

The NetBpm Samples

Table of contents

1 Hello world 1.....	2
2 Hello world 2.....	5
3 Hello world 3.....	6
4 Hello world 4.....	7

1. Hello world 1

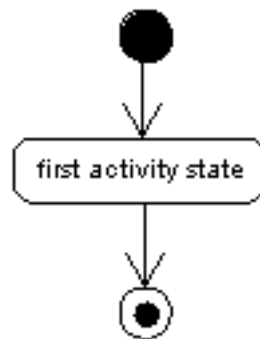
FIXME (pbolle):

write a introduction

FIXME (pbolle):

add the location of the examples

Business processes in NetBpm are modelled and expressed in xml and other files. All the files that define one business process are zipped together in what we call a process archive (see also [Process Archives](#)) So lets build our first process archive. We'll start with the simplest possible case : one start-node, one activity-state and one end-node.



The simplest business process

The simplest business process

First I'll explain what can be seen on the diagram : The upper full circle is the start-state. When a process instance is started in the NetBpm-engine, the runtime-state is calculated. In this case it will be the engine will put the root-flow of the new process instance in the activity state 'first activity state'. Activity-states are states like in a state machine. The NetBpm-engine will make sure that state-changes occur transactional. So creating an instance of the process 'Hello World 1' will cause the engine to put the root-flow in the activity state 'first activity state' and then wait for an external trigger. Providing the external trigger is what we call performing the activity. When a user or system provide this trigger, the engine will calculate the next state of the process instance in a transactional way. In case of the activity-state 'first activity state' the next state will be the end state which will cause the process instance to finish. If you are asking yourself "what about actions which are executed

automatic initiated by the business process ?", see hello world 2.

Note that the core NetBpm engine is a component. The component has an API but no graphical interface. That is why we build a web-application that exposes the core NetBpm API's to web-users. In the process archive the web-application information is completely separated from the information that describes the business process logic. So note that all information web-app related information that we will present here is optional.

The first file to write is the `processdefinition.xml` file. That one contains the logic of the business process. Here's the `processdefinition.xml` of the `helloworld1.par`.

```
<?xml version="1.0"?>
<process-definition>
  <name>Hello world 1</name>
  <description>This is the simples process.</description>
  <start-state name="start">
    <transition to="first activity state" />
  </start-state>
  <end-state name="end" />
  <activity-state name="first activity state">
    <description>this is the first state</description>
    <assignment
handler="NetBpm.Workflow.Delegation.Impl.Assignment.AssignmentExpressionResolver,
NetBpm">
      <parameter name="expression">processInitiator</parameter>
    </assignment>
    <transition to="end" />
  </activity-state>
</process-definition>
```

the processdefinition.xml for the Hello World 1 process

The `processdefinition.xml` is in fact all the NetBpm engine needs to know about your process. But to allow you to execute this process we'll make use of the NetBpm web-application. Therefor we will add 2 files to the process archive : `web/helloworld1.gif` and `web/webinterface.xml`. The `web/helloworld1.gif` is the image that you see above. The `web/webinterface.xml` provides the engine with a little extra information so it can generate the forms for performing activities. This is how the `web/webinterface.xml` of the `helloworld1.par` looks like :

```

<?xml version="1.0"?>

<web-interface>

  <image name="web/helloworld1.gif" mime-type="image/gif"
width="104" height="155" />

  <!-- ===== -->
  <!-- == ACTIVITIES == -->
  <!-- ===== -->
  <activity-state name="start">
    <image-coordinates>
      <x1>43</x1>
      <y1>0</y1>
      <x2>65</x2>
      <y2>21</y2>
    </image-coordinates>
  </activity-state>

  <activity-state name="first activity state">
    <image-coordinates>
      <x1>0</x1>
      <y1>51</y1>
      <x2>101</x2>
      <y2>78</y2>
    </image-coordinates>
  </activity-state>

</web-interface>

```

the optional web-form-generation

The 3 files `processdefinition.xml`, `web/helloworld1.gif` and `web/webinterface.xml` are then zipped together in a process archive with the following layout :

```

/root
|
+- processdefinition.xml
|
+- web
  +- webinterface.xml
  +- helloworld1.gif

```

the helloworld1.par process archive

This process archive can now be deployed in the NetBpm engine. Deploying means that the information in the process archive is parsed and stored in the NetBpm database. The NetBpm

runtime engine will use process definition information in the database to manage the process executions. The state of the process executions is managed in different tables of the same database.

2. Hello world 2

The second hello world example will add an action to the first hello world process. An action is an automatic action that has to be executed on an event that is triggered related to the process execution. Typical actions would be sending emails, putting a message on the message broker, calling a web-service somewhere, ... For flexibility, a process developer can provide his own ActionHandler implementations in the process archive. This means that a process developer implements the ActionHandler interface

```
public interface IActionHandler {
    void Run( IActionContext actionContext );
}
```

The ActionHandler interface

and puts the .dll file directly in the process archive or in the path of NetBpm.

Note:

dynamic loading of assemblies is not yet supported

Since this manual is targetted at developers, we have chosen for a very meaningfull :-) action that prints 'HELLO WORLD' to the logfile :

```
public class HelloWorldAction : IActionHandler
{
    private static readonly ILog log =
LogManager.GetLogger(typeof (HelloWorldAction));

    public virtual void Run(IActionContext actionContext)
    {
        // Lets show that we have been here :- )
        log.Debug(" ");
        log.Debug("H H EEEEE L L OOO W
W OOO RRRR L DDDD ");
        log.Debug("H H E L L O O W
W O O R R L D D");
        log.Debug("HHHHH EEE L L O O W
W O O R R L D D");
        log.Debug("H H E L L O O W W
W O O RRRR L D D");
        log.Debug("H H E L L O O W W
```

```

W O O R R L D D");
OOO R R LLLLL DDDD ");
log.Debug("H H EEEEE LLLLL LLLLL OOO W W");
log.Debug(" ");
// The next log message will be stored in the
netbpm-logs of the flow.
actionContext.AddLog("HELLO WORLD");
}
}

```

The HelloWorldAction implementation

Note that you can do much more meaningful stuff in an ActionHandler implementation because you can access the process instance attributes (see hello world version 3) on the one hand and your complete IT-environment. So ActionHandlers can be used for 2-way communication between the process and e.g. an SAP-system, a database, ...

In the `processdefinition.xml`, the action will be scheduled on the transition from 'first activity state' to the end-state :

```

...
<activity-state name="first activity state">
  <description>this is the first state</description>
  <assignment
handler="NetBpm.Workflow.Delegation.Impl.Assignment.AssignmentExpressionResolver,
NetBpm">
    <parameter name="expression">processInitiator</parameter>
  </assignment>

  <transition to="end">
    <action event="transition"
handler="NetBpm.Example.Delegate.HelloWorldAction, NetBpm.Example" />
  </transition>
</activity-state>
...

```

the changes in processdefinition.xml for the Hello World 2 process

3. Hello world 3

The third version of the Hello World process will add 2 attributes to the process. Attributes are also called process-variables. The nice thing about attributes in NetBpm is that they are simple .Net objects. This means that you are not bound to a predefined set of types. NetBpm handles persistent storage of attribute values by means of `Serializers`. `Serializers` convert

the values in the attributes (the POJO's) to text and also the other way round :

```
public interface ISerializer : IConfigurable
{
    String Serialize(Object valueObject);
    Object Deserialize(String text);
}
```

The Serializer interface

The basic way to specify an attribute in the `processdefinition.xml` is by specifying the serializer implementation class-name. However, NetBpm contains a number of default serializers for the most standard types. These can be specified as shortcuts with the `type` attribute. see the nPdl reference documentation for learning which type names correspond to which serializer implementation.

This is how to specify attributes in the `processdefinition.xml` :

```
...
<attribute name="evaluation result"
serializer="NetBpm.Workflow.Delegation.Impl.Serializer.EvaluationSerializer,
NetBpm" />
<attribute name="the text attrib"
type="text"
initial-value=":-)" />
...
```

Specifying attributes in the processdefinition.xml

4. Hello world 4

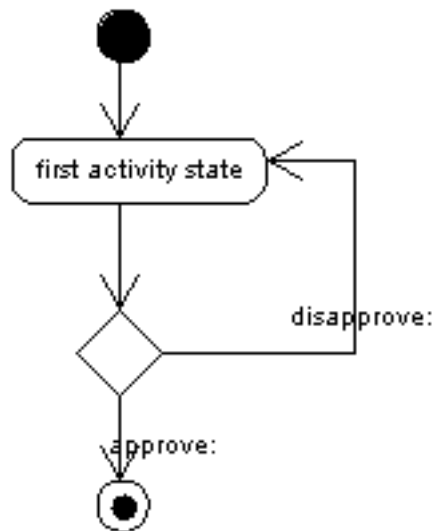
The forth version of the hello world process adds a decision. Decisions are also based on an interface : The `DecisionHandler` interface :

```
public interface IDecisionHandler
{
    String Decide(IDecisionContext decisionContext);
}
```

The DecisionHandler interface

Whenever the execution arrives at a decision, a DecisionHandler implementation will be called. As for all delegation implementation classes, in a process definition you can use the NetBpm provided default implementations or you can write your custom DecisionHandler implementation and add it in the process archive.

In this example we will use the `NetBpm.Workflow.Delegation.Impl.Serializer.EvaluationSerializer`, which is a default implementation available in NetBpm. (so we can use it without including it in the process archive). The EvaluationDecision will decide on the basis of the 'evaluation result' attribute value. If the decision result contains an approval, the process is finished; otherwise, the process is put back in the 'first activity state'. This can be seen in the graph :



The simplest business process extended with a decision

The simplest business process extended with a decision

This is how we express the decision in the `processdefinition.xml` :

```

...
<activity-state name="first activity state">
  ...
  <transition to="the looping decision">
    <action event="transition"
handler="NetBpm.Example.Delegate.HelloWorldAction, NetBpm.Example" />
  </transition>
</activity-state>

```

```
<decision name="the looping decision"  
handler="NetBpm.Workflow.Delegation.Impl.Decision.EvaluationDecision,  
NetBpm">  
  <parameter name="attribute">evaluation result</parameter>  
  <transition name="disapprove" to="first activity state" />  
  <transition name="approve" to="end" />  
</decision>  
  
...
```

Specifying a decision in the processdefinition.xml

The decision also shows another interesting characteristic : The delegation classes can be configured. So it is possible to pass parameters via the processdefinition.xml to the delegation implementation objects. This mechanism enables the writing of reusable delegation classes.