Table of Contents

Programmers Reference, Custom Processes

Adding custom processes

The following section describes how to write extensions implementing custom processes and plug them into ***Aware IM***.

To implement custom processes and add them to the configuration of your application follow the steps below:

1. Write the code for the process, compile it and add it to the jar file with your custom extensions. Make sure that the jar file is placed in the `AwareIM/CustomJars` directory (see section 2)
2. Add the process to your application configuration using the Configuration Tool (see Adding/Editing Processes section). When adding the process click on the "Implementation" property in the list of process properties and select the "Process is implemented by custom component" radio button. Then specify the fully qualified name of the class that implements the component, for example `com.myextensions.MyProcessExtension`

The rest of the section describes how to write the code implementing the custom process (step 1 above).

Writing code for a custom process component

When writing a process it is important to understand how processes fit into the architectural framework of ***Aware IM***. When a process is started ***Aware IM*** creates the *execution context*, which contains the information about the environment in which the process is running. This information contains the data about the user who started the process, the process start-up time, the information about the current database transaction as well as many other things. The execution context is available to the process as the IExecutionContext interface. Because a process in ***Aware IM*** runs in a "sandbox" there are certain limitations on what the process can do (see guidelines later in this section).

A process communicates with ***Aware IM*** through the call-back interface called IExecutionEngine. This interface has a variety of methods that a process can use to change attributes of objects, start other processes, call services of service providers etc.

Sometimes a particular process calls a service, which takes a long time to fulfil. In this case the call to the service may time out. If this happens the process is *suspended*. This means that ***Aware IM*** moves the process into the "standby" mode – it is no longer active and is waiting for the reply from the service provider. Whenever such reply arrives (the reply may arrive in a few seconds, hours, days or months or never arrive at all) ***Aware IM*** automatically *resumes* the process and the process continues execution from where it left off.

The custom process component must implement `IProcess` interface, which has two main methods – execute and resume. The execute method is called when the process is started and the resume method is called when the process is resumed after having been suspended. Both methods are described in detail later in the section.

The following guidelines should be used when writing a process:

- A process is not supposed to perform any operations with the database directly (such as create, open or modify database tables, manage transactions etc) – this is taken care of by the ***Aware IM*** framework.
- A process generally should not take long to execute as it is running within a context of a database transaction. If a process does take a long time to execute it should every now and again send a special signal to the ***Aware IM*** framework. Having received such a signal ***Aware IM*** would suspend the process, commit the current transaction and continue the process execution (resume the process). This signal can be sent if the process throws `SuspendProcessException` and sets its `resumeImmediately` flag to true, for example:

  ```
  throw new SuspendProcessException (true)
  ```

  Before doing so the process should obviously remember its current state so that it can correctly resume itself in the resume method.

- If a process calls a method of the IExecutionEngine interface, which throws `ServerTimeOutException` the process is supposed to catch this exception and throw `SuspendProcessException` passing the original `ServerTimeOutException` to the `SuspendProcessException`, for example:

  ```
  try
  {
      engine.updateEntity (this, entity, null, null);
  }
  catch (ServerTimeOutException se)
  {
      throw new SuspendProcessException (se);
  }
  ```

The methods of the `IProcess` interface that the custom process components must implement are described below:

- Appendix A: correspondence of attribute types and Java types