Nova Bonita documentation

BonitaTeam ()

- June 2008 -

Table of Contents

Introduction

This documentation is targetted for Bonita users. It presents the installation procedure and a quick user guide of Bonita v4 Release Candidate 2 features.

Chapter 1 , General information describes the new version Bonita v4 called Nova Bonita

Chapter 2 , Prerequisites describes the hardware and software prerequisites

Chapter 3, Installation guide describes how to install the Bonita v4 RC2

Chapter 4, Change history between Bonita v3 and Bonita v4

Chapter 5, Configuration and Services describes main configuration features and default services

Chapter 6, Concepts describes the main workflow concepts

Chapter 7, User Guide guides you through the descovery Nova Bonita functionalities.

Chapter 1. General information

1.1. Nova Bonita introduction

Nova Bonita is the name of new version of Bonita v4.

“Nova” technology is based on the “Process Virtual Machine” conceptual model for processes. The Process Virtual Machine defines a generic process engine enabling support for multiple process languages (such BPEL, XPDL…).

On top of that, it leads to a pluggable and embeddable design of process engines that gives modelling freedom to the business analyst. Additionally, it enables the developer to leverage process technology embedded in a Java application.

For more information about the Process Virtual Machine, check Nova Bonita FAQs [http://wiki.bonita.objectweb.org/xwiki/bin/view/Main/FAQ] on the Bonita web site [http://bonita.objectweb.org].

1.2. Feature list

Nova Bonita (aka Bonita v4) is a lightweight workflow/BPM solution that provide XPDL support. Nova Bonita Release Candidate 2 comes with an enhanced XPDL extension module, a reworked workflow API, support for iterations and deadlines and a set of new features and services such journal, history and timers services.

This RC2 version adds minor feature enhacements compared to the RC1 (typed hooks, activity body, getTaskList operation by userId or get/set local vs global variables operations). The main focus of this version has been QA: code cleaning, validation, stress tests, unit tests coverage and performance improvements. Together with that some new examples has been added as well as some "ant" administration based operations allowing to deploy/undeploy "bar" files.

Hereafter you can find the list of features included in this RC:

- Powerful workflow API covering deployment, definition, runtime and history workflow data
- ReadOnly API and advanced resources (hooks, mappers and performer assignments)
- Standard (J2SE) vs Enterprise (J2EE) deployment
- JEE deployment includes support for both 1.4 and 1.5 standards
- Support for XPDL 1.0 activities : Join, Split, Activity (Route, implementation no and subFlow) in both start and finish modes automatic and manual
- Support of main XPDL 1.0 elements : Datafield, DataType, Participant, Transition, RedefinableHeader, Transition Restriction and Package...
- Support of advanced entities/resources: Hooks, mappers and performer assignments
- Persistent execution
- Subprocesses support
- Iterations support

- Activities deadlines support through the Process Virtual Machine generic and configurable Timer service
- Configurable journal and history workflow modules
- Advanced process deployment capabilities including ".bar" file deployment and local vs global resources (hooks, mappers)
- Default persistence service implementation based on Hibernate
- Standard security service based on JAAS LoginModules: Test, standard and J2EE login modules are included in the package
- Configurable identity service. Default in-memory vs FileSystem vs database based implementations
- Unified life cycle for workflow activities (XPDL activities types) execution handling synchronization with Tasks, also known as manual activities, life cycle.
- Task Management module handling init, ready, executing, finished, dead, suspend and resume states
- Transitions conditions basic support based on BeanShell scripting language
- Support of ProEd XPDL designer
- Workflow data: both process and activity level variables support
- Default mapper implementation: Initiator Role (aka properties roleMapper in Bonita v3)
- Process Virtual Machine 1.0.beta2 integration

1.3. Restrictions

Nova Bonita RC2 comes out with an innovative architecture based on a generic and extensible engine, called "The Process Virtual Machine" and a powerful injection technology allowing services pluggability.

Nova Bonita Release Candidate 2 also include basic support for elements defined in the XPDL 1.0 standard. Next release candidates versions will continue to improve the standard coverage and services support: asynchronous execution, versioning... Check the roadmap [http://wiki.bonita.objectweb.org/xwiki/bin/view/Main/Roadmap] for more information.

This Release Candidate does not support the following features available in Bonita v3:

- block activities
- Process definition and process modifications via Java APIs
- process versionning
- Hooks: processes hook (onInstantiate) as well as activity onCancelled hook are not yet supported
- Role mapper: ldap type is not supported

Chapter 2. Prerequisites

2.1. Hardware

A 1GHz processor is recommended, with a minimum of 512 Mb of RAM. Windows users can avoid swap file adjustments and get improved performance by using 1Gb or more of RAM

2.2. Software

- Nova Bonita requires Java Development Kit (JDK) 1.5 (also called JDK 5.0) but also runs with with next release.

  The JDK software can be downloaded from http://java.sun.com/j2se/1.5.0

- Nova Bonita requires Apache Ant 1.6.5 or higher

  It can be downloaded from http://ant.apache.org

Chapter 3. Change history between Bonita v3 and Nova Bonita

Main concepts and features that made the friendly usage and the Bonita v3 brand have been kept: hooks, role mapper, performer assignments, local/global variables, rich and powerful API. Most of these features have been revisited in order to become even more efficient thanks to the PVM execution environment. Aim was to be the most compatible with the last version but of course some changes are required.

Goal of this chapter is to list/focus all these differences.

3.1. Concept of package

The concept of package has been introduced by XPDL specification from the WfMC in order to be a container for main workflow objects that can be shared by multiple workflow processes that can support an overall business application. Amoung these elements are: participants, datafields, others process workflows/sublows.

This concept has been natively taken in account by Nova Bonita engine. According the requirements and needs of our customers this concept should be enforced.

3.1.1. Package life cycle

States for package: UNDEPLOYED, DEPLOYED

3.2. Processes, instances, activities and tasks life cycles

One major change concerns the adding of task entity. If the activity is manual (ie. startMode=manual) when the execution enters the graph node of the activity a task is created. This task has its own life cycle with some synchronisation with the activity entity. Within this version task is still managed by the engine but in the future, it will be possible to plug an external task module to manage the tasks.

3.2.1. Process life cycle

States for process: UNDEPLOYED, DEPLOYED

Deployment of processes implies deployment of a package. Same thing for the undeployment. Package can be deployed and undeployed several times in order to make modifications onto its contained elements (process, participants, activities, ...). This is the way to maintain processes before the introduction of versionning in next version.

See the developpement guide for more details.

3.2.2. Instance life cycle

States for process instance: INITIAL, STARTED, FINISHED

No difference with bonita v3.

CANCELED state is not yet supported.

3.2.3. Activity life cycle

Activity has no state (oly started and finshed date). Notion of state at this level depends only on the type of behavior defined within the activity. A specific body of the activity is created according the type of the activity (Task, Subflow, Route, Automatic). The state is implemented by the body. At now only task has a life cycle.

3.2.4. Task life cycles

States for task: INITIAL, READY, EXECUTING, SUSPENDED, FINISHED, CANCELED

State SUSPENDED has been introduced. This state can be reached either from READY or from EXECUTING. transition is also reciproque.

Tasks are particular types of activities (such subflow, route...) associated to human actors.

3.3. APIs

Bonita v3 APIs were divided into 5 different areas and can be compared to bonita v4 API (see Chapter 4)

- ProjectSessionBean: is covered by both DefinitionAPI (for set/add methods) and QueryDefinitionAPI (for get methods)
- UserSessionBean: is covered by both RuntimeAPI and QueryRuntimeAPI
- AdminSessionBean: has fonctions that could be found into QueryDefinitionAPI and QueryRuntimeAPI according on the type of information (runtime or definition information). At now there's no check for admin role
- UsersRegistrationBrean: is not relevant for bonita v4 because user base is not managed by the engine.
- historyAPI: is covered by QueryRuntimeAPI.

Bonita v3 can only be acceeded as a remote workflow server. Bonita v4 supports both java workflow library and remote workflow server (see Chapter 4).

A new API has also be added for improving workflow processes deployment as well as advanced entities deployment: hooks, mappers and performers assignments. This API is called ManagementAPI. No need anymore to deploy xpdl first and the compile and copy by hand advanced entities in a particular server directory. Any deployment/undeployment operation can be performed through the ManagementAPI.

Furthermore Nova Bonita v4 provides extensibility to the APIs by the addition of the commandAPI. Developper is now free to write and execute its own commands and consequently can extends the proposed API. This is a service oriented feature and it also should avoid to provide a querier language for complexe request (involving requests with multiple criteria).

3.4. Hooks

Pieces of end-users java code that are executed at particular moments of either a process instance or a task or an "automatic activity" (route and subflow type activity cannot have hook).

3.4.1. for task

Xpdl definition of hooks has changed in order to extend rollbacking capabilities to all hook types making by the way the usage of hook simpler. An example of the new one is given here after

```
<ExtendedAttribute Name="hook" Value="org.ow2.bonita.integration.hook.OnReadyHook">
  <HookEventName>task:onReady</HookEventName>
  <rollback>false</rollback>
</ExtendedAttribute>
```

The element rollback has been introduced to indicate if the hook will be or not rollbacked.

Hook events have also been adapted to match the constraints of task life cycle.

- task:onReady
- task:onStart
- task:onFinish
- task:onSuspend
- task:onResume

Main change is the suppression of before/after for terminate and before/after start types because of the introduction of the rollback parameter. An other change is the introduction of new events due to the new state: SUSPENDED.

Note: if using proEd, the designer can select for each hook:

- either rollback=true (case1)
- or rollback=false (case2)

Hooks are always executed into a transaction. In case1, if an exception has occured the exception is raised by the engine and the transaction is rollbacked. In case2, the occuring exception is catched by the engine.

To implement a hook class the developper has the choice between two interfaces. Look at the javadoc of these interfaces for more details:

- org.bonita.ow2.definition.TxHook
- org.bonita.ow2.definition.Hook

If rollback=false has been previously defined, only Hook interface can be implemented otherwise an exception is raised at runtime. Then it prevents the use of TxHook interface. These hooks are intended to execute not critical operations for the workflow. Only query API are proposed to be acceeded into the parameters of the execute() method of the interface.

If modification on hook class is required it can be hot deployed to replace the previous one (see the ManagementAPI). It can be also deployed within the bar archive or independendtly. It can be also undeployed if the class is not required by a deployed process.

3.4.2. for automatic activity

One type of event can be defined.

- automatic:onEnter

3.4.3. for process

ON_INSTANTIATE hook (set within the process element of XPDL definition) is not yet supported.

ON_CANCELLED hook is not yet supported.

3.4.4. Interactive hook

Interactive hooks (also called Bean Shell) are not yet supported for activity and process. Those hooks will be implemented soon adding support for others scripting languages such Groovy

3.5. Deadlines

Deadline feature within Bonita v4 is the same as for Bonita v3. org.ow2.bonita.definition.TxHook or org.ow2.bonita.definition.Hook interface must be implemented in case of deadline hook (ON_DEADLINE event). See javadoc for more details.

3.6. Mappers

org.ow2.bonita.definition.RoleMapper interface must be implemented (see javadoc for more details).

Main difference concerns the moment in which the searchMembers() method is executed. In Bonita v3 it was executed at process instantiation since in Bonita v4 it is at the creation of the task from which the activity has been defined with a role mapper. It has the advantage to take in account modification of the groups within the external user base

3.7. Performer assignments

org.ow2.bonita.definition.PerformerAssign interface must be implemented (see javadoc for more details).

3.8. Variables

Properties entity in Bonita v3 has been renamed to Variables in Bonita v4. This seems a more natural way to work with workflow relevant data.

Variables support and flexibility in Bonita v3 was too limited. Only String and enumerated types were supported. In Bonita v4 support for common variables types as well as as advanced ones (including own Java based ones) will be added in next RC (currently, the RC2 version support same types than v3).

Getting and Setting variables operations directly handles Java Objects, meaning that a get operation returns an Object so the developer only needs to use the instanceOf operator to determine the type of a particular variable.

3.9. Iterations

Iterations support in Nova Bonita follows the innovative mechanism included in Bonita v3, meaning supporting complex and advanced uses cases: unstructured iterations or arbitrary cycles.

Main difference between Bonita v3 and v4 related to iterations is that in v4 there is no need anymore for a dedicated entity called iteration. Transitions can be used in Bonita v4 to create a cycle in a workflow processes.

For compatibility reasons iterations entities defined as XPDL extended attributes (Bonita v3) are still supported.

The current implementation in Nova Bonita RC2 has some restrictions:

- A cycle must have at least one XOR entry point

- Split activities as exit points are only supported in case of XOR

- Join XOR inside iterations do not cancel/delete non selected execution path

Those restrictions will be fixed in next RCs with the addition of a new behaviour for XOR activities in which non selected execution path will be automatically deleted/removed

Chapter 4. Installation guide

4.1. Installation

Nova Bonita Release Candidate release adds support for both standard and enterprise deployments. After unzipping this release you could easily use Nova Bonita "as a library" inside your web or rich client application or to deploy it into you favorite application server and use it remotely.

So, first of all you should start by unzipping the Bonita distribution package:

```
>unzip bonita-4.0.RC2.zip
```

A new directory `bonita-4.0.RC2` will be created with the following structure:

```
README
build.xml
build.properties
License.txt
release_notes.txt
conf/
doc/
examples/
ear/
lib/
```

Let's describe those items :

- README

  This file gives the basic information related to Nova Bonita

- build.xml

  This file is an ant file that provides tasks to run both unit tests and examples (detailled command are given in following sections).

- build.properties

  This file contains the J2EE properties required to deploy and to use Nova Bonita APIs deployed in a remote J2EE server (by default properties are set to deploy in the EasyBeans EJB3 container)

- License.txt

  The license of Nova Bonita. Bonita is released under the LGPL license.

- conf/

  This directory contains default configuration files for Nova Bonita. That includes the "Environment" xml file (including services and objects used as default by the engine), login modules (JAAS compliant login modules samples) and hibernate persistence configuration (as a default implementation to handle Nova Bonita persistence). Standard (JSE) and Enterprise (JEE) versions are provided as well as out of the box integration with JBoss and JOnAS application servers as well as with Easybeans EJB3 container

- doc/

  This directory contains the documentation of Nova Bonita. It contains 2 directories :

  - html/

For HTML documentation

- pdf/

  For PDF documentation

- examples/

  This directory contains an example provided with Nova Bonita package. This sample application illustrates how to use Nova Bonita APIs from within a client application. That includes some samples projects illustrating hwo to use those APIs in both JSE and J2EE environments.

  - Approval Workflow sample

    This is a generic Approval Workflow process. Workflow deployment, definition and execution phases are illustrated in this sample. The sample application is provided in both standard and enterprise environments in which the workflow APIs are leveraged as POJOs or as Session Beans respectively.

  - Carpool sample

- lib/

  This directory contains the libraries used in Nova Bonita RC2. Nova Bonita can be integrated in your applciation/IS in different ways (integrated in a web application, inside a rich client application, remotly deployed in a JEE application server...). Depending on your integration environment only some of those libraries will be required.

4.2. Standard vs Enterprise installation

Find hereafter some instructions about how to deploy and to reach Nova Bonita in both Standard and Enterprise environments:

4.2.1. Standard installation

To integrate Nova Bonita in your application you only need to add a couple of libraries to your environment.

The following ones are the only required by Nova Bonita:

```
bonita.jar
pvm.jar
bsh.jar
novaBpmIdentity.jar
novaBpmUtil.jar
xstream.jar
```

To leverage the Nova Bonita default persistence service based on hibernate, you should also need those ones:

```
hibernate3.jar
dom4j.jar
commons-logging.jar
commons-collections.jar
hsqldb.jar
cglib.jar
antlr.jar
asm.jar
asm-attrs.jar
jboss-j2ee.jar
```

4.2.2. Enterprise installation

Move to the Nova Bonita installation directory and:

- call "ant ear.eb", "ant ear.jboss4" or "ant ear.jonas4" respectively task to generate the bonita.ear file corresponding to your favorite application server.
- deploy this ear into your favorite JEE 1.4 or 1.5 application server.
- In you are using another application server that the one listed before, you could easily reuse the bonita.ear generated by one of the previous tasks and deploy it in your environment (Weblogic, Websphere...)

4.2.2.1. EasyBeans EJB3 Installation and deployment (a complete example)

Here after you will find steps required to deploy Nova Bonita (JEE version) in EasyBeans EJB3 container:

- Download easybeans up to RC3 jar file at http://maven.objectweb.org/maven2/org/ow2/easybeans/easybeans-uberjar-toplink-essentials/1.0.0.RC3/easybeans-uberjar-toplink-essentials-1.0.0.RC3.jar
  Easybeans is using a directory called 'easybeans-deploy' in the basedir to deploy new archives

  Note: At the time of writing this documentation we got informed that EasyBeans team has released the 1.0 final version.
- Create a directory with this name in the folder from where you will start easybeans container and copy the bonita.ear file in to the created 'easybeans-deploy' directory
- Be sure to have all security permissions in your java.policy file: permission java.security.AllPermission
- Then start easybeans :

  java $JAVA_OPTS -Dorg.ow2.bonita.environment="name of the envrinoment file to use" -Djava.security.manager -Djava.security.policy="path to your java.policy" -jar "easybeans RC3 jar file"
- Then start easybeans :

  WARNING : if you want to use another DB than HSQL, you should start your EE container with the appropriate driver in the classpath. In that case, instead of starting easybeans with -jar option, you have to start it like : -cp "path to driver or $CLASSPATH...":"easybeans RC3 jar file":"easybeans main class" Easybeans main class can be found in the MANIFEST file of Easybeans jar file (org.ow2.easybeans.server.EasyBeans)

Chapter 5. Configuration and Services

This chapter introduces the services configuration infrastructure provided by Nova Bonita as well as main services included in this RC2 version.

5.1. Services Container

The Process Virtual Machine technology includes a services container allowing the injection of services and objets that will be required during workflow definition and execution. Objects and services used by the Bonita engine are defined through a XML file. A dedicated parser and a wiring framework are in charge of creating those objects. Security, identity, persistence, notifications, human task and timers are examples of pluggable services.

This services container (aka IoC container) can be configured through a configuration file. A default configuration file is included in the package under the /conf directory (environment.xml):

```
<environment-definition>

  <application>
    <variable-types resource='xpdl.type.resolver.xml' />
    <hibernate-session-factory>
      <properties resource='hibernate.properties' />
      <mappings resource='xpdl.hibernate.mappings.xml' />
      <cache-configuration resource='xpdl.cache.xml' usage='read-write' />
    </hibernate-session-factory>
    <standard-command-service>
      <environment-interceptor />
      <transaction-interceptor />
    </standard-command-service>

    <job-executor threads='1' auto-start='true' />
    <api  type='AutoDetect' />
    <retry nb='3' />
    <!-- XmlArchiver is thread safe, it can be shared by all environments -->
    <chainer name='archiver'>
      <archiver class='org.ow2.bonita.persistence.log.LoggerArchiver' />
      <history name='history' class='org.ow2.bonita.persistence.xml.XMLHistory'></history>
    </chainer>
    <chainer name='finished-instance-handler'>
      <object class='org.ow2.bonita.services.handlers.impl.DeleteFinishedInstanceHandler' />
      <object class='org.ow2.bonita.services.handlers.impl.ArchiveFinishedInstanceHandler' />
    </chainer>

    <chainer name='undeployed-package-handler'>
      <object class='org.ow2.bonita.services.handlers.impl.ArchiveUndeployedPackageHandler' />
    </chainer>

  </application>

  <block>
    <!-- DbJournal cannot be shared by several environments.
    It contains a session cache that needs to be recreated for each environment -->
    <chainer name='recorder'>
      <recorder class='org.ow2.bonita.persistence.log.LoggerRecorder' />
      <journal name='journal' class='org.ow2.bonita.persistence.db.DbJournal' />
    </chainer>
    <!-- Query Api has an object reference to the journal,
     so it cannot be shared by multiple environments -->
    <queryApi name='queryList'>
       <ref object='journal' />
       <ref object='history' />
    </queryApi>
    <!-- DbRepository cannot be shared by several environments.
    It contains a session cache that needs to be recreated for each environment -->
    <repository class='org.ow2.bonita.persistence.db.DbRepository' />
    <timer-session />
    <standard-transaction />
    <job-session />
```

```
    <hibernate-session />
    <hibernate-xpdl-persistence-service />
  </block>

</environment-definition>
```

Currently, following objects implementations can be injected in the environment:

- repository: data repository storing workflow processes, instances, activities... Db persistence (class org.ow2.bonita.repository.db.DbRepository) implementation is included in this RC.
- recorder: object responsible of workflow execution logs. Default implementation handles workflow logs in the command line console (org.ow2.bonita.services.record.impl.LoggerRecorder). Recorder and Journal (see next) objects can be chained (new ones can be added as well on top of the recorder chainer). This give you a powerful mechanism to handle workflow execution data
- journal: object responsible for storing or retrieving workflow execution data. Db persistence (class org.ow2.bonita.services.record.impl.DbJournal) implementation is provided.
- archiver: object intended for workflow logs archiving. Default implementation handles logs on workflow data archiving through the default implementation (class org.ow2.bonita.services.record.impl.LoggerArchiver). Archiver and History (see next) objects can be chained (new ones can be added as well on top of the archiver chainer). This give you a powerful mechanism to handle workflow archived data
- history: object intended for storing or retrieving workflow archieved data. Default implementation provided is the one available in the following class: org.ow2.bonita.services.record.impl.XMLHistory. This class will store workflow history in the file system as XML files
- queryList: object intended to configure how the QueryRuntimeAPI will retrieve the workflow execution data. This retrieval could be configured to chain with the expected order into the journal and the history.
- finished-instance-handler: action to perform when a workflow instance is finished. This object could chain two distinct actions: for a given workflow instance, deleting the runtime object including its tasks from the repository and then store data in the archive and remove data from journal. Default implementations are proposed for both chained actions.
- undeployed-package-handler action to perform when a workflow package is undeployed. Default implementations are proposed for both chained actions stores undeployment data into the archive

* Note 1: the attribute "api type" defines the execution context for Bonita: JSE vs JEE (1.4 or 1.5). Possible values are auto-detect, standard, ejb2 and ejb3. Auto-detect meaning that if Nova Bonita is running on a JEE application server this will be the default execution context. Execution context is a key concept as it give to Nova Bonita the right nformation about security (JSE vs JEE) and APIs types (POJO vs Session Beans).

* Note 2: As explained before persistence objects are provided as default implementations in the environment. Notice that in a persistence configuration additional resources are required, i.e for hibernate persistence you can specify mapings, cache configuration...

* Note 3: The environment is divided in two different contexts: application and block. Objects declared inside the application context are created once and reused while objects declared inside the block context are created for each operation.

5.2. Services

Services in Nova Bonita is all about pluggability. Standard (StandAlone Java based) and Enterprise (JEE Server based) versions of Nova Bonita can be easily configured thanks to the services container. To

allow that, each workflow related service has been thought in terms of an interface with different possible implementations. In the following lines you will find a description of main services supported in Nova Bonita:

5.2.1. Persistence

Persistence is one of key technical services injected into the services container. This service, as well as other major services in Nova Bonita, is based on a service interface. That means that multiple persistence implementations can be plugged on top.

The Persistence service interface (called DbSession) is responsible to save and load objects from a relational database. By default, a persistence implementation based on the Hibernate ORM framework (called HibernateDbSession) is provided (JPA and JCR to come).

The Process Virtual Machine core definition and execution elements (processes, nodes, transitions, events, actions, variables and executions) as well as the XPDL extension ones (join, split, manual and automatic activities, conditions, variables...) are persisted through this service. Process Virtual Machine core elements are also cached by leveraging the default persistence service implementation (Hibernate based). Workflow packages, processes, instances, tasks and advanced classes (such hooks or mappers) are are stored through this persistence service. Workflow repository is the term used in Nova Bonita to store those entities.

5.2.2. Identity

Identity service main objective is to give freedom to system administrators to leverage your organization user repository. Traditional user repositories such LDAP, ActiveDirectory as well as any other user repository (database or API) can be plugged as implementations of this service.

By default, some user repositories implementations are provided for testing purposes: in memory, basic FileSystem based persistence, and basic database persistence (based on a predefined database schema). Those implementations can also be used in production if there is no other user repository available.

The Identity service is so an extensible interface (known as IdentityServiceOp) build around three main concepts: Users, Groups and Memberships:

- User: a particular user inside an users repository. Users can be created, modified, removed and queried (some of those operations could be not allowed for some repositories (i.e LDAP) through the IdentityService API).
- Group: a group of users in a particular users repository. A group could contain either users security restrictions or hierarchical information. As for users, groupes can also be created, removed, modified and queried.
- Membership: a membership represents a user position in a particular group. An user could have two different membership in two different groups. Membership related operations concern set, remove or updates on users position inside groups.

Both Security and Human Task services will use the Identity one by checking user login/password and user rights (Security) and by resolving workflow logical roles with users and so to assign manual activities to users based on some hierarchical information (Tasks Management)

By default, Nova Bonitais packaged with a test based identity module based on a properties file. This file contains the user/login allowed to reach Nova Bonita APIs. This properties file is in fact a Test Login Module (see security module description below), meaning that the same properties file is used for security and identity configuration.

5.2.3. Security

The security service is based on JAAS standard. Main purpose of this service is to provide both authentication and authorization capabilities to the workflow engine. As security directly relates to users permissions, this service also relates to the identity one (commonly security is based on top of the identity service).

As for other services, the Nova Bonita team is concerned on let you the freedom to choose and plug your favorite security implementation. At the same time we also want to provide one ore more default implementations that allow users to quickly set up and start playing with Nova Bonita.

For testing purposes Nova Bonita includes a default JAAS login module checking user/password values stored in a file. This easily allow to start playing with Nova Bonita in a testing security environment in which the login module acts as a lightweight users repository. This login module (org.ow2.novabpm.identity.auth.PlainLoginModule) is the one provided in Nova Bonita examples directory.

The current implementation of the security service allows you directly work with the default identity service to handle users authentication. Users must login before start calling the Bonita APIs. The current Nova Bonita RC is not yet leveraging the Authorization capabilities provided by traditional security services, meaning that any authenticated user can reach Nova Bonita APIs. In next releases autorization will apply at APIs methods level. We could also imagine fine grained security uses cases in which authorization is not only applied at API method level but also applies to the engine itself, to a particular workflow instance or to the actions executed by the engine.

The Security service is composed by two different JAAS LoginModules. The first one (called PlainLoginModule) is responsible to handle security authentication and authorization. This one could be just replaced by you favorite JAAS Login Module. The second one (StorageLoginModule) is responsible to keep data of authenticated users (basically for security context initialization). Those login modules can be configured in both standard and enterprise environments (note that most of JEE servers already provides a Storage Login Module so you could just replace the one proposed by Nova Bonita by the one leveraged by you app server.)

5.2.4. Task Management

Task management is all about providing the right information to the right people at the right time !. This is one of the most important services that must be provided by a workflow solution.

As human task management can be re-used in other domains (not only by workflow solutions but by any Java based application) we wanted those features to be a service rather than an internal workflow module. As a result, this service is generic and extensible Task Management service that can be either used in Nova Bonita extension to handle manual task assignments and executions or either by any Java application or Domain Specific Language (i.e BPEL4People extension for instance).

Traditional features such users - roles/group mapping, delegation, scalation, task deadlines handling or manual activities execution life cycle are in scope of this service. Advanced features such configurable activity life cycle, interactions with other task managements system, services or collaborative applications and integration with organizational rules are also part of the main responsabilities of this service.

The current implementation focus on support of manual tasks (also known as manual activities) in Nova Bonita. Basic features such Bonita RoleMappers and Performer Assignments entities allowing users - roles mapping are already supported. Together with the identity and security service, users can login into the system, get their tasks todo list and execute them. As other service in Nova Bonita this module is executed in a persistent environment.

5.2.5. Journal and History

This module concerns the way in which the workflow data is stored during the workflow execution and archived when the execution is completed. This is indeed a crucial module in a workflow solution.

While in Bonita v3 journal data (aka execution workflow data) and history data (aka archived data) where handled by different mechanism, in Nova Bonita we decided to unify them as the underlying essence of both is to handle workflow data. For that to be done, we created the concept of workflow record. A record is a minimal set of attributes describing a workflow entity execution. That means that each workflow entity related to the execution has its own associated record: instance record, task record, hook record...

Those records are recorded during the workflow execution and stored depending on the persistence service implementation (db, xml...). The Nova Bonita API will retrieve record data from the records storage and sent them back to the users (meaning that records also acts as value objects in Nova Bonita APIs).

As soon as a workflow instance is finished, a typical scenario would be (by default) to move instance related workflow data from the production environment to a history one. While the physical device and the data structure could changed from one workflow engine deployment to another (XML, BI database...), the internal format could remain the same (records). This is exactly what is happening in Nova Bonita, when archiving data the engine just move execution records from the production to the history environment without data transformation inbetween.

5.2.6. Timers

To handle activities deadlines, a timer service is required that can schedule timers to be executed in the future. Timers must have the ability to contain some contextual information and reference the program logic that needs to be executed when the timer expires. A typical scenario would be a manual activity (task) that needs to be monitored with a timer. For example, if this task is not completed within 2 days, notify the manager.

This service, as well as any other asynchrous service in Nova Bonita is based on the Process Virtual Machine Job executor framework. Job executor framework is responsible for handling jobs. A job could be a timer scheduling or an asynchronous message for instance. When a job is created and stored in the database, the job executor starts a new transaction, fetch the job from the database and perform the instructions contained in the message.

Chapter 6. Concepts

6.1. Terminology

As Bonita is XPDL compliant, refer for more details to the terminology defined within XPDL.

- A **package** issued from XPDL acts as a container for main workflow objects that can be shared by multiple workflow processes. .
- A **process** (called Workflow Process into XPDL) contains the elements that make up a workflow: activities, data fields, patrticipants, transitions, ...
- An **activity** is the base workflow entity to build a process. It contains others sub entities that will determine the behavior of the activity (the implementaion : no or subflow, the start mode: manual/ automatic, the performer, the performer assignment, the transition restrictions : Split or Join).
- A **task** is a runtime object created for specific activity type also called manual activity. Workflow tasks should be manage by an independant module receiving tasks from other applications.
- A **participant** is one of the following type: system, human, role. It is defined within a task.
- A **transition** is a dependency expressing an order constraint between two activities.
- A **variable** (called Workflow Relevant Data into XPDL) is a workflow unit of data. Variables can be local to an activity or global to the process.
- A **hook** is user defined logic adding automatic or specific behavior to activities and workflow processes
- A **mapper** is a unit of work allowing dynamic role resolution each time an activity with human task behavior is created (instantiated).
- A **performer assignment** is a unit of work adding additional activity assignment rules at run time.

For most of the entities defined here before, definition data, runtime recorded data, archived data are managed within the engine. To easily play with these three aspects that characterizes workflow entities Bonita has introduced UUID (universally unique identifier). Each type of data has its own typed UUID that can be used within operations of the facade API.

6.2. Package

As introduced in section 3.1 (Conceps), the package element within an XPDL file contains: processes, participants, datafields..... Multiple processes can be deployed. These processes can share participants and datafields.

Deploying processes implies to deploy at leat a package (ie. the XPDL file that contains the package element). Package is the unit of deployment.

The notion of package concerns also the scope for the deployed java classes (for hooks, mappers and performer assignments). If these classes are deployed at same time of the package deployment, these classes are dependant on the package and visible for all processes of the package (and there activities). The undeployment of the package will undeploy all these classes. Classes could be also deployed at global level and acceded by the processes/activities of all the packages.

QueryDefinitionAPI gives access to all deployed and undeployed package informations with the default environment configuration.

6.2.1. LifeCycle

A package has its own life cycle:

- **Deployed:** after the deployment operation (calling the Management API) state of the package is **deployed**.
- **Undeployed**: when undeploy() operation is performed the state of the package becomes **undeployed**. An 'undeployed-package-handler' is then called. With the default environment configuration, recorded data for deployed package are archived (history).

6.2.2. Versioning

Versionning is not fully managed for this RC2 version. Contraints on regards to the version for the deployment are the following:

- It's forbidden to deploy two time in sequence a package with the same package Id even the version are not the same.
- The (deployed) package must be undeployed and the version of the package changed to be able to deploy again the package

6.3. Process

Processes are defined into a package and deployed by deploying a package.

6.3.1. Process Basics

- **Process** (aka process model): workflow process containing the workflow definition logic. These projects can be instantiated by users.
- **Instance**: workflow process representing a specific execution of a workflow process.

6.3.2. Life Cycles (process/instance)

A process has the following life cycle:

- **Deployed**: when the package containing the process has been deployed, the process has been created within the engine and its state is **deployed**.
- **Undeployed**: the process is also undeployed via the undeployment of the package containing the process. It state become **undeployed**.

A process instance has the following life cycle:

- **Initial**: once the process instance has been created its state is initial.
- **Started**: when instantiateProcess() method of the RuntimeAPI is called, firstly the instance is created (**Initial** state) and secondly the execution is automatically started which causes the state of the instance to become **started**.
- **Finished: when the execution has reached the bonitaEnd activity the instance state is set to finished.**

6.3.3. Processes (aka models) & process Instances

There are scenarios where the re-use of a process definition is of key importance; in these scenarios, a long-time is spent carefully defining a generic process model that instantiates in the same way many times. These processes are called administrative processes (aka process models).

A process is a specific definition of a process that may be instantiated multiple times. These processes are based on a process-instance workflow paradigm. Processes are created into the the engine via the **ManagmentAPI** (deploy operations giving an XPDL file). No **java DefinitionAPI** allowing the creation of a process is supported for this release canditate. When the process is created, the workflow users are able to instantiate the workflow process via **RuntimeAPI** to create process instance(s). Once the process instance(s) are created, workflow participants can access the **QueryRuntimeAPI** to accomplish the following: obtain their "ToDo list", done list, suspended list, execute assigned activities, or access the **QueryDefinitionAPI** to get complementary/static definition informations onto tasks, instances, participants....

A process model keeps track of all its instances. That is, all instances of this process are retrievable through the **QueryRuntimeAPI** functions.

Bonita instantiation mechanism:

At process instantiation a new (XPDL) process object issued form the deployed process definition is created (and initiated with definition elements and specific parameters such as variables). A root execution object pointing to the process object is created and started. It causes the execution to point/enter onto the first activity of the process. The root execution references a ProcessInstance object representing the runtime data to be recorded in the journal all along the life of the process instance.

6.3.4. Versioning

Versionning is not fully managed for this RC2 version. Contraints on regards to the version for the deployment are:

- It's forbidden to deploy in sequence a process with the same process id even if the versions are not the same.
- The (previously deployed) process must be undeployed and the version of the process changed to be able to deploy again the process with the same id.

6.3.5. Concept of Hooks

Hooks are user-defined logic that can be triggered at some defined point in the process life cycle. No process hooks **are supported for this RC**.

The types of Hooks are:

- OnInstantiate hook is called before the workflow instance is created. The OnInstantiate hook is not considered to be in the same transaction as the process instantiation action.
- OnTerminate hook is called automatically after workflow instance termination ends.

6.3.6. SubProcesses

Sometimes, an independently existing business process can take part in another more sophisticated process. Instead of redefining the activities, edges, properties, and hooks in the parent process, the independent

process could be included as a “subProcess” within a specific node. As the execution logic is inside the subProcess, the subProcess activities are started and terminated automatically by the Workflow engine according to the subProcess state. **Creating a SubProcess Activity:** When a subProcess activity is defined in the process, a specific activity with subflow behavior is created with the definition of the parsed activity (process id of the process, local variables, in/out/in-out parameters of the sub...).. **Instantiating a Process with a SubProcess Activity:** At execution phase when the execution enter into the subflow type activity, the following operations are done:

- an instance of the process referenced by the subflow activity is created.
- A new root execution is created into this instance and is automatically started (then execution enters in the first activity of the subflow).
- local variables of the subflow activity (defined with an extended attribute) are created as global variables of the instance of the subflow (this is the way to pass variables to a subflow when the processes are defined under **proEd editor**), As variables are not propagated, transmission of variables/values could be done thru hooks executed within activities of the subflow (setting variables into the parent process).
- if both formal parameters into the subflow process and actual paremeters into the subflow activity have been defined, the list of actual parameters are passed to the subflow (these XPDL definitions are supported by Bonita engine but not by proEd editor).

6.4. Activities

Activity has no state (only started and finished date). Notion of state at this level depends only on the type of behavior defined within the activity. A specific body of the activity is created according the type of the activity (Task, Subflow, Route, Automatic). The state is implemented by the body. At now only task has a life cycle.

States of task are: INITIAL, READY, EXECUTING, SUSPENDED, FINISHED. Suspended state can be reached either from READY or from EXECUTING. Transition is also reciproque.

6.4.1. Activity Basics

The activity is the basic unit of work within a process.

Several types of activity exist according on the activity definition.

- **Manual activity** (startMode = manual, Implementation = No): When the execution enters a manual activity a task (aka human task or user task) is created. QueryRuntimeAPI allows to get access to the task according on its state. RuntimeAPI allows to manage the task (start, suspend, resume, finish). In further release task could be managed by an other module.
- **Automatic activity** (startMode = automatic, Implementation = No) : the activity is automatically executed by the engine.
- **Route**: (Route element): the activity is automatically executed by the engine.
- **Subflow activity** (implementation = Subflow startMode = automatic): the activity is automatically executed by the engine.

Task has its own life cycle whith the folowing states:

- **Initial**: This is the state of an activity ready to be started. There are two possible situations for this state to occur. In the first, an activity has no parent activity (this is the first activity of the workflow process).

In the second, a normal activity has parent activities that have all terminated successfully, and whose transition conditions to the activity have been successfully evaluated.

- **Ready**: This is the state of an activity waiting for some processing to complete before being ready to run. In the case of normal activities, at least one of the parent activities is still executing. In the case of an activity that can be anticipated, at least one of the parent activities has not started.
- **Executing**: An activity in execution.
- **Suspended**: An activity having initially a Ready state or Executing state has been suspended. Resume operation put back the activity with its initial state. Its intial assigned user is also restored.
- **Finished**: An activity that has terminated successfully.

Bonita engine creates a common object in order to manage runtime informations for all activities. The interface InstanceActivity is returned by a lot of the QueryRuntimeAPI methods to give access to these runtime informations. According the type of the activity, a specialized body can be acceeded.

- Task body
- Automatic body
- Route body
- Subflow body

Activity itself has no states. Only start and finish date are managed.

Route and subflow activity type can't execute hook. Task has its own hook events. Automatic activity can execute one type of hook event executed after the execution enters in the activity.

Chapter 7. User guide

This chapter describes how to start playing with Nova Bonita RC2. More precisely it describe main steps to define and deploy workflow processes and how to start running them in Nova Bonita:

- How to create a process definition
- How to deploy a process definition
- How to develop a simple application by leveraging Bonita APIs

7.1. Designing a xpdl process with ProEd

Like in previous Bonita versions, processes could be created either through a java api or through a graphical editor : ProEd. The java api to build Bonita v4 processes is not yet developped, so processes should be deployed as .xpdl files. That can easily be done under ProEd (see restrictions below) and then imported as xpdl files. For that, use the stand alone version of the process editor that can be downloaded on Bonita download [http://forge.objectweb.org/project/showfiles.php?group_id=56&release_id=302/] page.

Please, refer to the Nova Bonita development guide, under /doc directory to learn more about how to use ProEd.

As final result ProEd saves the description of the designed process as an xpdl file that should be imported as described in the following section.

7.2. Nova Bonita APIs

If you are already familiar with previous Bonita versions and you have already developped your own applications on top of Bonita, we want to minimize your effort when migrating to Bonita v4. Compatibility from Bonita v3 to v4 is one of our main concern.

At the same time, we took the chance to review and to improve Bonita v3 APIs in this new major version so basically the Bonita v3 API spirit is still there but we applied some improvements in Nova Bonita to simplify some operations and to add added value features.

Nova Bonita APIs has been refactored each milestone release until become stable in the first release candidate version.

7.2.1. Getting started with Bonita APIs

Nova Bonita APIs are divided into 5 different areas:

- **DefinitionAPI**: to create/modify major process elements into the engine (packages, processes, activities, role mappers, variables by calling java methods instead of importing xpdl files. It will allows also to modify the execution of runtime elements such as tasks and instances.
- **QueryDefinitionAPI**: to get workflow definition data for packages, processes, activities, role mappers, ...
- **RuntimeAPI**: to manage process, instance and task life cycle operations as well as to set/updates variables
- **QueryRuntimeAPI**: to get recorded/runtime informations for packages, processes, instances, activities, tasks (support for dynamic queries will be added in the future). It allows also to get tasks (aka manual

activities) list parametrized by the with state for the authenticated user and as well to get/list variables in a particular or a set of workflow instances.

- **ManagementAPI**: to deploy workflow processes into the engine. XPDL files and advanced entities such hooks, mappers and performer assignments can be deployed individually or in one shot

There's also a generic API that allow to execute specific commands that should be needed by the workflow based application.

- **CommandAPI**: to allow developpers to write and execute its own commands packaged within its application.

7.2.2. Nova Bonita APIs, playing with !

Nova Bonita RC2 is an extensible and embeddable workflow solution that can be easily deployed in both standard (JSE) and Enterprise (JEE) environments.

- Nova Bonita can be easily integrated in your application as a workflow library. In that case your application can directly reach the workflow APIs as POJOs.
- Nova Bonita can also be deployed in a JEE application server and so reached it remotely thanks to the Workflow Session Beans APIs.

7.2.2.1. Nova Bonita as a java workflow library in your application

The java APIs can be accessed thru an API accessor object as described in the example below:

```
APIAccessorImpl accessor = new APIAccessorImpl();
RuntimeAPI runtimeAPI = accessor.getRuntimeAPI();

QueryAPIAccessorImpl accessorQ = new QueryAPIAccessorImpl();
QueryRuntimeAPI queryRuntimeAPI = accessorQ.getQueryRuntimeAPI();
```

There are 2 different accessor interfaces available:

- QueryAPIAccessor: to get access to QueryRuntimeAPI QueryDefinitionAPI interfaces.
- APIAccessor: to get access to getRuntimeAPI, ManagementAPI, DefinitionAPI, CommandAPI interfaces.

APIAccessorImpl and QueryAPIAccessorImpl classes are implementing the two interfaces. Instantiation of these classes gives access to the expected bonita API.
There is also an utility class called AccessorUtil (under the org.ow2.bonita.util package) that allows developers to directly get the APIs in both standalone or Session Bean modes. You will find a sample application leveraging this API under the /examples directory

7.2.2.2. Nova Bonita as a remote workflow server

The session Bean APIs can be accessed by simply executing a lookup on the expected API interface. The jndi name is the name of the java interface:

```
RuntimeAPI runtimeAPI = (RuntimeAPI) initialContext.lookup("runtimeAPI");
QueryDefinitionAPI queryDefinitionAPI =
(QueryDefinitionAPI)  initialContext.lookup("QueryDefinitionAPI ");
```

7.2.2.3. Unifiying client applications source code !

An utility class has been provided to unify access to Bonita APIs and to avoid the use of lookups in JEE deployments: org.ow2.bonita.util.AccessorUtil. Through this class, Nova Bonita APIs can be reached in a unified way in both local and remote applications.

For that to be done, the system property called "org.ow2.bonita.api-type" must be defined at client side to specify whether the APIs will be reached locally or remotely (possible values are "standard","auto-detect","ejb2" and "ejb3").

Hereafter you will find an example on how to use the Accessorutil class from your client application:

```
RuntimeAPI runtimeAPI = AccessorUtil.getRuntimeAPI();
QueryDefinitionAPI queryDefinitionAPI = AccessorUtil.getQueryRuntimeAPI();
```

For a detailed insight on Nova Bonita APis, please take a look to the Nova Bonita javadoc [http://forge.objectweb.org/project/download.php?group_id=56&file_id=10312]

7.3. Running the examples

The Nova Bonita package contains some complete workflow examples. Those examples can be easily compiled, deployed and executed (in both JSE and JEE modes), thanks to a set of "ant" tasks available under examples directory. Hereafter you will find some information about how to play with one of those examples. For others, you can just replace the workflow name on the "ant" tasks:

- The Bonita Approval Workflow: a simple Approval Workflow application illustrating the workflow definition, workflow deployment and execution phases.

The build.xml in the root directory contains required targets to compile and launch the example in both standard (JSE) and enterprise (JEE) environments:

The Approval Workflow sample is configured to run together with the default hibernate persistence service. Core Process Virtual Machine entities, XPDL extension as well as execution related data will be persisted in a relational database. By default Nova Bonita embeeds HSQL database and uses it as a default database. You can easily change this default configuration and use your favorite database by modifiying the hibernate.properties file located under "conf" directory.

```
>ant aw-standalone
```

This sample application leverages the Security and Identity services so you must provide a right user login/passwd to run the sample. The default identity module (based on a properties file) is provided with three users ("John", "Jack" and "James" logins with "bonita", "jackpass" and "jamespass" as password). All three can login into to system but only John and Jack are able to play in the Approval Workflow sample. This behaviour is due to the users - role mapping defined in the previous workflow sample.

The java example simply deploy the xpdl file as well as advanced java entities such hooks and mappers in the engine (deployBar method in ManagementAPI), then creates a workflow instance and calls getTasksList, startTask and finishTask methods from the RuntimeAPI

This sample application can be launched in both standard and enterprise modes. In the enterprise mode Nova Bonita APIs are available as Stateless Session Beans. Enterprise sample version can easily be executed with the following ant tasks:

```
>ant aw-jonas4 (for a deployment in JOnAS application server version 4)
```

```
>ant aw-jboss4 (for a deployment in JBoss AS version 4)
```

```
>ant aw-eb (for a deployment in EasyBeans EJB3 container)
```

*Note that the enterprise version requires Nova Bonita (bonita.ear file) to be deployed in an application server

Nova Bonita RC2 unit test suite can also be launched from the installation directory (this operation is a good way to check that Nova Bonita was propertly configured)

```
ant tests
```