Enhydra Shark

Table of Contents

What is Enhydra Shark?

Enhydra Shark is Java workflow engine completely based on WfMC [http://www.wfmc.org/standards/docs/if2v20.pdf] and OMG [http://www.omg.org/docs/formal/00-05-02.pdf] specifications.

- Shark is using WfMC's XML Process Definition Language [http://www.wfmc.org/standards/docs/TC-1025_10_xpdl_102502.pdf] (XPDL) as its native workflow definition format.

- Shark is a POJO library which provides APIs based on WfMC and OMG spec as well as a lot of additional Shark specific APIs for easier and more powerful workflow handling

  Since Shark is a library, it does not open its own threads, but everything works from client application thread, which makes shark a kind of workflow state machine - a thin layer on top of the database.

  This enables Shark to be used in many different environments. Basically shark can be used either directly through its POJO interface by integrating engine within WEB, Swing or pure console application, or it can be used as CORBA, EJB, RMI or WEB Service by making CORBA/EJB/RMI/WEB Service wrappers on top of the POJO interface.

  Shark project currently provides partial CORBA wrappers, full EJB wrappers and WEB Service wrappers based on stateless EJB interface and AXIS based WEB Service wrappers deployable on Tomcat. There are also several client applications (including administrative application) in Shark project which are able to access Shark through POJO interface, as well as through CORBA, EJB and WEB Service wrapper interfaces.

- Shark is very configurable, and all of its "internal" plug-in interfaces, as well as complete kernel could be replaced by another implementation.

- Shark library can be used from many VMs simultaneously (in cluster scenario).

- Shark can be configured to use organizational structure defined on LDAP server (through the use of specific implementation of shark's UserGroup plug-in component)

- Shark does not use any XPDL's Extended Attributes for its execution rules.

- Shark has full JTA support

- Shark uses DODS (OR/M tool from Enhydra), which enables shark to use almost any DB system for storing information, and it can be easily configured to switch target DB vendor and/or url (it has predefined scripts, and means to automatically create appropriate tables in those DBs using Octopus - ETL tool from Enhydra)

- Shark has implemented ToolAgent concept defined by WfMC to execute tools of automatic, server-side activities of XPDL definition. Several useful ToolAgents are coming with Shark, and anybody can create its own tool agents based on ToolAgent API, which provides enormous capabilities for integration with other systems.

- Shark can use custom Java classes (and even interfaces or abstract classes) as process variables.

Getting started

This section describes how to start with Enhydra Shark: where to download it, how to configure it and how to test it.

Installing a Binary Distribution

Download Binary Distribution

You can download the most recent community version of Shark's binary distribution from OW2. [http://forge.objectweb.org/project/showfiles.php?group_id=74] There you can choose between various distribution types (zip, tar.gz, rpm, exe). Release notes are also available at same location.

If you want to see advantages of professional version, the demo version is available at Together site [http://www.together.at/together/prod/tws/twsdemo/index.html] . You can see about additional features of professional version here. [http://www.together.at/together/prod/tws/twsfeatures/index.html]

Installing Shark from a Binary Distribution

If you are installing *exe* or *rpm* distribution just follow the usual installation procedure. If you are installing *tar.gz* or *zip* distribution, after unpacking it to a convinient location on your disk that we will refer to as SHARK_HOME, you should do the following steps:

- open configure.properties file from SHARK_HOME with your favorite text editor
- find the following section:

```
# HypersonicSQL
hsql_JdbcDriver=org.hsqldb.jdbcDriver
hsql_Connection_Url=jdbc:hsqldb:C:/sasaboy/tmp/Shark/output/tws/db/hsql/hsql
hsql_user=sa
hsql_passwd=
```

- replace the value of *hsql_Connection_Url* property with the location to the example hsql database that will be created, to correspond to the location of your shark installation. E.g. in the example above, you should replace the part:

  C:/sasaboy/tmp/Shark/output/tws

  with SHARK_HOME. If your SHARK_HOME is e.g. *D:/tws-community-2.0-3* you will have *hsql_Connection_Url* property defined as follows:

  ```
  hsql_Connection_Url=jdbc:hsqldb:D:/tws-community-2.0-3/db/hsql/hsql
  ```

  NOTE: be sure to use slashe characters when specifying this location.

- execute configure script from SHARK_HOME (configure.sh for unix or configure.bat for windows)

Now you will have the following directory structure:

Table 1. Shark directory structure

Directory	Description
SHARK_HOME	The root directory, referred as SHARK_HOME
..... .dist	d
..... bin	Executable scripts
..... conf	Configuration directory
..... dods	DODS configuration for various database vendors
..... sql	SQL scripts for creating shark database table structure for various database vendors
..... db	Directory for sample HSQL database
..... doc	Documentation
..... EJB	Contains EAR file for deployment in specific EJB container
..... JSPClient	Contains WAR file with sample JSP worklisthandler application
..... lib	Runtime libraries and third party dependencies
..... client	Client application libraries
..... clientcontrib	Third party libraries used in client applications
..... contrib	Third party libraries for engine
..... engine	Engine libraries
..... wrapper	CORBA and WfXML wrapper libraries
..... Licenses	Third party library's licenses
..... logs	Directory for execution logs
..... repository	XPDL Repository
..... external	Holds sample XPDL files
..... SharkWebClient	Contains WAR file (and zip file) with Shark WEB Client application for Tomcat
..... twe	XPDL Editor (Together Workflow Editor/JaWE)
..... WS-Plain	Contains WAR file for Shark Plain Web Service deployment

Building from Sources

To build Shark you need to use anonymous CVS to synchronize with the OW2 source repository at :pserver:anonymous@cvs.forge.objectweb.org:/cvsroot/shark.

Synchronizing with CVS

Follow these instructions [http://forge.objectweb.org/scm/?group_id=74] for using the Shark OW2 anonymous CVS repository.

If you didn't specify 'Shark' for the modulename, you should get both Shark and SharkWebClient sources.

Configuring Build Environment

Open console window and go to the root folder of Shark's CVS sources.

To configure build environment execute configure script (*configure.sh* on unix and *configure.bat* on windows) from Shark sources root folder. This will create *build.properties* files in the root folder and in the *util/dods* folder containing information neccessary to build shark.

The most important property there is *jdk_dir* which specifies which Java compiler will be used. If you want to change Java to be the different than the one registered with your system, you can execute configure script with additional options specified:

configure -jdkhome %JAVA_HOME%

where JAVA_HOME is the path to your Java installation.

You can also manually edit two files specified above to change *jdk_dir* property.

If you leave default settings, shark will be built without Web service support (which is much faster). If you need to build classes necessary to deploy shark as web service, you should either manually set the value of *build_ejb_ws* property from build.properties file to *on*, or execute:

configure -buildejbws on

This will enable you to generate EARs for different application servers which will have the ability to expose EJBs as web services as well as to access these web services through client applications coming with shark.

Compiling sources

Open console window and go to the root folder of Shark's CVS sources.

To compile sources, execute make script without any parameters. If you don't want documentation to be generated in the output use *make buildNoDoc*. Building without documentation is much faster procedure.

When make process finishes, you will get shark binaries in *output/tws* folder which will have the same structure as described for the binary distribution.

Note

After you build Shark, and if you want to "move" your binaries to the new location you need to execute:

configure -instdir %INST_DIR%

where INST_DIR is the place where you want to put shark binaries, and then execute:

make install

In the route of the project, there are eclipse project files that can help you to configure Shark project in eclipse.

Note

Not all the source files are included in Shark project. This is because some of them are being generated during "make" procedure (e.g. WfXML stubs, CORBA stubs, DODS layer objects, ...).

That's the reason why you should execute configure/make commands before being able to make valid project in eclipse or some other IDE.

Supported platforms

Operating systems

Shark can theoretically run on any operating system that supports Java 2, although it only comes with launch scripts for Windows and Unix/Linux. Shark is known to work with the following:

- Windows 2000, XP
- Linux
- AIX

J2SE Platform

Shark is tested to work with JDK 1.4 and later.

Note

Actually, core engine could probably be easily set up to work even with JDK1.3.1

Application Servers

The Shark can be adapted to run on any J2EE application server. It is currently known to work with the following application servers:

- Tomcat 5.5.x
- JBoss 4.x
- JOnAS Tomcat (4.5.2 and 4.7.5)
- Geronimo Tomcat 1.1.1
- WebLogic 8.x

Databases

When using DODS as implementation of persistence APIs, shark can work with different databases - practically, any database supported by DODS can be used.

Shark is known to work with the following databases:

- DB2
- Hypersonic
- MSQL
- MySQL
- Oracle
- PostgreSQL

The default database coming with Shark distribution is Hypersonic.

Starting Shark

Shark can be started from a client application by configuring it first (which can be done in several different manners), and then by getting an instance of it. This is the most common way to use shark from an application:

```
String confFilePath="c:/Shark.conf";
Shark.configure(confFilePath);
Shark shark=Shark.getInstance();
```

Everything else can be done through the Shark interface.

Before configuring shark, it must be ensured there is a Data source and TransactionManager accessible to Shark through JNDI.

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 configuring it with Shark.conf, you need to execute the following command:

```
LocalContextFactory.setup("sharkdb");
```

where there must be "sharkdb.properties" file in the class path, and this file should hold your datasource definition, i.e. something like:

```
jdbc.wrapper=org.enhydra.jdbc.standard.StandardXADataSource
jdbc.minconpool=12
jdbc.maxconpool=180
jdbc.connmaxage=30
jdbc.conncheclevel=1
datasource.description=Shark WfEngine DataSource
jdbc.conntesteststmt=SELECT 1
datasource.name=sharkdb

datasource.classname=org.hsqldb.jdbcDriver
datasource.url=jdbc:hsqldb:C:/sasaboy/prozonecvs/Shark/output/tws/db/hsql/hsql
datasource.username=sa
```

```
datasource.password=

datasource.isolationlevel=0
```

In the example above, you can see that datasource name is "sharkdb", so on the other side, shark must get the information for DODS how to search the datasource in JNDI, and there is a DODS property for this purpose that is called:

DatabaseManager.DB.sharkdb.Connection.DataSourceName and the default value for this property is

"jndi:java:comp/datasource/sharkdb" This value is appropriate for DODS to search for data source which name is "sharkdb" when we use LocalContextFactory, so we do not need to re-define it in Shark.conf in this case. During shark execution, both Shark kernel and DODS need access to TransactionManager which they are looking for through JNDI. There are also two default properties for Shark kernel and for DODS which are defining the lookup names for TransactionManager, and the default values are set to be adequate for Shark usage in a stand-alone application using LocalContextFactory. These properties are respectively:

SharkTxSynchronizationFactory.XATransactionManagerLookupName

and

DatabaseManager.defaults.XATransactionManagerLookupName and they both have the same default value:

javax.transaction.TransactionManager (NOTE: when we use Shark in some container like Tomcat or JBoss, we need to change the properties mentioned above according to container specification). Finally, the client application must know how to obtain UserTransaction from JNDI so it can perform begin/commit/rollback of the transaction. When using shark outside any container and with LocalContextFactory as described above, the UserTransaction lookup name is:

java:comp/UserTransaction So, the right procedure for starting stand-alone shark application could be:

```
    LocalContextFactory.setup("sharkdb");
    UserTransaction ut = null;
    try {
       ut = (UserTransaction) new InitialContext().lookup("java:comp/UserTransaction");
       ut.setTransactionTimeout(15 * 60);
       ut.begin();

       String confFilePath="c:/Shark.conf";
       Shark.configure(confFilePath);
       Shark shark=Shark.getInstance();

       ut.commit();

    } catch (Throwable ex) {
        throw new Error("Something really bad happened",ex);
    }
```

Configuring Shark

There are five different ways to configure shark:

1. use configure () method without parameters:

   then shark is configured only from config file that is placed in its jar file. Shark that is configured in this way works with default settings, and without many internal API implementations (Caching, EventAudit, Logging, ...).

NOTE: this way of default configuration is possible only when shark database is not HSQL, and only when data source lookup name equals to "jndi:java:comp/datasource/sharkdb" and TransactionManager lookup name equals to "javax.transaction.TransactionManager"

2. use configure (String filePath) method:

   it creates File object out of the path given in the filePath string, and calls the configure (File configFile) described next.

3. use configure (File configFile) method:

   shark first does basic configuration based on properties given in its jar file, and then does additional configuration from the file specified. If the configuration File defines same properties as in default configuration file from the jar, these property's values will override the default ones, plus all additional properties from File/Properties will be added to shark configuration. The configuration files you are passing as a parameter actually does not need to define whole configuration, but they could just redefine some default configuration parameters (i.e. how to handle otherwise transition, to re-evaluate deadlines or not, to create default assignment, ...) and add some additional configuration parameters (i.e. AssignmentManagerClassName).

4. use configure (Properties props) method:

   it does basically the same as previous method (in fact, the previous method converts the file content into Properties object), but it offers the possibility for client applications to use Java Properties object to configure shark.

5. use configure (Config config) method:

   this configuration through EAF's Config object makes possible to configure shark with properties defined in web.xml of your WEB application.

You can use many shark instances configured differently (you just need to specify different config files/ paths, or define different Property object). If you want to use several shark instances (from more than one VM) on the same DB, you should ALWAYS set the values for DODS cache sizes (must set it to zero), and CacheManagerClassName property should not exist).

As already mentioned, shark is very configurable engine, and all of its components, including kernel, can be replaced by a custom implementation.

The most common way for configuring shark is defining custom Shark.conf file, and here we will describe how you can configure shark, by briefly explaining the meaning of entries in standard Shark.conf file coming with shark distribution:

NOTE: Since Shark is singleton, it is currently not possible to use more then one shark instance in the same class loader.

Setting "enginename" parameter

You can set the name of shark instance by editing enginename property. Here is a part of configuration file for setting this property:

```
######################### NAME
# the name of shark instance
enginename=Shark
```

Can be used to identify shark instance (NOTE: in shark versions before 2.0 this parameter had also other meaning, and it was required to have different name for each shark instance).

Setting kernel behaviour in the case of unsatisfied split conditions

You can set the way how the standard shark kernel will react when the process has nowhere to go after an activity is finished, and all activity's outgoing transitions are unsatisfied (evaluated to false). Of course, this parameter has meaning only for the activities that have at least one outgoing transition.

Here is a part of configuration file for setting this property:

```
######################### KERNEL SETTING for UNSATISFIED SPLIT CONDITIONS
# There can be a cases when some activity that has outgoing transitions other
# than to itself (other then circular one), has nowhere to go based on
# calculation of these conditions (all of the conditions are evaluated to false)
# In that case, the process could hang (it will not go anywhere, and it will
# also not finish), finish (if there is no other active activities), or
# the last transaction that finishes the activity will be rolled back.
# This settings apply to the block activity's activities also, but the difference
# is that if you set parameter to FINISH_IF_POSSIBLE, shark will actually
# finish block activity if possible.
# The possible values for the entry are IGNORE, FINISH_IF_POSSIBLE and ROLLBACK,
# and default kernel behaviour is FINISH_IF_POSSIBLE
#SharkKernel.UnsatisfiedSplitConditionsHandling=FINISH_IF_POSSIBLE
```

So, there are three possible solutions as described, and the default one is to finish the process if possible.

Setting kernel to evaluate OTHERWISE conditions last

XPDL spec does not say that *OTHERWISE* transition should be executed only if no other transition condition is evaluated to true (in the case of XOR split). So, if you i.e. put *OTHERWISE* transition to be the first outgoing transition of some activity, other transition's condition won't be even considered.

You can configure shark to deviate from the spec, so that *OTHERWISE* transition is evaluated and executed only if no other transition condition is evaluated to true. To do that, you should set the following property to true.

```
SharkKernel.handleOtherwiseTransitionLast=false
```

This parameter could be saving lot of headaches to XPDL designers, by removing the extra care on *OTHERWISE* transition positioning.

Setting kernel for assignment creation

Determines if kernel will create assignments - default is true. There are situations when assignment creation is not necessary, and this is the case when all the processes are such that the whole process belongs to a user which created it.

```
SharkKernel.createAssignments=true
```

Since this setting affects the complete engine, you should carefully consider if this is your use case. In this case users won't have anything in their worklists, and client application should provide a way to bind user with its process.

Setting kernel for default assignment creation

Determines if kernel will create default assignment for the process creator if assignment manager return zero assignments.

NOTE: if this property is set to true, there can be side-effect with Tool activities with Manual Start and Finish mode.

```
SharkKernel.createDefaultAssignment=true
```

Default kernel value is true.

Setting kernel for resource handling during assignment creation

Defines the limit number for loading all WfResources from DB before creating assignments.

When kernel determines that more assignments than the number specified by the limit should be created it will make a call to retrieve all WfResources from DB.

When DODS is used as a persistence layer, it can improve the performance if there are not too many WfResource objects in the system:

```
SharkKernel.LimitForRetrievingAllResourcesWhenCreatingAssignments=5
```

Default kernel value is 5.

Setting kernel behaviour to re-evaluate assignments at engine startup

It is possible to force kernel to re-evaluate assignments during shark initialization. This can be done by changing the following property:

```
#Assignments.InitialReevaluation=false
```

If you set this property to true, all not-accepted assignments are going to be re-evaluated (old ones will be deleted, and new ones will be created based on current mappings, current state of User/Group information and current implementation of AssignmentManager class).

Default kernel setting is not to re-evaluate assignments.

Setting kernel for assignment handling

Determines if kernel will delete other assignments from DB everytime when someone accepts/rejects assignment, and will re-evaluate assignments each time this happens. If it is set to true, the side-effect is that if there was reassignment, and the user that got this reassigned assignment rejects it, he will not get it afterwards.

```
SharkKernel.deleteOtherAssignments=true
```

The shark kernel default is true.

Setting kernel behaviour to fill the caches on startup

If you want shark to fill its Process and Resource caches at startup, you should edit the following entries from configuration file:

```
#Cache.InitProcessCacheString=*
#Cache.InitResourceCacheString=*
```

If you uncomment these lines, all processes and resources will be created based on DB data, and will be filled into cache (actually, this number is restricted by the cache size).

The value of these properties can be set as a comma separated list of the process/resource ids that need to be put into cache on engine start, e.g.: Cache.InitProcessCacheString=1_test_js_basic, 5_test_js_Game

Shark kernel default is not to initialize caches.

Setting kernel behaviour for reevaluating deadline limits

If you want shark not to reevaluate deadlines each time external deadline management checks for deadlines, you should set following entry to false (default kernel setting is true)

```
#Deadlines.reevaluateDeadlines=true
```

Determines if process or activity context will be used when re-evaluating deadlines Default kernel setting is activity context.

```
Deadlines.useProcessContext=false
```

Determines if asynchronous deadline should be raised only once, or every time when deadline check is performed. Default kernel setting is true (to raise deadline only once).

```
Deadlines.raiseAsyncDeadlineOnlyOnce=true
```

Setting kernel and event audit mgr for persisting old event audit data

Determines if old event audit data should be persisted or not. Default is to persist. The value of this property must be respected by both, the kernel, and event audit manager.

```
PERSIST_OLD_EVENT_AUDIT_DATA=true
```

Default kernel setting is true.

Setting kernel for the priority handling

Determines if it is allowed to set the priority of the WfProcess/WfActivity out of the range [1-5] as defined by OMG spec:

```
#SharkKernel.allowOutOfRangePriority=false
```

Default kernel setting is false.

Setting properties for browsing LDAP server (only available in professional version)

If you are using a LDAP server to hold your organization structure, you can configure shark to use our LDAP implementation of UserGroup and Authentication interface (it will be explained later in the text how to set it up), and then you MUST define some LDAP properties.

At the moment, shark implementations of UserGroup interfaces support two types of LDAP structures. The first structure is marked as type 0, and the second is marked as type 1. The LDAP structures are detailly explained in the document LDAP structures in Shark (html [../ldap_structure/LDAP_structure.html], pdf [../ldap_structure/LDAP_structure.pdf])

You can set this properties based on your LDAP server configuration, by changing the following part of configuration file:

```
# Shark can use LDAP implementation of UserGroup interfaces,
# and these are settings required by this implementations to access and
# browse the LDAP server

# possible values for LDAPStructureType parameter are 0,1 and 2
# 0 is simple structure, the possibility that one group or user belongs to more
# than one group is not supported
# 1 is more complex structure that supports the possibility that one group or
# user belongs to more than one group
# 2 Active Directory server (default) structure
LDAPStructureType=2

LDAPHost=localhost
LDAPPort=389

LDAPSearchBase=cn=Users,dc=prozone,dc=co,dc=yu
LDAPGroupObjectClasses=group
LDAPUserObjectClasses=person

LDAPGroupUniqueAttributeName=sAMAccountName
LDAPUserUniqueAttributeName=sAMAccountName

LDAPGroupDescriptionAttributeName=description

LDAPUserPasswordAttributeName=userPassword
LDAPUserRealNameAttributeName=displayName
LDAPUserFirstNameAttributeName=givenName
LDAPUserLastNameAttributeName=sn
LDAPUserEmailAttributeName=mail

LDAPUser=sasaboy@prozone.co.yu
LDAPPassword=

# Specifies the size of LRU cache for holding user attributes (for shark performance reason)
LDAPClient.userAttributesCacheSize=100

# Specifies the size of LRU cache for holding group attributes (for shark performance reason)
LDAPClient.groupAttributesCacheSize=100

# Active Directory specifics (when LDAPStructureType is set to 2)
----------------------------------------------------------------
# holds information about the member that belongs to (group) entry
```

```
LDAPMemberAttributeName=member

# holds information about the membership of entity
LDAPMemberOfAttributeName=memberOf

# Unique representation of entry
LDAPDistinguishedNameAttributeName=distinguishedName


# specifics for LDAPStructureType=1
----------------------------------
LDAPRelationObjectClasses=groupOfNames
LDAPRelationUniqueAttributeName=cn
LDAPRelationMemberAttributeName=member
LDAPGroupGroupsName=Groups
LDAPGroupUsersName=Users
LDAPGroupGroupRelationsName=GroupRelations
LDAPGroupUserRelationsName=UserRelations
```

- LDAPHost - the address of the machine where LDAP server is running
- LDAPPort - the port through which LDAP server can be accessed
- LDAPStructureType - if set to 0, the simple structure is used in which the possibility that one group or user belongs to more than one group is not supported, if set to 1, the more complex structure is used which supports the possibility that one group or user belongs to more than one group is not supported. If set to 2, it is configured to access standard ActiveDirectory structure.
- LDAPSearchBase - the name of the context or object to search (this is the root LDAP node where all queries will start at).
- LDAPGroupObjectClasses - the comma separated list of LDAP object classes representing Group of users. It is important that these classes must have a mandatory attribute whose value uniquely identifies each entry throughout the LDAP tree.
- LDAPUserObjectClasses - the comma separated list of LDAP object classes representing shark users. It is important that these classes must have a mandatory attribute whose value uniquely identifies each entry throughout the LDAP tree.
- LDAPGroupUniqueAttributeName - the name of attribute that is mandatory for each LDAP object class representing Group of users. The value of this attribute MUST be unique for each LDAP entry for these object classes throught the LDAP tree.
- LDAPGroupDescriptionAttributeName - the name of attribute of LDAP object classes representing Group of users that represents the Group description.
- LDAPUserUniqueAttributeName - the name of attribute that is mandatory for each LDAP object class representing User. The value of this attribute MUST be unique for each LDAP entry for these object classes throughout the LDAP tree. When shark uses LDAP for authentication and user group management, this attribute represents the username for logging into shark.
- LDAPUserPasswordAttributeName - the name of attribute that is mandatory for each LDAP object class representing User. When shark uses LDAP for authentication and user group management, this attribute represents the password needed for logging into shark.
- LDAPUserRealNameAttributeName - the name of the attribute of LDAP object classes representing User, that represents the real name of the shark user.
- LDAPUserFirstNameAttributeName - the name of the attribute of LDAP object classes representing User, that represents the first name of the shark user.

- LDAPUserLastNameAttributeName - the name of the attribute of LDAP object classes representing User, that represents the last name of the shark user.
- LDAPUserEmailAttributeName - the name of the attribute of LDAP object classes representing User, that represents user's email address.
- LDAPUser - when LDAP server requires credentials for reading, this is the username that will be used when connecting LDAP server
- LDAPPassword - when LDAP server requires credentials for reading, this is the password that will be used when connecting LDAP server
- LDAPMemberAttributeName - only used in structure type 2 (Active Directory). Holds information about the member that belongs to (group) entry.
- LDAPMemberOfAttributeName - only used in structure type 2 (Active Directory). Holds information about the membership of entity.
- LDAPDistinguishedNameAttributeName - only used in structure type 2 (Active Directory). Unique representation of entry.
- LDAPRelationObjectClasses - only used in structure type 1, the comma separated list of LDAP object classes representing relations between shark users and group or between shark groups. It is important that these classes must have a mandatory attribute whose value uniquely identifies each entry throughout the LDAP tree.
- LDAPRelationUniqueAttributeName - only used in structure type 1, the name of attribute that is mandatory for each LDAP object class representing Relation of groups or group and users. The value of this attribute MUST be unique for each LDAP entry for these object classes throught the LDAP tree
- LDAPRelationMemberAttributeName - only used in structure type 1,the name of attribute of LDAP object classes (representing Relation of groups or group and users) that represents member that is included (user or group) in the relation.
- LDAPGroupGroupsName - only used in structure type 1, the name of the specific group that must be created and which will contain all groups
- LDAPGroupUsersName - only used in structure type 1, the name of the specific group that must be created and which will contain all users
- LDAPGroupGroupRelationsName - only used in structure type 1, the name of the specific group that must be created and which will contain all relations between groups
- LDAPGroupUserRelationsName - only used in structure type 1, the name of the specific group that must be created and which will contain all relations between groups and users

Setting kernel's CallbackUtilities implementation class

If one wants to give its own implementation of CallbackUtilities interface, he can do it by changing the following attribute:

```
######################### CALLBACK UTILITIES
# used for logging, and getting the shark properties
# the default kernel setting is as follows
#CallbackUtilitiesClassName=org.enhydra.shark.CallbackUtil
CallbackUtil.TimeProfiler.default=120
```

```
CallbackUtil.TimeProfiler.level=info
```

The name of the class that is used by default is commented.

This interface implementation is passed to all internal interface implementations, and is used by those implementations to read shark property values, to log events and to utilize profiling options.

Property *CallbackUtil.TimeProfiler.default* specifes the value in milliseconds for profiling log. If some shark API method takes more time to execute than the value specified, it will be logged. If property *CallbackUtil.TimeProfiler.level* is set to *debug* the whole stack-trace is logged, otherwise the normal information about which method took too long is logged.

Setting kernel's ObjectFactory implementation class

If one wants to replace some parts of kernel with its own implementation (i.e. to replace WfActivityInternal, WfProcessInternal, ... implementations), he should create its own class based on this interface, and configure shark to use it.

This can be done by changing the following part of configuration file:

```
######################### OBJECT FACTORY
# the class name of the factory used to creating kernel objects
# the default kernel setting is as follows
#ObjectFactoryClassName=org.enhydra.shark.SharkObjectFactory
```

The name of the class that is used by default is commented.

Setting kernel's ToolActivityHandler implementation class

If one wants to set its own ToolActivityHandler implementation, that will communicate with tool agents in a different way than the standard implementation does, he can configure the following:

```
######################### TOOL ACTIVITY HANDLER
# the class name of the manager used to execute tool agents
# the default kernel setting is as follows
#ToolActivityHandlerClassName=org.enhydra.shark.StandardToolActivityHandler
```

The name of the class that is used by default is commented.

Setting kernel's TxSynchronizationFactory class

Implementation of TxSynchronizationFactory interface is responsible to support shark to work in JTA environment.

```
######################### Tx SYNCHRONIZATION FACTORY
#TxSynchronizationFactoryClassName=org.enhydra.shark.SharkTxSynchronizationFactory
#SharkTxSynchronizationFactory.XATransactionManagerLookupName=javax.transaction.TransactionManager
#SharkTxSynchronizationFactory.debug=false
```

Default factory is org.enhydra.shark.SharkTxSynchronizationFactory.

It is important to configure the parameter *SharkTxSynchronizationFactory.XATransactionManagerLookupName* to specify JNDI lookup name of the TransactionManager.

Database configuration

This section of configuration file is related to DODS implementation of persisting APIs.

In shark distribution, we provide SQL scripts for creating tables for the most DBs supported by DODS, and appropriate LoaderJob files that can be used by Octopus to create DB tables if providing appropriate drivers. This files can be found in conf/sql folder.

```
#
# Turn on/off debugging for transactions or queries. Valid values
# are "true" or "false".
#
DatabaseManager.Debug="false"

# Special settings for Postgresql DB
#DatabaseManager.ObjectIdColumnName=ObjectId
#DatabaseManager.VersionColumnName=ObjectVersion

#
# Maximum amount of time that a thread will wait for
# a connection from the connection pool before an
# exception is thrown.  This will prevent possible dead
# locks.  The time out is in milliseconds.  If the
# time out is <= zero, the allocation of connections
# will wait indefinitely.
#
#DatabaseManager.DB.sharkdb.Connection.AllocationTimeout=10000

#
# Required for HSQL: column name NEXT must be used
# with table name prefix
# NOTE: When working with other DBs, you should comment these two properties
#
DatabaseManager.DB.sharkdb.ObjectId.NextWithPrefix = true
DatabaseManager.DB.sharkdb.Connection.ShutDownString = SHUTDOWN

#
# Used to log database (SQL) activity.
#
DatabaseManager.DB.sharkdb.Connection.Logging=false
```

There is another important DODS configuration aspect - the cache sizes:

```
#
# Default cache configuration
#
DatabaseManager.defaults.cache.maxCacheSize=100
DatabaseManager.defaults.cache.maxSimpleCacheSize=50
DatabaseManager.defaults.cache.maxComplexCacheSize=25
```

If you know that several instances of shark will be used in several VMs, using the same DB, you should set all this cache sizes to zero. Along with this, cache manager implementation (explained later in the text) should not be used.

Setting persitence components variable data model

Following options are described together, although they affect different components, because option's intention and the effect produced are the same.

Determines the maximum size of String that will be stored in VARCHAR field. String which size is greater than specified value will be stored as a BLOB. The maximumum size that can be set is 4000 (the default one)

```
DODSPersistentManager.maxVARCHARSize=4000
DODSEventAuditManager.maxVARCHARSize=4000
```

Determines which data model will be used for storing process and activity variables. There are two options:

1. using standard data model, where all data types are in one table (including BLOB data type for persisting custom Java objects and large Strings
2. using optional data model, where one table contains all data types except BLOB, and there is another table that references previous table, and is used only for storing BLOB information (for persisting custom Java objects and large Strings)

Default is to use standard data model, but using optional data model can improve performance in use cases where there are not so many custom Java objects and large String objects, and when shark and DODS caches are not used, and this is especially better choice if using Oracle DB.

```
DODSPersistentManager.useStandardVariableDataModel=true
DODSEventAuditManager.useStandardVariableDataModel=true
```

Setting Assignment manager implementation class

If one would like to create its own Assignment manager, which would decide which assignments are to be created for an activity, he can implement its own Assignment manager, based on AssignmentManager interface, and configure shark to use it by changing the following setting:

```
AssignmentManagerClassName=org.enhydra.shark.assignment.StandardAssignmentManager
```

Shark comes with three different implementations of this manager:

- Standard - just returns the list of users passed as a parameter, or if there are no users in the list, it returns the user that created corresponding process.
- History Related - if there are some special "Extended attributes" defined in XPDL for some activity definition, this implementation checks the assignment history (who has already executed activity with such definition, ...) to make a decission about assignments that should be created.
- XPDL Straight Participant Mapping - it makes assignments for the user that has the same Id as XPDL performer of activity.
- Workload Related (only professional version) - it makes assignments by taking into account user workload

NOTE: if you do not set any implementation (you simply comment line above), shark will use the default procedure. Actually, standard implementation of assignment API is not very useful, it basically just returns the first valid options.

Setting user group implementation

Shark's standard and history related assignment managers can be configured to use some implementation of UserGroup API when determining which user(s) should get the assignment.

Shark comes with DB based implementation of this API and professional version also brings LDAP implementation of this API . DB based implementation uses DB for retrieving information about

organizational structure, and LDAP based implementation uses LDAP server for getting organizational information.

Here is a part of configuration file for setting UserGroup manager implementation for standard assignment manager:

```
StandardAssignmentManager.UserGroupManagerClassName=org.enhydra.shark.usergroup.DODSUserGroupManager
```

NOTE: shark can work without implementation of this API - if you do not want to use any implementation, simply comment line above.

Setting participant map persistence implementation

Shark's standard and history related assignment managers can be configured to use some implementation of ParticipantMapping API when determining which user(s) should get the assignment.

This API is to retrieve mapping information between XPDL participants and shark users. Shark application comes with DODS based participant map persistence implementation.

You can provide your own implementation of participant map persistence API.

Here is a part of configuration file for setting ParticipantMapping manager implementation for standard assignment manager:

```
StandardAssignmentManager.ParticipantMapPersistenceManagerClassName=org.enhydra.shark.partmappersistence.DODSI
```

NOTE: if you comment the lines above, shark will work without participant map persistence API implementation.

Setting Caching implementation

Shark comes with LRU based cache implementation for holding Process and Resource objects. By default, shark is configured to use this cache implementation, which can speed-up its use by the clients.

This is the section of configuration file that defines cache implementation, and its sizes:

```
#==============================================================================
# Default cache is LRU
#
#------------------------------------------------------------------------------
# Cache defaults
#
CacheManagerClassName=org.enhydra.shark.caching.LRUCacheMgr

# Default LRU cache sizes (LRU implementation default is 100 for each cache)
#LRUProcessCache.Size=100
#LRUResourceCache.Size=100
```

NOTE: if you do not set any implementation (you simply comment line above), shark will not perform any caching.

Setting instance persistence implementation

The implementation of this API is used to store information about shark's processes, activities, ... into DB. Shark comes with DODS based instance persistence implementation. One can write its own

implementation of this interface (maybe using Hibernate or EJB), and to configure shark to work with this implementation, he needs to edit the following section of configuration file:

```
#
# DODS instance persistent manager defaults
#
InstancePersistenceManagerClassName=org.enhydra.shark.instancepersistence.DODSPersistentManager
```

Shark can't work without instance persistence implementation.

NOTE: If one would like to implement other instance persistence implementation, he should also give its own implementation of SharkTransaction API.

Configuring DODS instance persistence implementation to delete processes when they finish

By default, DODS implementation of instance persistence interface does not delete finished processes, but they are left in DB. This behaviour can be changed by setting the following parameter to true:

```
# Determines if finished processes should be deleted from DB (DODS persistence
# manager default is false)
#DODSPersistentManager.deleteFinishedProcesses=false
```

Setting logging API implementation

Shark comes with a default logger implementation, implemented by the use of log4j. You can write your own implementation of Logging API, and set it by editing configuration file, and probably adding some additional entries in configuration file that will be read by your logger implementation. Here is a complete logger configuration for shark standard logger:

```
#
# Standard logging manager defaults
#
LoggingManagerClassName=org.enhydra.shark.logging.StandardLoggingManager


# Standard Logging manager is using log4j, and here is log4j configuration
#
log4j.rootLogger=info, SharkExecution

log4j.appender.Database=org.apache.log4j.RollingFileAppender
log4j.appender.Database.File=@WD_PATH@/logs/SharkPersistence.log
log4j.appender.Database.MaxFileSize=10MB
log4j.appender.Database.MaxBackupIndex=2
log4j.appender.Database.layout=org.apache.log4j.PatternLayout
log4j.appender.Database.layout.ConversionPattern=%d{ISO8601}: %m%n

log4j.appender.XMLOutFormatForPersistence=org.apache.log4j.FileAppender
log4j.appender.XMLOutFormatForPersistence.File=@WD_PATH@/logs/chainsaw-persistence.log
log4j.appender.XMLOutFormatForPersistence.append=false
log4j.appender.XMLOutFormatForPersistence.layout=org.apache.log4j.xml.XMLLayout

log4j.appender.PackageEvents=org.apache.log4j.RollingFileAppender
log4j.appender.PackageEvents.File=@WD_PATH@/logs/SharkPackageHandlingEvents.log
log4j.appender.PackageEvents.MaxFileSize=10MB
log4j.appender.PackageEvents.MaxBackupIndex=2
log4j.appender.PackageEvents.layout=org.apache.log4j.PatternLayout
log4j.appender.PackageEvents.layout.ConversionPattern=%d{ISO8601}: %m%n

log4j.appender.DatabaseManager=org.apache.log4j.RollingFileAppender
log4j.appender.DatabaseManager.File=@WD_PATH@/logs/dods.log
```

```
log4j.appender.DatabaseManager.MaxFileSize=10MB
log4j.appender.DatabaseManager.MaxBackupIndex=2
log4j.appender.DatabaseManager.layout=org.apache.log4j.PatternLayout
log4j.appender.DatabaseManager.layout.ConversionPattern=%d{ISO8601}: %m%n

log4j.appender.XMLOutFormatForPackageEvents=org.apache.log4j.FileAppender
log4j.appender.XMLOutFormatForPackageEvents.File=@WD_PATH@/logs/chainsaw-packageevents.log
log4j.appender.XMLOutFormatForPackageEvents.append=false
log4j.appender.XMLOutFormatForPackageEvents.layout=org.apache.log4j.xml.XMLLayout

log4j.appender.SharkExecution=org.apache.log4j.RollingFileAppender
log4j.appender.SharkExecution.File=@WD_PATH@/logs/SharkExecutionFlow.log
log4j.appender.SharkExecution.MaxFileSize=10MB
log4j.appender.SharkExecution.MaxBackupIndex=2
log4j.appender.SharkExecution.layout=org.apache.log4j.PatternLayout
log4j.appender.SharkExecution.layout.ConversionPattern=%d{ISO8601}: %m%n

log4j.appender.XMLOutFormatForExecution=org.apache.log4j.FileAppender
log4j.appender.XMLOutFormatForExecution.File=@WD_PATH@/logs/chainsaw-execution.log
log4j.appender.XMLOutFormatForExecution.append=false
log4j.appender.XMLOutFormatForExecution.layout=org.apache.log4j.xml.XMLLayout

log4j.appender.NTEventLog=org.apache.log4j.nt.NTEventLogAppender
log4j.appender.NTEventLog.source=SharkCORBA-Service
log4j.appender.NTEventLog.layout=org.apache.log4j.PatternLayout
log4j.appender.NTEventLog.layout.ConversionPattern="%d{ISO8601}: [%t], %p, %c: %m%n"

log4j.appender.TP=org.apache.log4j.RollingFileAppender
log4j.appender.TP.File=@WD_PATH@/logs/tp.log
log4j.appender.TP.MaxFileSize=10MB
log4j.appender.TP.MaxBackupIndex=2
log4j.appender.TP.layout=org.apache.log4j.PatternLayout
log4j.appender.TP.layout.ConversionPattern=%d{ISO8601}: [%t], %p, %c: %m%n

log4j.appender.TP-IP=org.apache.log4j.RollingFileAppender
log4j.appender.TP-IP.File=@WD_PATH@/logs/tp-ip.log
log4j.appender.TP-IP.MaxFileSize=10MB
log4j.appender.TP-IP.MaxBackupIndex=2
log4j.appender.TP-IP.layout=org.apache.log4j.PatternLayout
log4j.appender.TP-IP.layout.ConversionPattern=%d{ISO8601}: [%t], %p, %c: %m%n

log4j.appender.Console=org.apache.log4j.ConsoleAppender
log4j.appender.Console.layout=org.apache.log4j.PatternLayout
log4j.appender.Console.layout.ConversionPattern=%d{ISO8601}: %m%n

log4j.logger.Persistence=INFO,Database
#log4j.logger.Persistence=INFO,Database,XMLOutFormatForPersistence

log4j.logger.PackageEventLogger=INFO,PackageEvents
#log4j.logger.PackageEventLogger=INFO,PackageEvents,XMLOutFormatForPackageEvents

log4j.logger.TimeProfiler=INFO,Console,TP
log4j.logger.TimeProfiler-InstancePersistence=INFO,Console,TP-IP

log4j.logger.Shark=INFO,@MAIN_LOG_CHANNEL@,SharkExecution
#log4j.logger.Shark=INFO,Console,SharkExecution,XMLOutFormatForExecution

log4j.logger.Scripting=INFO,Console,SharkExecution
#log4j.logger.Scripting=INFO,SharkExecution,XMLOutFormatForExecution

log4j.logger.DatabaseManager=INFO,DatabaseManager
```

The standard logger implementation is written in a way that it could log even if there are no log4j settings defined in configuration file (so the implementation can't configure log4j), but log4j is configured from client application using shark.

The following log outputs are generated by default:

- Server execution flow log - logs every significant shark operation like package loading, process instantiation, activity completion, ... These logs are also displayed in the console during shark execution.

- Package Handling Events - logs every operation performed with Package definition files (XPDL files). These operations are:
  - loading of the package from external repository into shark's memory
  - unloading of the package from the shark
  - updating of the package that is already in the shark's memory
- Server persistence log - logs every operation related to communication among DODS instance persistence implementation, and underlying database.

You have the possibility to force Shark to make log files that can be viewed using log4j's chainsaw viewer. To do so, for each type of logger, you have to comment first and uncomment the second line that refers to the logger at the bottom of logger configuration.

Then, the output logs will be also generated into XML log files (chainsaw-execution.log, chainsaw-packageevents.log and chainsaw-persistence.log) that can be read by chainsaw.

The chainsaw can be started by using proper "chainsaw" script from the root of the project. When it is started, you have to open wanted log file by using its "File->Load file..." menu item, and it will present you the proper logs.

NOTE: If you do not want any logging, comment LoggingManagerClassName line above, and shark will not log anywhere.

Setting repository persistence implementation

This API is used to store information about XPDL definitions and versions. Shark comes with two implementations of this API: FileSystem based, and DODS based.

You can provide your own implementation of this API, and replace the current implementation. The default implementation is DODS implementation.

```
# Default repository persistent manager is DODS
#

#RepositoryPersistenceManagerClassName=org.enhydra.shark.repositorypersistence.FileSystemRepositoryPersistence

# The location of xpdl repository.
# If you want to specify it by relative path, you must know that this path must
# be relative to the Shark.conf file (in conf folder)
FileSystemRepositoryPersistenceManager.XPDL_REPOSITORY=repository/internal

# The location of xpdl history repository.
# If you want to specify it by relative path, you must know that this path must
# be relative to the Shark.conf file (in conf folder)
FileSystemRepositoryPersistenceManager.XPDL_HISTORY_REPOSITORY=repository/internal/history


RepositoryPersistenceManagerClassName=org.enhydra.shark.repositorypersistence.DODSRepositoryPersistenceManager

# The database used for Repository persistence when using DODS implementaion
#DODSRepositoryPersistenceManager.DatabaseName=sharkdb

# If set to true, the debug information on repository transaction will be
# written to console
#DODSRepositoryPersistenceManager.debug=false
```

NOTE: Shark can't work without implementation of this API.

Setting scripting manager implementation

Shark comes with standard scripting manager implementation. This is a factory for returning appropriate script evaluator, and standard implementation offers three different script evaluators: Python, Java script and Bean shell.

```
#============================================================================
# Default Scripting manager is Standard
#
#----------------------------------------------------------------------------
#
ScriptingManagerClassName=org.enhydra.shark.scripting.StandardScriptingManager
```

Shark can't work without Scripting API implementation.

Setting security (authorization) API implementation

This API contains methods to authorize shark usage on the level of particular methods (i.e. user is authorized to create, abort, terminate or suspend some process, ...).

```
#============================================================================
# Default Security manager is Standard
#
#----------------------------------------------------------------------------
#
SecurityManagerClassName=org.enhydra.shark.security.StandardSecurityManager
```

NOTE: If you don't want any authorization, you just need to comment line above - shark can work without this API implementation.

Setting tool agents

Shark comes with standard ToolAgentFactory implementation, and with several example tool agents (JavaScript, BeanShell, RuntimeApplication, SOAP, Mail and JavaClass tool agent), and with default tool agent implementation.

To learn more about tool agent, you should look at ToolAgent documentation.

These are configuration settings for tool agents:

```
#============================================================================
# Default Tool agent settings
#
#----------------------------------------------------------------------------
#
ToolAgentManagerClassName=org.enhydra.shark.toolagent.StandardToolAgentManager

# The list of tool agents
ToolAgent.JavaClassToolAgent=org.enhydra.shark.toolagent.JavaClassToolAgent
ToolAgent.JavaScriptToolAgent=org.enhydra.shark.toolagent.JavaScriptToolAgent
ToolAgent.BshToolAgent=org.enhydra.shark.toolagent.BshToolAgent
ToolAgent.RuntimeApplicationToolAgent=org.enhydra.shark.toolagent.RuntimeApplicationToolAgent
ToolAgent.MailToolAgent=org.enhydra.shark.toolagent.MailToolAgent
ToolAgent.SOAPToolAgent=org.enhydra.shark.toolagent.SOAPToolAgent
ToolAgent.SchedulerToolAgent=org.enhydra.shark.toolagent.SchedulerToolAgent

# Pool size for Scheduler Tool Agent
SchedulerToolAgent.threadPoolSize=3

# Default tool agent is used when there is no mappings for some
```

```
# XPDL application definition
DefaultToolAgent=org.enhydra.shark.toolagent.DefaultToolAgent

# Specifies the size of cache for holding ext. attributes (for shark performance reason)
# Default -1 means unlimited
#AbstractToolAgent.extAttribsCacheSize=-1
```

NOTE: shark can work without tool agent API implementation, but then it can only execute processes that do not contain any "Tool" activity.

Setting application map persistence implementation

This API is used to retrieve mapping information between XPDL applications and tool agent applications. Shark comes with DODS based application map persistence implementation.

For a standard tool agent manager, you can specify which implementation of application map persistence API you want to use.

```
# Application map details for StandardToolAgentManager
StandardToolAgentManager.ApplicationMapPersistenceManagerClassName=org.enhydra.shark.appmappersistence.DODSAp
```

NOTE: shark can work without application map persistence API implementation.

Setting WfXML interoperability implementation

This API is used to communicate with other engines via WfXML protocol (spec defined by WfMC).

```
#==============================================================================
# WfEngineInterpoerability manager
#
#------------------------------------------------------------------------------
#
#WfEngineInteroperabilityManagerClassName=org.enhydra.shark.interoperability.WfXMLInteroperabilityImpl
#Interoperability.Host=localhost
#Interoperability.Port=8080
#Interoperability.ObserverPath=/axis/services/asapObserverBinding
#Interoperability.IgnoreTerminateAndAbortRemoteExceptions=false
```

NOTE: shark can work without implementation of this API.

Setting DODS Id generator cache size(s)

You can specify cache sizes for object Ids (activity and process Ids). When some process or activity is created, shark asks its data layer (default DODS layer) for unique Id. This Id generation is synchronized on DB, so that shark can be used from different VMs at a time. To tell shark not to go to the DB so often, you can specify an Id cache for objects:

```
#==============================================================================
# DODS Settings for Id Generator
#------------------------------------------------------------------------------
# default cache size for Ids (if cache size for particular object Id is not
# specified, then this size is used, and if this cache size also isn't
# specified, program default is used)
DODS.defaults.IdGenerator.CacheSize=100

# cache size for process instance Ids
#DODS.IdGenerator._process_.CacheSize=100
```

```
# cache size for activity instance Ids
#DODS.IdGenerator._activity_.CacheSize=100
```

About data model

You can find here DODS generated documentation of various data models used in default shark configuration:

Instance persistence data model - (html [../SharkInstancePersistence-DODS.html], pdf [../SharkInstancePersistence-DODS.pdf])

Event audit data model - (html [../SharkEventAudit-DODS.html], pdf [../SharkEventAudit-DODS.pdf])

Repository persistence data model - (html [../SharkRepositoryPersistence-DODS.html], pdf [../SharkRepositoryPersistence-DODS.pdf])

Participant map persistence data model - (html [../SharkParticipantMapPersistence-DODS.html], pdf [../SharkParticipantMapPersistence-DODS.pdf])

UserGroup persistence data model - (html [../SharkUserGroup-DODS.html], pdf [../SharkUserGroup-DODS.pdf])

Application map persistence data model - (html [../SharkApplicationMapPersistence-DODS.html], pdf [../SharkApplicationMapPersistence-DODS.pdf])

Id Counter data model - (html [../SharkUtilities-DODS.html],pdf [../SharkUtilities-DODS.pdf])

Database support

What Needs to be Configured in Order to Use Database Other Then HypersonicSQL

The scripts for creating tables for various databases (by using Octopus) are distributed with Shark. If you want to use different database then the one originally configured to work with Shark (HypersonicSQL database), you should do the following:

- 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_user`	username for database authentication
`db_passwd`	password for database authentication
`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.

- 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.