How To...

Copyright © 2006 Together Teamlösungen EDV-Dienstleistungen GmbH

Table of Contents

You can find here answers to some frequently asked questions.

Database

How to configure Shark to work with another database?

After setup procedure finishes you'll have HipersonicSQL database created. While this is quite usable, Shark provides you an option to use other database vendor: *DB2*, *PostgreSQL*, *MySQL*,....

- first you'll need to stop any Shark instance that may be running.
- Edit the `configure.properties` file and set values for:

`db_loader_job`	name of the directory containing Octopus loader job, options are: *db2*, *hsql*, *informix*, *msql*, *mysql*, *oracle*, *postgresql*, *sybase*
`db_ext_dirs`	directory containing jar file(s) with JDBC driver, if you need more then one directory specified here - use `${path.separator}` to concatenate them
`${db_loader_job}_JdbcDriver`	classname of the *JDBC* driver you want to use These entries are already filled with default values.
`${db_loader_job}_ConnectionUrl`	full database *URL*

These entries are already filled with default values, too.

`${db_loader_job}_user`	username for database authentication
`${db_loader_job}_passwd`	password for database authentication

- run the `configure.[bat|sh]`

> **Note**
>
> When loading newly created database, Octopus will complain about not being able to drop indices and tables, but theses warnings should be ignored.

At this time, sharkdb.properties file(that is placed in lib/client folder) and Shark.conf are adjusted to use selected database.

How to clear Shark database?

In the process of testing there will come the point you'll want to clear the database and start from the scratch. For clearing the database you may run the main `configure.[bat|sh]` file. If you don't want to wait for unnesessary filtering, archiving of `war` - you have `bin/recreateDB.[bat|sh]`.

The latter method runs only Octopus loader job to drop and create tables and indices.

Client interface

How to use Shark library

Client application uses Shark library through a set of interfaces in `org.enhydra.shark.api.client` package. The first thing the application should do is to configure library either by calling `configure()` method by specifying filename (as `String` or `File` object) or by preparing and calling the method with a `Properties` object. After configuration `org.enhydra.shark.Shark.getInstance()` returns an instance of `SharkInterface`. From this point on, it's application developer preference (or task) how it would use library, either getting a connection and executing assignments or getting some of the administrative interfaces (PackageAdministration, ExecutionAdministration, AdminMisc, ...) and managing packages, process instances, ...

Example 1. Not very useful work-list handler

First line of this example configures engine using `conf/Shark.conf` file. Then after getting a connection and successfully connecting it asks `Resource` object how many assignments there are for this user (in line 4).

```
      UserTransaction ut = (UserTransaction) new InitialContext().lookup("java:comp/
UserTransaction");
      ut.begin();
      Shark.configure("c:/Shark/conf/Shark.conf");
      ut.commit();

      ut.begin();
      WMConnectInfo wmci=new WMConnectInfo("test", "", "", "");
      SharkConnection sConn = Shark.getInstance().getSharkConnection();
      sConn.connect(wmci);
      if (0 < sConn.getResourceObject().how_many_work_item()) {
          System.err.println("Oh, let these tasks wait until tomorrow!");
      }
      ut.commit();

      System.out.println("Job done!");
```

NOTE: Shark 2.0 is JTA oriented, and thus when shark works outside container which provides JTA TransactionManager, we have to start one by our own. Also, in shark 2.0 for defining database we work with, we use DataSource which should be registered in JNDI, and thus when working outside container we also need to take care about registering data source in JNDI. For the purpose of stand-alone shark usage we made LocalContextFactory which is implementation of InitialContextFactory interface, and which purpose is to: 1. start TransactionManager 2. provide a JNDI context - register TransactionManager in JNDI context (so we can afterwards obtain TransactionManager and UserTransaction from JNDI) - register DataSource in JNDI context So, when using shark outside container before executing code above, you need to call setup method on LocalContextFactory class (read shark.pdf document to see how to start shark).

Example 2. Loading package into Shark library

First you need the location of the package's XPDL file, then you need a `PackageAdministation` instance.

```
      UserTransaction ut = (UserTransaction) new InitialContext().lookup("java:comp/
UserTransaction");
      ut.begin();
      Shark.configure("c:/Shark/conf/Shark.conf");
      ut.commit();

      ut.begin();
      WMConnectInfo wmci=new WMConnectInfo("test", "", "", "");
      SharkConnection sc = Shark.getInstance().getSharkConnection();
      sc.connect(wmci);
      WMSessionHandle sh=sc.getSessionHandle();
      ut.commit();

      String xpdlName = "c:/test.xpdl";

      ut.begin();
      PackageAdministration pa = Shark.getInstance().getPackageAdministration();
      if (!pa.isPackageOpened(sh,pkgId)) {
         pa.openPackage(sh,xpdlName);
      }
      ut.commit();
      System.out.println("Package " + xpdlName + " is loaded");
```

Example 3. Creating and starting process

After loading XPDL into shark, lets create, fill with initial variable values, and start some processes based on their XPDL definitions.

```
      String pkgId = "test";
      String pDefId1 = "basic";
      String pDefId2 = "complex";

      UserTransaction ut = (UserTransaction) new InitialContext().lookup("java:comp/
UserTransaction");
      ut.begin();
      Shark.configure("c:/Shark/conf/Shark.conf");
      ut.commit();

      ut.begin();
      WMConnectInfo wmci=new WMConnectInfo("test", "", "", "");
      SharkConnection sc = Shark.getInstance().getSharkConnection();
      sc.connect(wmci);
      ut.commit();

      ut.begin();
      XPDLBrowser xpdlb = Shark.getInstance().getXPDLBrowser();
      String procDefName = xpdlb.getUniqueProcessDefinitionName(sc.getSessionHandle(),
                                                                pkgId,
                                                                "",
                                                                pDefId);
      WfProcessMgr mgr = sc.getProcessMgr(procDefName);
      ut.commit();

      ut.begin();
      WfProcess proc1 = mgr.create_process(null);
      Map m1 = new HashMap();
      m1.put("test_var",
            "This is String variable defined in XPDL for the process basic");
      proc1.set_process_context(m1);
      proc1.start();
      ut.commit();

      ut.begin();
      WfProcess proc2 = mgr.create_process(null);
      Map m2 = new HashMap();
      m2.put("counter", new Long(55));
      proc2.set_process_context(m2);
      proc2.start();
      ut.commit();
```

Example 4. Setting a variable

After successfully connecting to Shark, and acquiring list of assignments, lets do something useful, like setting a variable and completing the activity.

```
/*
SharkConnection sConn;
String activityId;
String vName;
String vValue;
 */
WfAssignment a = null;
ut.begin();
String uname=sc.getResourceObject().resource_key();
WfAssignment[] ar = sc.getResourceObject().get_sequence_work_item(0);
for (int i = 0; i < ar.length; ++i) {
   if (activityId.equals(ar[i].activity().key())) {
      a = ar[i];
      break;
   }
}
ut.commit();

if (null == a)
   throw new Exception("Activity:"
                           + activityId
                           +" not found in "
                           + uname
                           +"'s worklist");

ut.begin();
boolean as=a.get_accepted_status();
if (!as) {
   a.set_accepted_status(true);
}
ut.commit();


ut.begin();
Map _m = new HashMap();
WfActivity activity = a.activity();
Object c = activity.process_context().get(vName);
if (c instanceof Long) {
   c = new Long(vValue);
} else {
   c = vValue;
}
_m.put(vName, c);
activity.set_result(_m);
activity.complete();
ut.commit();
```

Example 5. Getting process managers based on some criteria

This example shows how to get process managers based on some criteria. It'll try to get all process managers which package Id is "test", and which state is enabled.

```
      UserTransaction ut = (UserTransaction) new InitialContext().lookup("java:comp/
UserTransaction");
      ut.begin();
      Shark.configure("c:/Shark/conf/Shark.conf");
      ut.commit();

      ut.begin();
      WMConnectInfo wmci=new WMConnectInfo("user", "secret", "", "");
      SharkConnection sc = Shark.getInstance().getSharkConnection();
      sc.connect(wmci);
      ut.commit();

      ut.begin();
      WfProcessMgrIterator pmi = eAdmin.get_iterator_processmgr();
      String query = "packageId.equals(\"test\") && enabled.booleanValue()";
      pmi.set_query_expression(query);
      WfProcessMgr[] procs = pmi.get_next_n_sequence(0);
      ut.commit();
```

Another approach is to use so called Filter builders to create expression that will be (in this case) executed directly on DB:

```
      UserTransaction ut = (UserTransaction) new InitialContext().lookup("java:comp/
UserTransaction");
      ut.begin();
      Shark.configure("c:/Shark/conf/Shark.conf");
      ut.commit();

      ut.begin();
      WMConnectInfo wmci=new WMConnectInfo("user", "secret", "", "");
      SharkConnection sc = Shark.getInstance().getSharkConnection();
      sc.connect(wmci);
      WMSessionHandle shandle=sc.getSessionHandle();
      ut.commit();

      ut.begin();
      WfProcessMgrIterator pmi = sc.get_iterator_processmgr();
      ProcessMgrFilterBuilder fb=Shark.getInstance().getProcessMgrFilterBuilder();
      WMFilter f=fb.addPackageIdEquals(shandle, "test");
      WMFilter f2=fb.addIsEnabled(shandle);
      f=fb.and(shandle, f, f2);

      String query = fb.toIteratorExpression(shandle, f);
      pmi.set_query_expression(query);
      WfProcessMgr[] procs = pmi.get_next_n_sequence(0);
      ut.commit();
```

Example 6. Getting processes based on some criteria

This example shows how to get processes created by some process manager based on some criteria. It'll try to get all processes which state is "open.running", which are started in last 10 minutes, which have more than 3 active activities, and which String variable called "myvariable" has value "test".

```
      /*
      WfProcessMgr mgr;
      WMSessionHandle shandle;
       */
      ut.begin();
      WfProcessIterator wpi = mgr.get_iterator_process();
      String query = "state.equals(\"open.running\") && "
                     +
 "startTime.longValue()>(java.lang.System.currentTimeMillis()-10*60*1000) && "
                     + "activeActivitiesNo.longValue()>3 && "
                     + "context_myvariable.equals(\"test\")";
      wpi.set_query_expression(query);
      WfProcess[] procs = wpi.get_next_n_sequence(0);
      ut.commit();
```

Another approach is to use so called Filter builders to create expression:

```
      /*
      SharkConnection sc;
      WMSessionHandle shandle;
       */
      ut.begin();
      WfProcessIterator wpi = sc.get_iterator_process();

      ProcessFilterBuilder fb=Shark.getInstance().getProcessFilterBuilder();
      WMFilter f=fb.addStateEquals(shandle, "open.running");
      WMFilter f2=fb.addStartTimeAfter(shandle,
(java.lang.System.currentTimeMillis()-10*60*1000));
      WMFilter f3=fb.addActiveActivitiesCountGreaterThan(shandle, 3);
      WMFilter f4=fb.addVariableEquals(shandle, "myvariable", "test");
      f=fb.and(shandle, new WMFilter[] {f, f2, f3, f4});

      String query = fb.toIteratorExpression(shandle, f);
      wpi.set_query_expression(query);
      WfProcess[] procs = wpi.get_next_n_sequence(0);
      ut.commit();
```

XPDL process definitions

(You can easily create XPDLs by using our XPDL editor JaWE [http://jawe.objectweb.org].)

How to write deadline expressions for activities?

In shark deadline expressions along with all process variables, you can use special variables. The Java type of these variables is java.util.Date, and here is their description:

- PROCESS_STARTED_TIME - the time when the process is started
- ACTIVITY_ACTIVATED_TIME - the time when process flow comes to activity and assignments for the activity are created
- ACTIVITY_ACCEPTED_TIME - the time when the first assignment for the activity is accepted

Note

If activity is being rejected after its acceptance, or it is not accepted at all, the ACTIVITY_ACCEPTED_TIME is set to some maximum value in the future.

Here are some rules when creating deadline expressions:

- Deadline expressions has to result in java.util.Date
- If shark is setup not to re-evaluate deadlines, but to initially evaluate deadline limit times, ACTIVITY_ACCEPTED_TIME should not be used in expressions because it will contain some maximum time in the future
- There shouldn't be process variables (DataField or FormalParameter entities from XPDL) that have the same Id as the one of previously listed.

Here are few examples of deadline expressions:

```
// Deadline limit is set to 15 secunds after accepting activity
var d=new java.util.Date();
d.setTime(ACTIVITY_ACCEPTED_TIME.getTime()+15000);
d;


// Deadline limit is set to 5 minutes after activity is started (activated)
var d=new java.util.Date();
d.setTime(ACTIVITY_ACTIVATED_TIME.getTime()+300000);
d;


// Deadline limit is set to 1 hour after process is started
var d=new java.util.Date();
d.setTime(PROCESS_STARTED_TIME.getTime()+3600000);
d;
```

How to write extended attributes to be able to update/view activity variables in shark's admin application

In order to update activity variable (defined by XPDL) in shark admin application(s), XPDL activity definition must contain some predefined extended attributes.

Suppose that XPDL process definition contains variable (XPDL DataField tag) called "x", and variable (XPDL FormalParameter type) called "input_var".

If while executing activity you would like admin user only to be able to view those variables, you should define following Activity's extended attributes:

```
<ExtendedAttribute Name="VariableToProcess_VIEW" Value="x"/>
<ExtendedAttribute Name="VariableToProcess_VIEW" Value="input_var"/>
```

If you would like user to update the same variables, you should define following Activity's extended attributes:

```
<ExtendedAttribute Name="VariableToProcess_UPDATE" Value="x"/>
<ExtendedAttribute Name="VariableToProcess_UPDATE" Value="input_var"/>
```

How to write XPDL to use custom Java classes as variables in Shark

To be able to do that, you should define variable as XPDLs external reference, and set its location attribute to be the full name of the Java class you want to use. I.e., it should look like:

```
...
<DataField Id="participants" IsArray="FALSE">
   <DataType>
      <ExternalReference location="org.enhydra.shark.wrd.Participants"/>
   </DataType>
</DataField>
...
...
<FormalParameter Id="participantGroup" Mode="INOUT">
   <DataType>
      <ExternalReference location="org.enhydra.shark.wrd.Participants"/>
   </DataType>
</FormalParameter>
...
```

Maybe the better approach is to define TypeDeclaration element that would be of that type. In that case you can use it everywhere (you do not need time to define appropriate DataType when creating Application's/ SubFlow's FormalParameters):

```
...
<TypeDeclaration Id="participants_type">
   <ExternalReference location="org.enhydra.shark.wrd.Participants"/>
</TypeDeclaration>
...
```

and than define DataField or FormalParameter as follows:

```
...
<DataField Id="participants" IsArray="FALSE">
   <DataType>
      <DeclaredType Id="participants_type"/>
   </DataType>
</DataField>
...
<FormalParameter Id="participantGroup" Mode="INOUT">
   <DataType>
      <DeclaredType Id="participants_type"/>
   </DataType>
</FormalParameter>
...
```

The classes specified by ExternalReference element must be in shark's classpath.

How to define in XPDL that initial value of some variable should be 'null'

You should simply write "null" for InitialValue element of DataField:

```
<DataField Id="participants" IsArray="FALSE">
   <DataType>
      <DeclaredType Id="participants_type"/>
   </DataType>
   <InitialValue>null</InitialValue>
</DataField>
```

This enables you to use interfaces or abstract java classes as workflow variables. Concrete implementation of these variables can be created by some tool agent.

How to specify scripting language

Shark currently supports three script interpreters: JavaScript, BeanShell and Python (the last one is not fully tested). To tell shark which scripting language is used for writting conditional expressions (i.e. in Transition conditions), you should specify Package's script element:

```
# if you want to use java-like syntax (interpreted by BeanShell), specify:

<Script Type="text/java"/>

# if you want to use java script syntax, specify:

<Script Type="text/javascript"/>

# if you want to use python syntax, specify:

<Script Type="text/pythonscript"/>
```

Shark will complain if you do not specify Script, or if you specify value not supported by shark.

How to use XPDL to directly map Application definition to particular ToolAgent (no need for Application mapping in runtime)

If you would like to specify directly in XPDL what particular ToolAgent will be executed by Tool activity, you should define some extended attributes for XPDL Application definition.

The main extended attribute that should be defined by each Application definition that tends to be mapped to ToolAgent has name "ToolAgentClass", and its value should be full name of the class representing tool agent to be executed, i.e.:

```
<ExtendedAttribute Name="ToolAgentClass"
 Value="org.enhydra.shark.toolagent.JavaScriptToolAgent"/>
```

This attribute is read by Default tool shark's tool agent, and he executes specified ToolAgent based on the value of this attribute.

Other extended attributes are specific to implementation of the tool agent, and are read by them. I.e., JavaScript and BeanShell tool agent specify extended attribute named "Script", and its content is the script to be executed by this tool agent at the runtime. In this case, you are actually using XPDL for programming, i.e.:

```
<ExtendedAttribute Name="Script" Value="java.lang.System.out.println(&quot;I'm
 going to perform operation c=&quot;+a+&quot;*&quot;
+b);&#10;c=a*b;&#10;java.lang.System.out.println(&quot;The result is c=&quot;+c);"/>
```

This script performs multiplication of variable "a" and "b", and stores it in "c" (those variables are formal parameters of XPDL Application definition).