



Nova Bonita documentation

Bonita Workflow Team ()

- March 2008 -

Copyright © Bull SAS - OW2 Consortium

Table of Contents

Introduction	iii
1. General information	1
1.1. Nova Bonita introduction	1
1.2. Feature list	1
1.3. Restrictions	2
2. Prerequisites	3
2.1. Hardware	3
2.2. Software	3
3. Installation guide	4
3.1. Installation	4
3.2. Standard vs Enterprise installation	5
3.2.1. Standard installation	5
3.2.2. Enterprise installation	5
4. Configuration and Services	7
4.1. Services Container	7
4.2. Services	7
4.2.1. Persistence	8
4.2.2. Identity	8
4.2.3. Security	8
4.2.4. Task Management	9
4.2.5. Journal and History	9
5. User guide	11
5.1. Designing a xpdI process with ProEd	11
5.2. Nova Bonita APIs	11
5.2.1. Getting started with Bonita APIs	11
5.2.2. Nova Bonita APIs, playing with !	12
5.3. Running the example	12

Introduction

This documentation is targeted for Bonita users. It presents the installation procedure and a quick user guide of Nova Bonita Milestone 3 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 Nova Bonita Milestone 3

Chapter 4, Configuration and Services describes main configuration features and default services

Chapter 5, User Guide guides you through the discovery 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 M3 comes with an enhanced XPDL extension module, a reworked workflow API and a set of new features and services such journal and history services. Hereafter you can find the list of features included in this milestone:

- 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
- 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
- In-memory vs persistent execution
- Subprocesses support
- 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.beta1 integration

1.3. Restrictions

Nova Bonita Milestone 3 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 M3 also include basic support for elements defined in the XPDL 1.0 standard. Next versions will continue to improve the standard coverage and services support: asynchronous execution, versioning, iterations... Check the roadmap [<http://wiki.bonita.objectweb.org/xwiki/bin/view/Main/Roadmap>] for more information.

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

- block activities
- Process definition and process modifications via Java APIs
- process versionning
- iterations
- deadlines
- Hooks: processes hook (onInstantiate) as well as activity hooks (types onDeadline and OnCancel) 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. Installation guide

3.1. Installation

Nova Bonita M3 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.M3.zip
```

A new directory `bonita-4.0.M3` 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 (detailed command are given in following sections).

- build.properties

This file contains the J2EE properties required to deploy and leverage 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 in both in memory and persistence modes are provided

- 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 leverage Nova Bonita APIs from a client application. That includes the sample source required to leveraged 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.

- lib/

This directory contains the libraries used in Nova Bonita M3. Nova Bonita can be integrated in your application/IS in different ways (integrated in a web application, inside a rich client application, remotely deployed in a JEE application server...). Depending on your integration environment only some of those libraries will be required.

3.2. Standard vs Enterprise installation

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

3.2.1. Standard installation

To integrate Nova Bonita in your application you only need to add a couple of libraries to your environment. The number of libraries depends on the kind of services you use:

- Nova Bonita integration running in memory + basic identity and security services requires:

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

- Nova Bonita integration with workflow persistence + basic identity and security services requires previous libraries as well as hibernate 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
```

3.2.2. Enterprise installation

Move to the Nova Bonita installation directory and:

- call "ant ear" task. This will generate the bonita.ear file
- deploy this ear into your favorite EJB3 compliant container or JEE application server.

3.2.2.1. EasyBeans EJB3 Installation and deployment

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

- Download easybeans 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
- 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 environment 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 4. Configuration and Services

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

4.1. Services Container

The Process Virtual Machine technology includes a services container allowing the injection of services and objects that will be leveraged 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.mem).

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

- repository: data repository storing workflow processes, instances, activities... Both in-memory (class `org.ow2.bonita.repository.memory.MemoryRepository`) and db persistence (class `org.ow2.bonita.repository.db.DbRepository`) implementations are included in this milestone.
- 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. Both in-memory (class `org.ow2.bonita.services.record.impl.InMemoryJournal`) and db persistence (class `org.ow2.bonita.services.record.impl.DbJournal`) implementations are 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 archived 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
- finished-instance-handler: action to perform when a workflow instance is finished. Default implementation is to store data for a particular workflow instace and to remove data from journal (class `org.ow2.bonita.services.impl.DefaultActivityFactory`). Journal and History objects implementations (described previously) are leveraged by this class.

As explained before both memory based vs 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...

4.2. 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:

4.2.1. 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 stored through this persistence service. Workflow repository is the term used in Nova Bonita to store those entities.

4.2.2. 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, roles 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 leverage 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 Bonita M3 is 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.

4.2.3. 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 leveraged in Nova Bonita examples directory.

The current implementation of the security service allows you directly leverage default identity service to handle users authentication. Users must login before start calling the Bonita APIs. The current Nova Bonita Milestone 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 leveraged 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.)

4.2.4. 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 leveraged 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 responsibilities 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 any other service in Nova Bonita this module can be executed either in-memory or in a persistent environment.

4.2.5. 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.

Chapter 5. User guide

This chapter describes how to start playing with Nova Bonita M3. 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

5.1. 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 online [http://wiki.bonita.objectweb.org/xwiki/bin/download/Main/Documentation/Bonita_DevelopmentGuide.pdf] documentation 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.

5.2. 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 are becoming more and more stable and so it will be completely finished in the next Milestone. A complete review of changes/updates between v3 and Nova Nonita will be provided in the RC version.

5.2.1. Getting started with Bonita APIs

Nova Bonita APIs are divided into 5 different areas:

- DefinitionAPI: to get workflow definition data for packages, processes, activities, role mappers,
- RuntimeReadOnlyAPI: to get tasks (aka manual activities) for the authenticated user as well to get/ list variables in a particular or a set of workflow instances
- RuntimeAPI: to manage process, instance and task life cycle operations as well as to set/updates variables
- QueryAPI: to get recorded/runtime informations for packages, processes, instances, activities, tasks (will add support for dynamic queries in the future)
- Deployer: to deploy workflow processes into the engine. XPD L files and advanced entities such hooks, mappers and performer assignments can be deployed individually or in one shot

5.2.2. Nova Bonita APIs, playing with !

Nova Bonita M3 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 leverage it remotely thanks to the Workflow Session Beans APIs.

5.2.2.1. 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();

ReadOnlyAPIAccessorImpl accessorRO = new ReadOnlyAPIAccessorImpl();
RuntimeReadOnlyAPI runtimeReadOnlyAPI = accessorRO.getRuntimeAPI();
```

There are 2 different accessor interfaces available:

- **ReadOnlyAPIAccessor:** to get access to DefinitionAPI, RuntimeReadOnlyAPI, QueryAPI interfaces
- **APIAccessor:** to get access to RuntimeAPI, DeployerAPI interfaces

APIAccessorImpl and ReadOnlyAPIAccessorImpl classes are implementing the two interfaces. Instantiation of these classes gives access to the expected bonita API.

5.2.2.2. 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");
RuntimeReadOnlyAPI runtimeReadOnlyAPI = (RuntimeReadOnlyAPI) initialContext.lookup("RuntimeReadOnlyAPI");
```

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]

5.3. Running the example

The Nova Bonita package contains a complete workflow example. This sample is located under examples directory:

- The Bonita Simple Project Ever: a simple Approval Workflow process definition and execution

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

- The Approval Workflow sample can be run in a in-memory environment. That means, no persistence service will be used to store process definition or instances execution related data.

```
>ant aw-mem
```

- Same application (Approval Workflow) can be run using the 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 embeds HSQL database and uses it as a default database. You can easily change this default configuration and use your favorite database by modifying the hibernate.properties file located under "conf" directory.

```
>ant aw-db
```

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" with "bonita" 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 DeployerAPI), then creates a workflow instance and leverage 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 task:

```
>ant aw-eb
```

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

Nova Bonita Milestone 3 unit test suite can also be launched from the installation directory (this operation is a good way to check that Nova Bonita was properly configured)

- Unit test suite in-memory mode:

```
>ant test-mem
```

- Unit test suite in persistent mode:

```
>ant test-db
```

- Unit test in both in-memory and persistent modes:

```
>ant test-mem
```