet()) {
            UserLookup getUsers = new UserLookup(key, states.get(key));
            results.add(Xots.getService().submit(getUsers));
        }

        // Now loop through the Xots tasks, and get the results
        TreeSet<String> output = new TreeSet<String>();
        for (Future<String> f : results) {
            output.add(f.get());
        }
        ExtLibUtil.getViewScope().put("MessageFromIterableXots", output);
    } catch (Throwable t) {
            XspOpenLogUtil.logError(t);
    }
}

@Tasklet(session = Tasklet.Session.NATIVE, context = Tasklet.Context.DEFAULT)
private static class UserLookup extends AbstractXotsCallable<String> {
        private String state_;
    private String stateName_;

    public UserLookup(String state, String stateName) {
            state_ = state;
            stateName_ = stateName;
    }

    public String call() {
            try {
                StringBuilder sb = new StringBuilder();
            View people = Factory.getSession(SessionType.CURRENT).getCurrentDatabase()
                    .getView("AllContactsByState");
            System.out.println("Processing " + state_);
            sb.append(stateName_ + ": ");
            for (Document doc : people.getAllDocumentsByKey(state_, true)) {
                    String name = doc.getItemValueString("FirstName") + " " + doc.getItemValueString
("LastName");
                sb.append(name);
                sb.append(", ");
            }
            return sb.substring(0, sb.length() - 2) + "<br/>";
        } catch (Throwable t) {
            XotsUtil.handleException(t, getContext());
            return t.getMessage();
        }
    }
}
```

Lines 1 - 25 are a utility method to trigger the Xots tasks, aggregate the output and post it to viewScope.

Lines 3 - 7 create a HashMap of the state key (e.g. FL for Florida) and names. This runs as the current user.

Line 10 creates a List to contain the Futures, each of which is a tasklet to kick off and wait for the response. The type for the Futures is set as a String.

Lines 11 - 14 iterate the HashMap and create a new UserLookup object, passing in the relevant state key and state name. The object is then passed to the Xots service and loaded into the List created on line 10. As we'll see later, the result of the tasklet is a String. Ten will run at any one time, the others sitting in the queue to start as and when a Xots thread is available.

Line 17 creates a TreeSet. Using a TreeSet enables the results to be sorted alphabetically, because each String will start with the state name. This needs to be done because although the threads are kicked off in state order, they may not complete in state order. Some may complete quicker than others.

Lines 18 - 20 iterate through the List of Futures and as and when a tasklet completes, it will be processed. `f.get()` gets the result of the tasklet and, as we'll see later, that's a String.

Line 21 posts the TreeSet of Strings to the viewScope variable.

Lines 28 onwards are the Xots tasklet, in this case a Callable. Because we don't need access to any scoped variables or XPages runtime, we just extend the more limited `AbstractXotsCallable` class. This still has access to the database etc required for error handling.

Line 27 sets the session as `Tasklet.Session.NATIVE` which means the tasklet will run as the server. The context is set as `Context.DEFAULT`, because no XPages scopes need passing into the tasklet.

Lines 29 and 30 declare two properties, `state_` and `stateName_`, to hold the state key and state name.

Lines 32 - 35 store the variables passed into the properties of this object. This means that they can be used in the `call` method.

Lines 37 onwards are the `call` method. A class that implements Callable will need to have this method, it's the one that is triggered when Xots runs the tasklet. Because this is a Callable - a background task that is kicked off and monitored for a response - it needs to return something, in this case a String.

Lines 39 - 48 create a StringBuilder, get the view and iterate the documents for the relevant state. The view is retrieved using `SessionType.CURRENT`, but because the tasklet is running as the server, that is in reality the same as `SessionType.NATIVE`. The names of all contacts for the state are output to the StringBuilder.

Line 49 returns the names.

Lines 50 - 53 do the error handling, writing to OpenLog (like the Runnable) and returning the error message.