The Danet Workflow Component

User Manual

Dr. Michael Lipp, Danet GmbH

For version 2.1.2

The Danet Workflow Component: User Manual

by Dr. Michael Lipp

Table of Contents

List of Figures

List of Tables

Introduction

This document describes how to integrate and use Danet's J2EE based workflow component. The Maintenance Guide [../../../../../../../api/de/danet/an/workflow/doc-files/maintenance-guide.html] describes the internal structures and design principles of the component.

What is Danet's workflow component?

Danet's workflow component is a J2EE based implementation of a workflow facility (workflow engine) as proposed by the Workflow Management Coalition (WfMC) and the Object Management Group (OMG).

The workflow component includes and is based on a set of JAVA interfaces that define an API for a workflow management facility. These "omgcore" interfaces follow OMG's *Workflow Management Facility Specification, V1.2* very closely, while making some changes to adapt the CORBA service to the established design practices for a Java API.

Danet's workflow component provides a collection of EJBs that implement the omgcore interfaces together with some additional interfaces for tasks not covered by the OMG specification. The EJBs are provided as a J2EE module that can be used in any J2EE compliant application server.

The workflow engine is designed as a J2EE component. It is intended to be integrated in an application that requires a workflow engine. As a proof of concept and test environment, we provide a demo application. The demo application consists of the workflow component and some additional modules with a portal based GUI. Combined they provide a small workflow system that may be used to get acquainted with the workflow engine. You can find more information about this demo application in Appendix C, *The demo applications* [337]

Required knowledge

This manual assumes that you are familiar with Sun's *Java™ 2 Platform Enterprise Edition Specification, v1.2*.

Structure of the remaining document

Chapter 1, *Integrating the workflow component* [1]
: describes all steps required to integrate the workflow component in an application.

Chapter 2, *States and state transitions of processes and activities* [15]
: describes all defined and supported states and state transitions of processes and activities.

Chapter 3, *Using the workflow component* [23]
: gives an inside view of the internal structure of Danet's workflow component.

Chapter 4, *Process definitions* [33]
: helps managing process definitions.

Chapter 5, *Tools* [43]
: lists the tools available in this workflow component. Tools are applications that can be started from an activity.

Chapter 6, *The sample resource assignment service* [65]
: shows how to develop or integrate a resource assignment service with the help of a simple example.

Chapter 9, *Known bugs and limitations* [79]
: lists the bugs and limitations of the workflow component currently known.

Appendix A, *The API documentation* [81]
describes the java user interface of the workflow components in javadoc format.

Appendix C, *The demo applications* [337]
gives an inside view of the provided demo applications.

Documentation conventions

The following conventions will be applied throughout this document:

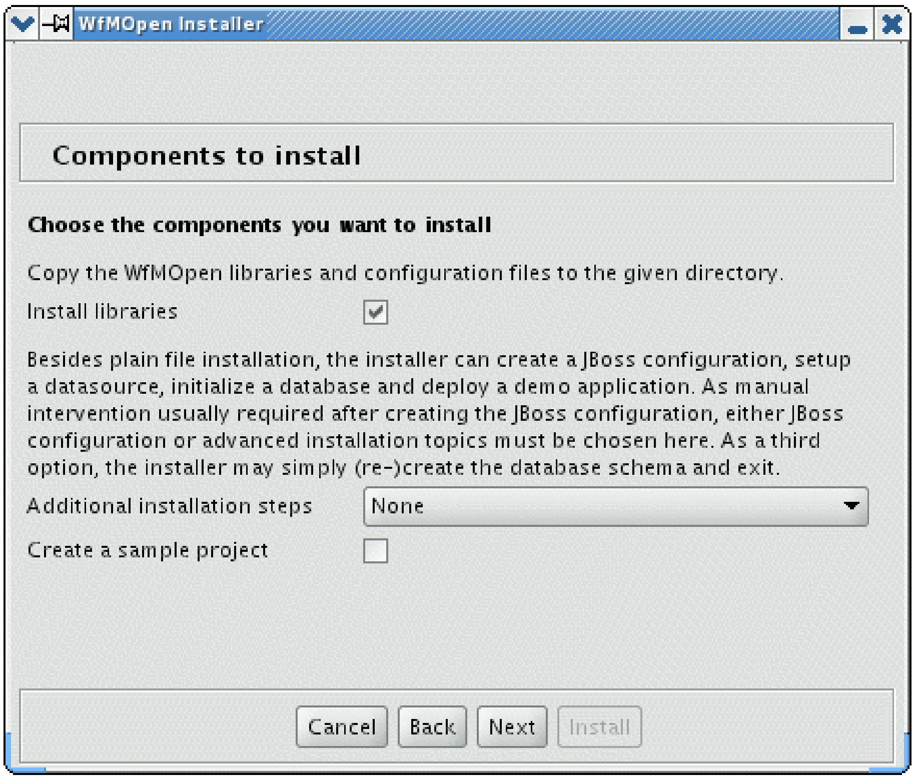
`name`	to state a common name or id as it is used within the workflow component.
`package`	for package name.
`class`	for class names.
`variable`	to describe a name of a variable.

Chapter 1. Integrating the workflow component

The workflow engine is designed as a component. Therefore the typical usage is to integrate it into your application. The following sections describe the steps required to do this. If you want to get acquainted with the workflow engine by installing a demo application, please proceed to Appendix C, *The demo applications* [337]. If you want to experiment with the API, step by step instructions how to build a basic sample client can be found in Section 3.3, “Sample client” [26]. If you want to integrate the workflow component with your application or understand its structure, continue reading here.

The binary distribution comes as an installer. In the simplest case, the installer only copies the libraries and documentation to your filesystem. It provides, however, additional operations to configure your application server and initialize a database. The installer is distributed as an executable jar. It is invoked as "java -jar `wfmopen-`*`n`*`.`*`m`*`-installer.jar`".

The figures below show the installer screens for copying the components to the filesystem.



Copy the WfMOpen libraries and configuration files to the given directory.

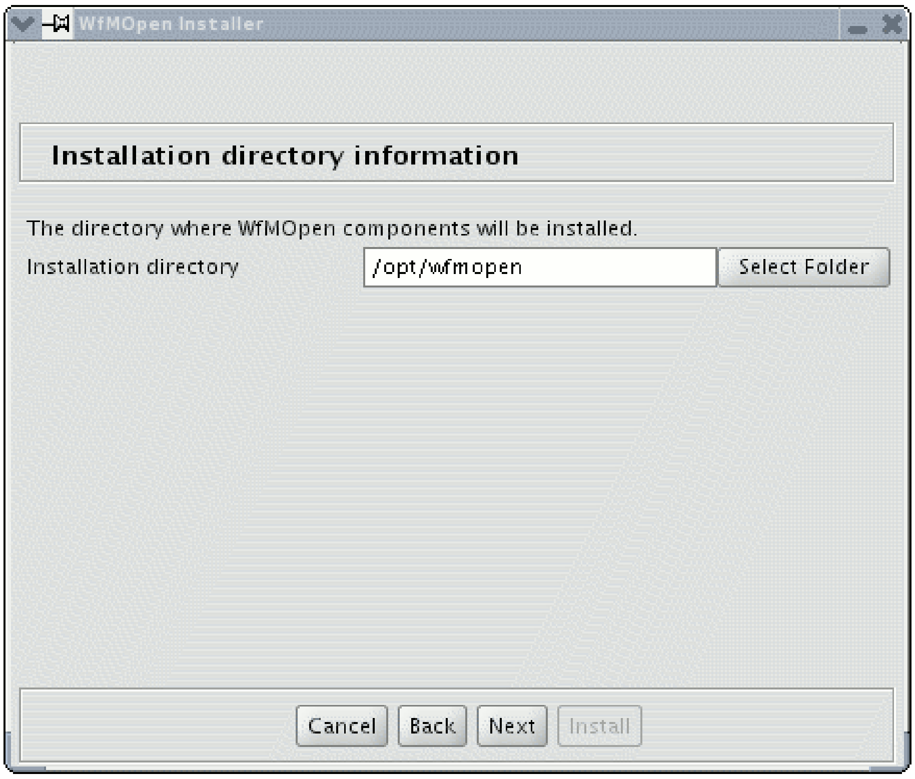
Install libraries ✓

Besides plain file installation, the installer can create a JBoss configuration, setup a datasource, initialize a database and deploy a demo application. As manual intervention usually required after creating the JBoss configuration, either JBoss configuration or advanced installation topics must be chosen here. As a third option, the installer may simply (re-)create the database schema and exit.

Additional installation steps: None

Create a sample project ☐

Figure 1.1. Copying option


WfMOpen Installer

Installation directory information

The directory where WfMOpen components will be installed.

Installation directory /opt/wfmopen

Figure 1.2. Copying destination

In the sections following, the term `$DIST` represents the top level directory of the binary distribution, i.e. the installation directory that you have specified in the dialog above.

1.1. Prerequisites

WfMOpen requires JDK 1.4.2 (see Section 1.3.7, “Additional libraries” [9] for more details). Older JDK versions back to 1.3 may work, tests have, however, only been made with JDK 1.4.2_15 and JDK 1.6.0_02.

The workflow component requires a database that supports distributed transactions. WfMOpen relies on operations on queues and on the database being made in a single transaction. This can only be accomplished by using distributed transactions[1]. If you are looking for an OpenSource solution, we recommend using MAXDB (formerly known as SAPDB).

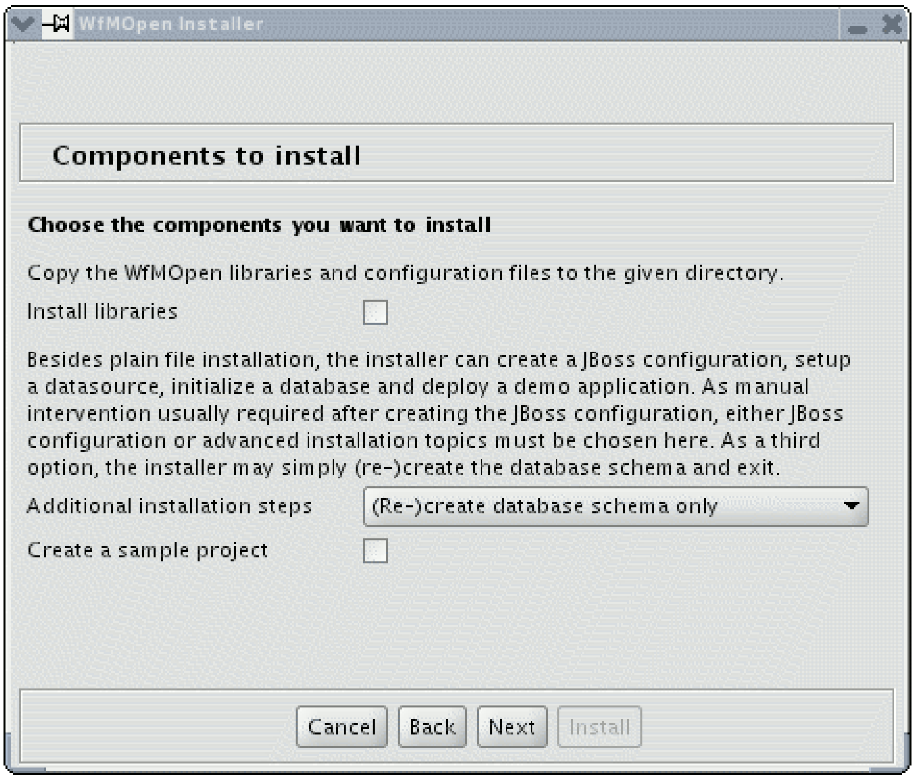
1.2. Preparing the database

The workflow component relies on the presence of a number of tables in a database. The table definitions are provided as file `$DIST/lib/wfdemo/database/`*`RDBMS-type`*`/create.sql`. The files contain SQL statements that can be executed by appropriate database tools.

If you cannot find scripts for your database, you can still use the generic table definition in `$DIST/lib/wfdemo/database/database-schema.xml`. It is suited to be processed by the Apache DDLUtils (see Apache DDLUtils site [http://db.apache.org/ddlutils])[2].

[1]Most applications servers still support distributed transactions if one participating resource does not support distributed transactions. If your application server does, you may use a database that does not support distributed transactions. Be prepared to get some warnings from your application server, though.

As an alternative, the schema can be created by the installer. In order to do this, choose the option "(Re-)create schema" as shown below.



Components to install

Choose the components you want to install

Copy the WfMOpen libraries and configuration files to the given directory.

Install libraries ☐

Besides plain file installation, the installer can create a JBoss configuration, setup a datasource, initialize a database and deploy a demo application. As manual intervention usually required after creating the JBoss configuration, either JBoss configuration or advanced installation topics must be chosen here. As a third option, the installer may simply (re-)create the database schema and exit.

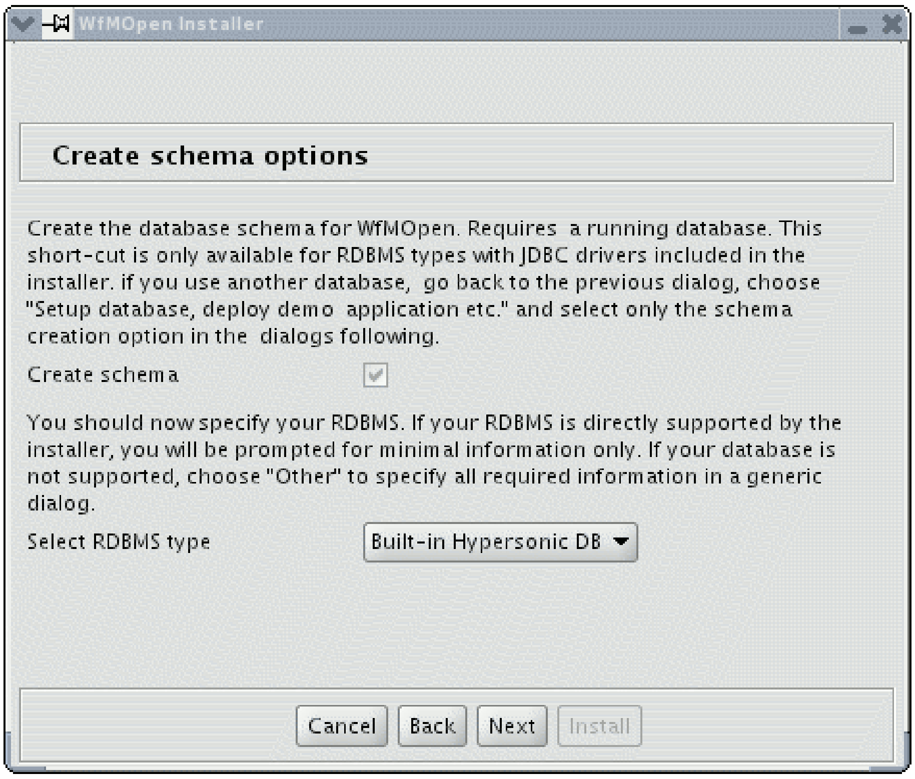
Additional installation steps: (Re-)create database schema only

Create a sample project ☐

Figure 1.3. Creating the schema with the installer (1)

You will now be prompted for an RBMS type.

[2]The provided RDBMS specific scripts were generated by processing this file with the DDLUtils for the specific RDBMSs.


WfMOpen Installer

Create schema options

Create the database schema for WfMOpen. Requires a running database. This short-cut is only available for RDBMS types with JDBC drivers included in the installer. if you use another database, go back to the previous dialog, choose "Setup database, deploy demo application etc." and select only the schema creation option in the dialogs following.

Create schema ✓

You should now specify your RDBMS. If your RDBMS is directly supported by the installer, you will be prompted for minimal information only. If your database is not supported, choose "Other" to specify all required information in a generic dialog.

Select RDBMS type: Built-in Hypersonic DB

Figure 1.4. Creating the schema with the installer (2)

If you select one of the specific types, you will only be prompted for database instance information and credentials on the next screen. If you select "Other", a generic dialog prompts you additionally for the driver class and a driver jar.

The installer then uses the DDLUtils to create the schema in your database. This will work for all databases that are supported by DDLUtils.

1.3. Application and client assembly

The workflow component software consists of several JAR files (modules and libraries) that have to be assembled into a J2EE application or have to be included in the classpath of a client. The figure shows an overview of the JAR files and their usage.

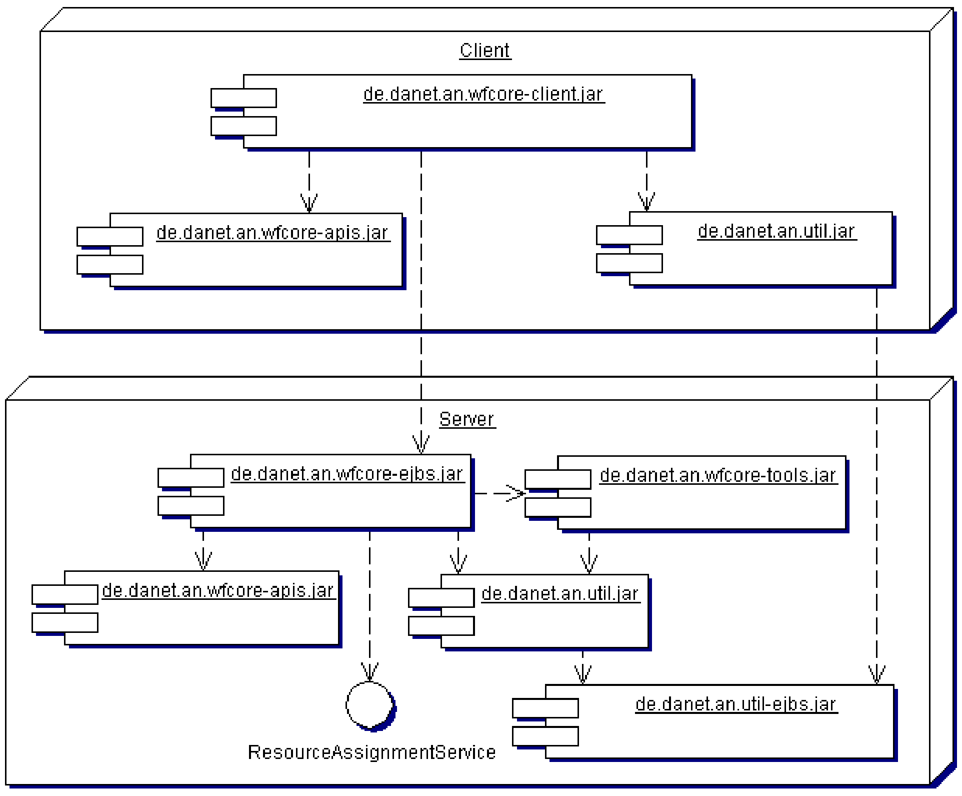

Client
de.danet.an.wfcore-client.jar
de.danet.an.wfcore-apis.jar
de.danet.an.util.jar
Server
de.danet.an.wfcore-ejbs.jar
de.danet.an.wfcore-tools.jar
de.danet.an.wfcore-apis.jar
de.danet.an.util.jar
de.danet.an.util-ejbs.jar
ResourceAssignmentService

Figure 1.5. Application assembly

The following sections describe each JAR file and its role in application assembly.

1.3.1. Workflow APIs

The specified workflow APIs (packages `de.danet.an.workflow.omgcore` and `de.danet.an.workflow.api`, see Section 3.2, “Client API” [24]) are packaged in a separate jar `de.danet.an.wfcore-apis.jar` located in the distribution directory `$DIST/lib/wfcore`. Only this JAR file should be used in the classpath when compiling clients. Consequently, it must also be included in the runtime classpath of a client.

1.3.2. Client JAR

The runtime-classes needed by a client that wants to use the workflow component are provided as `de.danet.an.wfcore-client.jar` in `$DIST/lib/wfcore`. This JAR file is not self contained, i.e. it relies on the presence of the workflow APIs (`de.danet.an.wfcore-apis.jar`), the Danet AN utility classes (`de.danet.an.util.jar`, see following section) and the additional libraries described in Section 1.3.7, “Additional libraries” [9]. In addition, the client needs some information about how to contact the server. See the description of the basic `WorkflowServiceFactory` (in Section A.2.52, “Class WorkflowServiceFactory” [250]), the description of the WfMOpen specific factory implementation (in de.danet.an.workflow.ejbs.client.StandardWorkflowServiceFactory [325]) and the remarks in Section 1.3.4, “Workflow module” [6] below.

1.3.3. The Danet AN utility library

All J2EE based software from the division AN uses a library with some common utility objects and components. This library is supplied as `de.danet.an.util.jar` (again located in the `$DIST/lib/wfcore`). Some components in this library (currently the logging service) need server side support. This support is provided by the corresponding EJB module described below.

The Danet AN utility library (package `de.danet.an.util`) is a general library used at our site. This library is packaged as two JAR files. By default, classes are in `de.danet.an.util.jar`. Some classes in the package (sub-package `de.danet.an.util.jsf`), however, are helper classes for JSF based portlets. These classes are are not included in `de.danet.an.util.jar`. They can be found in `de.danet.an.util-jsf.jar`. The reason for separating those classes is that they may not be put in the common classpath of a J2EE application. In order to avoid classpath problems, you have to put these JSF utility classes in the same directory as the JSF libraries (usually the `WEB-INF/lib` directory of your web module). Note that the classes from `de.danet.an.util.jsf` are only needed by the portlets that come with the workflow component. The deployment of the workflow EJBs requires `de.danet.an.util.jar` only.

1.3.4. Workflow module

The central (server side) workflow component is an EJB-type J2EE module. This module is not self contained, i.e. it relies on the presence of the workflow APIs, the Danet AN utility library and the additional libraries listed below. The module is located in the distribution directory `$DIST/lib/wfcore` as `de.danet.an.wfcore-ejbs.jar`.

The `manifest` file of `de.danet.an.wfcore-ejbs.jar` includes in its `Class-Path:` statement the entries `lib/de.danet.an.wfcore-apis.jar` and `lib/de.danet.an.util.jar`. Thus, the dependencies between the workflow module, the workflow APIs and the Danet AN utility library will be resolved automatically if `de.danet.an.util.jar` is put in the `lib/` directory of the enterprise archive (the `.ear`-file). An additional `Class-Path:` entry `lib/de.danet.an.wfcore-plugins.jar` serves as a hook for adding jars with tool agents (see Section 3.5, “Tool invocation SPI” [30]) to the class path. We recommend to put new tool agents or other libraries that are to be deployed with the engine core in extra add-on jar files. Then create a jar `de.danet.an.wfcore-plugins.jar` with only a `META-INF/MANIFEST.MF` that references your jars in its `Class-Path:` entry and add this (along with your add-on jars) to the EAR as `lib/de.danet.an.wfcore-plugins.jar`. The application server will then load `de.danet.an.wfcore-plugins.jar` (because it is referenced in `de.danet.an.wfcore-ejbs.jar`) and subsequently your jars with additional components.

1.3.4.1. Vendor specific deployment descriptors

The file `de.danet.an.wfcore-ejbs.jar` includes vendor specific deployment descriptors for some application servers. These deployment descriptors define, among other things, the binding of EJBs (EJB's home interfaces, to be precise) to global JNDI names[3]. As the workflow engine is intended to be integrated in different applications that may run on one application server concurrently, the JNDI names must be adapted to the specific application the engine is integrated with; else there will be conflicts between the applications. All deployment descriptors therefore use "generic" names for JNDI bindings such as "`ejb/@@@_JNDI_Name_Prefix_@@@SomeEJB`". When packing the workflow engine EJBs, the prefix "`@@@_JNDI_Name_Prefix_@@@`" should be replaced with some reasonable value (e.g. ""`@@@_JNDI_Name_Prefix_@@@ = de.danet.wfdemo.`").

Whereever the workflow module references the utility module (see Section 1.3.6, “The utility module” [8]) in the included vendor specific deployment descriptors, it uses the same symbolic names as the deployment descriptors of the utility module (again, see Section 1.3.6, “The utility module” [?]). These symbolic names must be adapted as well when assembling an application.

Note that "generic" names and "symbolic" names for resources appear only in vendor specific deployment descriptors. So supplying your own vendor specific deployment descriptors from scratch is an alternative integration strategy.

The workflow module includes Ant helper scripts for executing the required adaptions. See Section 1.4.1, “Preparing the modules” [9] for details.

[3]You should expect that only EJBs that are to be looked up by clients require binding to global JNDI names. It has turned out, however, that some application servers require explicit specification of global JNDI names for all EJBs, and others (even worse) choose default global JNDI names based on the EJB's names. Therefore, to avoid conflicts between different instances of the workflow engine on one application server, global JNDI names must be supplied in any case.

1.3.4.2. Accessing the engine from a client

One global JNDI name is of specific importance. It is the name of the `WorkflowEngineEJB`. This EJB is the only EJB looked up directly in JNDI by a workflow engine's client (remote interfaces of other EJBs are passed to the client directly or indirectly by methods of the `WorkflowEngineEJB`). Therefore, the global JNDI name of the `WorkflowEngineEJB`'s home interface must be made known to the client. The lookup of this home interface need not be made explicitly. Rather, it is encapsulated in the creation of a new workflow service by the `WorkflowServiceFactory` (see newWorkflowService() [252]). Thus the configuration of the JNDI name becomes a configuration issue of WfMOpen's implementation of the `WorkflowServiceFactory`. WfMOpen's implementation class and its configuration is described in detail in de.danet.an.workflow.ejbs.client.StandardWorkflowServiceFactory [325]. The preferred way to do the configuration[4] is to first create a file `de.danet.an.workflow-wfs.properties` with an entry `engine = ejb/@@@_JNDI_Name_Prefix_@@@WorkflowEngine` (applying the replacement described above) and then to add this file to the client JAR `de.danet.an.wfcore-client.jar`. This results in an "application specific" client library. Anywhere this client JAR is used, the resource `de.danet.an.workflow-wfs.properties` is found by the `StandardWorkflowServiceFactory` in the JAR and the proper workflow engine EJB is looked up.

Another value that may need replacement is the security domain. The JBoss vendor specific deployment descriptor defines the security domain as `java:/jaas/wfmopen`. This should be replaced by the application's domain.

1.3.4.3. Queues and Topics

In addition to the EJBs and the libraries, the workflow module requires three queues to be defined. By default, the JNDI names of these queues are "`queue/@@@_JNDI_Name_Prefix_@@@ApplicationInvocations`", "`queue/@@@_JNDI_Name_Prefix_@@@InternalEventQueue`" and "`topic/@@@_JNDI_Name_Prefix_@@@EventService`", with the latter being a topic queue. These names are referenced in the vendor specific deployment descriptors of the EJBs and must be adapted with the prefix replacement as described above.

You can find sample definitions of the required queues for JBoss in `$DIST/lib/wfcore/wfcore-destinations-service.xml`. Depending on your application server, you have to created these queues before the deployment of your application using e.g. some console application. Alternatively, you may usually trigger the creation and configuration of the queues by adding some vendor specific information to your enterprise archive (this mechanism is used in the demo applications, see Appendix C, *The demo applications* [337]).

1.3.4.4. Security roles

The workflow module as distributed uses in its deployment descriptors a predefined security role "WfMOpenAdmin" as only role and this role has all permissions[5].

1.3.5. The optional callback module

The actual work during process execution is done by tool agents (see Section 3.5, “Tool invocation

[4]The other supported methods also have some benefits; it depends on your usage scenario.
[5]Thus you have three possibilities to provide the required priviledges:

1. In the user management system used by your application server, make all users of the workflow engine members of the role "WfMOpenAdmin".
2. If your application server supports mapping between application roles and roles managed in your security domain (JBoss does not support this, e.g. Sun AS does) you can map "WfMOpenAdmin" to an existing role in your security domain, thus allowing all users in the existing role to access the workflow engine.
3. In all deployment descriptors, replace "WfMOpenAdmin" with the name of an exsting role in your user management system. This is typically done during deployment but may also be done during application assembly.

SPI” [30]). Several of these tool agents are included in WfMOpen (see Chapter 5,*Tools* [43]). Usually projects will add their own specific tool agents. In simple deployment scenarios, this can easily be done by adding jars with tool agent implementations as described above in Section 1.3.4, “Workflow module” [6].

In more complex deployment scenarios, there is a drawback to this approach. Consider an application with a core EAR that contains, among other components, WfMOpen. Other EARs provide additional services. In order to invoke these services from the workflow engine with specific tool agents, the agents would have to be bundled with the core EAR. But this would create a dependency on services and classes in the additional EAR and thus effectively a circular dependency. It would also be hard to have a consistent application server startup. If the engine is deployed first it may try to invoke a tool agent that requires services from an EAR that is not completely deployed yet.

In order to support such deployment scenarios, WfMOpen provides the callback module `de.danet.an.wfcore-callback-ejbs.jar` that may be bundled with the additional service EARs and executes tool agent invocations in the context of these service EARs. The callback module bundles two EJBs that are also contained in the core workflow module: an EJB that reads messages from the application invocation queue and an EJB (the `InvocationHelperEJB`) that invokes the tool agents. Tool agent invocations are forwarded to callback EJBs by specifying a `Handler` attribute for the application declaration in the process definition (see Section 4.2.5.1, “Extentions of Application Declaration” [35]). Configuring the callback module is described in Section 1.4.4, “Callback module deployment” [13].

1.3.6. The utility module

As described above, all J2EE based software from the division AN uses some common utility objects and components. While most of these elements can be simply supplied as a library, some components (currently the logging support) consist of a client and a server part. The J2EE server side elements are packaged in the distinct EJB-type J2EE module `de.danet.an.util-ejbs.jar` that is also located in the `$DIST/lib/wfcore` distribution directory. This module must be deployed along with the workflow module.

As the utility module is intended to be used in several applications, we have deliberately not defined defaults for some information needed during deployment. Providing defaults for these values would imply the risk that different applications inadvertently access each others utility EJBs. Instead of defaults we use symbolic names in the deployment descriptors that can reliably be replaced automatically.

All symbolic names are described below. The description should be sufficient to allow you to adapt the deployment descriptors for your application. For more information about these EJBs, see the maintenance manual.

The symbolic names used are:

`@@@_Utility-EJBs_UtilEJB_JNDI_Name_@@@`
: The utility EJB's global JNDI name. This EJB has a global JNDI name defined because it may be needed by stand-alone clients. A reasonable value if the utility module is used in a workflow application would be `de.danet.an.workflow.util-lib.Util`.

`@@@_Utility-EJBs_KeyGenEJB_JNDI_Name_@@@`
: The key generator EJB's global JNDI name. The key generator EJB has a global JNDI name defined because it may be needed by stand-alone clients. A reasonable value if the utility module is used in a workflow application would be `de.danet.an.workflow.util-lib.KeyGen`.

`@@@_Utility-EJBs_EJBSinkEJB_JNDI_Name_@@@`
: The log4j EJB-appender's global JNDI name. This EJB has a global JNDI name defined because it may be needed by stand-alone clients. A reasonable value if the utility module is used a workflow application would be `de.danet.an.workflow.util-lib.EJBSink`.

As distributed, the utility module's descriptors do not specify any security constraints. We do, however, strongly recommend adding such constraints as appropriate for an application's security domain when assembling the application[6]. A common configuration is to introduce the same security role as used for the workflow engine EJBs (i.e. "WfMOpenAdmin", see Section 1.3.4, “Workflow module” [6]) and allow this role to execute all methods.

The module includes support for executing the required adaptions. See Section 1.4.1, “Preparing the modules” [9] for details.

1.3.7. Additional libraries

As is usually the case with complex Java applications, the workflow component needs some third party libraries in addition to the standard JDK. Those libraries are:

- The commons-logging library [http://jakarta.apache.org/commons/logging.html] from the Apache Jakarta Project (for server).
- The log4j library [http://jakarta.apache.org/log4j] from the Apache Group (for client and server).
- The jdom library [http://www.jdom.org] (for server).
- The dom4j-full library [http://www.dom4j.org] (for server). This library also supplies the jaxen libraries [http://www.jaxen.org].
- The jsr173 (a.k.a StAX) library [http://www.jcp.org/aboutJava/communityprocess/first/jsr173/] (for server and client).
- The Rhino (JavaScript) library [http://www.mozilla.org/js/] (for server).
- The XMLBeans library [http://xmlbeans.apache.org/] (for server). This library is required by Rhino for the E4X support.
- The jelly core, xml-tags and jsl-tags libraries [http://jakarta.apache.org/commons/jelly/] (for server).
- The BeanUtils library [http://jakarta.apache.org/commons/beanutils/] (for server, needed by jelly).
- The axis library [http://ws.apache.org/axis/] (for server).

When you want to use WfMOpen with JDK/JRE 1.4, updated versions of the JRE XML packages (`org.w3c.dom`, `org.xml.sax`, `org.xml.sax.ext` and `org.xml.sax.helpers`) are needed. You you must therefore use the Endorsed Standards Override Mechanism of the JDK [http://java.sun.com/j2se/1.4.1/docs/guide/standards/index.html] to provide those newer versions, i.e. you have to set the system property `java.endorsed.dirs` to the directory `$DIST/lib/wfdemo/endorsed`[7]. For the versions of the XML libraries currently used see the CVS information in the `tools/endorsed` subdirectory.

1.4. Deploying the component

1.4.1. Preparing the modules

[6]The source distribution includes a stylesheet that, together with the appropriate invocations from ant scripts, adds security information to deployment descriptors.

[7]If you use JBoss, you do not have to set the endorsed library directory, because JBoss comes updated libraries in its `$JBOSS_HOME/lib/endorsed` directory. This directory is automatically set as endorsed directory in the JBoss run-scripts.

As has been described in the previous sections, the deployment descriptors of the EJB modules must be adapted to the application that integrates the workflow engine. In order to facilitate this task, the EJB JARs have in their subdirectory `assembly-resources` ant scripts with targets that may be called to perform the adaption. Of course, these scripts do not have to be used to adapt the EJBs, they are simply provided as useful helpers.

1.4.1.1. General helpers

File `assembly-resources/assembly-utils.xml` provides general purpose targets that can be used for any EJB JAR, not only for the workflow engine or utility EJBs.

`add-ejb-resource-ref`
: Add a resource reference to an EJB. Must be called with properties "src-ejb-jar=<the ejb jar to patch>", "target-ejb-jar=<the patch result>", "ejb-name=<the ejb that references the resource>", "ref-name=<the locigal name used by the EJB to reference the resource>", "jndi-name=<the (global) JNDI entry for the resource>", "res-auth" and "res-type" as to be put into the ejb-jar.xml. This target needs additional resources (DTDs and stylesheets) that must be provided in a directory specified with property "resource-dir=<directory>".

`add-remote-ejb-resource-ref`
: Add a reference to a remote EJB. Must be called with properties "src-ejb-jar=<the ejb jar to patch>", "target-ejb-jar=<the patch result>", "ejb-name=<the ejb that references the remote ejb>", "ref-name=<the locigal name used by the EJB to reference the resource>", "ref-type=<Session or Entity>", "home", "remote" and "link" as to be put into the ejb-jar.xml. This target needs additional resources (DTDs and stylesheets) that must be provided in a directory specified with property "resource-dir=<directory>".

`add-local-ejb-resource-ref`
: Add a resource reference to a local EJB. Must be called with properties "src-ejb-jar=<the ejb jar to patch>", "target-ejb-jar=<the patch result>", "ejb-name=<the ejb that references the resource>", "ref-name=<the locigal name used by the EJB to reference the resource>", "ref-type=<Session or Entity>", "local-home", "local" and "link" as to be put into the ejb-jar.xml. This target needs additional resources (DTDs and stylesheets) that must be provided in a directory specified with property "resource-dir=<directory>".

`adapt-ejb-jar`
: Modify the deployment descriptors of an ejb jar by replacing given strings with other strings as specified in a properties file. Must be called with properties "src-ejb-jar=<filename of distributed ejb jar>", "target-ejb-jar=<filename of adapted ejb jar>" and "props=<filename with properties>". If the special properties "security-roles", "security-identities" and "security-domain" are defined, security information will additionally be inserted into the deployment descriptors as specified by these properties. This target needs additional resources (DTDs and stylesheets) that must be provided in a directory specified with property "resource-dir=<directory>". This target also removes the sub-directory "assembly-resources" from the ejb jar if such a sub-directory exists. An additional properties file may be specified as property "token-replacements". The replacements specified in this file are applied to all deployment descriptors first. It is useful if you want to process the same EJB twice, using one replacement from your main properties file in the first step and another replacement in the second step, or to simply override replacements in the main properties file.

`add-ejb-env-entry`
: Add a resource reference to an EJB. Must be called with properties "src-ejb-jar=<the ejb jar to patch>", "target-ejb-jar=<the patch result>", "ejb-name=<the ejb that requires the entry>", and "env-entry-name=<name of environment entry>", "env-entry-type=<type of environment entry>" and "env-entry-value=<value of environment entry>". The target needs additional resources (DTDs, schemas and "assembly-utils.xml") which must be provided in a directory specified with property "resource-dir=<directory>".

`adapt-war`
: Modify the deployment descriptors of a war by replacing given strings with other strings as specified in a properties file. Must be called with properties "src-war=<filename of distributed

war>", "dest-war=<filename of adapted war>" and "props=<filename with properties>". An optional property "libs-to-remove" may be set to a comma separated list of files to remove from WEB-INF/lib (because they will be supplied by the containing EAR).

`add-env-entry-to-servlet`
: Modify the deployment descriptors of a war by adding an environment entry. Must be called with properties "src-war=<filename of distributed war>", "dest-war=<filename of adapted war>" and "env-entry-name=<name of environment entry>", "env-entry-type=<type of environment entry>" and "env-entry-value=<value of environment entry>". The target needs additional resources (DTDs, schemas and "assembly-utils.xml") which must be provided in a directory specified with property "resource-dir=<directory>".

1.4.1.2. Helpers for adapting the workflow engine EJBs

File `assembly-resources/assembly-helpers.xml` in the workflow engine EJB JAR uses the general purposes targets described above and adds to them, thus providing specific targets for the workflow engine EJBs.

`adapt-wfmopen-modules`
: Modify the deployment descriptors of WfMOpen modules "de.danet.an.wfcore-ejbs.jar", "de.danet.an.wfcore-callback-ejbs.jar", "de.danet.an.wfcore-client.jar" and "de.danet.an.workflow.wfxml.war". Must be called with properties "src-dir=<directory with distributed modules>", "target-dir=<directory for adapted modules>" and "props=<filename with properties>". The target needs additional resources (DTDs, schemas and "assembly-utils.xml") which must be provided in a directory specified with property "resource-dir=<directory>". An optional property "libs-to-remove" may be set to a comma separated list of files to remove from WEB-INF/lib of the WARs (because they will be supplied by the containing EAR).

`prepare-wfmopen-callback-module`
: Modify the deployment descriptors of WfMOpen module "de.danet.an.wfcore-callback-ejbs.jar". Must be called with properties "src-dir=<directory with distributed modules>", "target-ejb-jar=<name of adapted ejb-jar>" and "props=<properties file>". These properties are the same properties as used for "adapt-wfmopen-modules". In addition, they must include a value for the key "@@@_JNDI_Callback_Name_Prefix_@@@" used to derive the global JNDI name of the callback EJBs. These must, of course, be unique for every deployment of the callback EJBs. (Note that you can use the property "token-replacements=<properties file> to simply "add" this property to the general WfMOpen properties file, see general helper "adapt-ejb-jar".) Property "handler" specifies which tool agent invocations are handled by the adapted module, i.e. its value must correspond to the value used for the "Handler" attribute of the application definition in the process definition. The target needs additional resources (DTDs, schemas and "assembly-utils.xml") which must be provided in a directory specified with property "resource-dir=<directory>".

`adapt-client-jar`
: This target is called by "adapt-wfmopen-modules". It configures the StandardWorkflowEngineFactory by adding a file de.danet.an.workflow-wfs.properties with an appropriate entry to the client jar. Must be called with properties "src-client-jar=<filename of distributed client jar>", "target-client-jar=<filename of adapted client jar>" and "props=<filename with properties>".

`make-deployment-service`
: Replace strings with other strings in a deployment descriptor as specified by a properties file and wraps the result as a SAR. This target is provided because JBoss documentation does not explicitly state that you can put arbitrary deployment descriptors (e.g. for creating a data source) in an EAR. You can, however, wrap them in a SAR and put this SAR in the EAR (and have it deployed by an entry in "jboss-app.xml". This target must be called with properties "deployment-xml=<the deployment description>", "dest-file=<filename of the SAR>" and "props=<filename with properties>".

1.4.1.3. Helpers for adapting the utility EJBs

File `assembly-resources/assembly-helpers.xml` in the utility EJB JAR uses the general purposes targets described above and adds to them, thus providing specific targets for the utility EJBs.

`adapt-util-ejbs`
: Modify the deployment descriptors of utility ejbs by replacing given strings with other strings as specified in a properties file and adding security information retrieved from the same properties file. Must be called with properties "src-ejb-jar=<filename of distributed ejb jar>", "target-ejb-jar=<filename of adapted ejb jar>" and "props=<filename with properties>". The target needs additional resources (DTDs, schemas and "assembly-utils.xml") which must be provided in a directory specified with property "resource-dir=<directory>".

`adapt-util-modules`
: Modify the deployment descriptors of the utility library modules. Must be called with properties "src-dir=<directory with distributed modules>", "target-dir=<directory for adapted modules>" and "props=<filename with properties>". The target needs additional resources (DTDs, schemas and "assembly-utils.xml") which must be provided in a directory specified with property "resource-dir=<directory>".

1.4.1.4. Sample properties file

A sample properties file for use with the above ant scripts is shown below.

```
# # Information needed for
        application assembly #

# Adapt utility EJBs' JNDI names. They have no "reasonable" defaults,
# so we have to specify very much. Note that these JNDI names are
# define separately as the utility EJBs may be shared between
# different applications. Note that "ejb/" will be prepended to the names
# specified below automatically and the entries for local home interfaces
# have "Local" appended to the name.
@@@_Utility-EJBs_UtilEJB_JNDI_Name_@@@ = \
          de.danet.an.wfdemo.util-lib.Util
@@@_Utility-EJBs_KeyGenEJB_JNDI_Name_@@@ = \
          de.danet.an.wfdemo.util-lib.KeyGen
@@@_Utility-EJBs_KeyGenLockEJB_JNDI_Name_@@@ = \
          de.danet.an.wfdemo.util-lib.KeyGenLock
@@@_Utility-EJBs_EJBSinkEJB_JNDI_Name_@@@ = \
          de.danet.an.wfdemo.util-lib.EJBSink
# Adapt utility EJB's data source reference
java\:/DefaultDS = java:/WfMOpenDS

# Adapt other EJBs' JNDI names. Most application servers require that
# you specify JNDI names although all references to the EJB are via
# links within the application. We simply define a prefix for those.
# This prefix is also used for JNDI names of other resources. Note that
# "ejb/" will be prepended to the prefix specified below when used to
# derive JNDI names for EJBs.
@@@_JNDI_Name_Prefix_@@@ = de.danet.an.wfdemo.

util-ejbs-security-domain = java:/jaas/wfdemo
util-ejbs-security-roles = \
  WfMOpenAdmin: \
    KeyGen*KeyGenLock*EJBSink*Util;
util-ejbs-security-identities = TimeoutHandler:WfMOpenAdmin
util-ejbs-security-principals = TimeoutHandler:WfMOpenAdmin_Principal

# Adapt workflow engine (note that the engine uses the util EJBs,
# necessary replacements are already covered by the properties above).
java\:/jaas/wfmopen = java:/jaas/wfdemo
```

1.4.2. Basic deployment

The workflow EJBs can be deployed in the usual way defined by J2EE:

```
<application>
  <display-name>My application using WfMOpen</display-name>
  <module>
    <ejb>de.danet.an.util-ejbs.jar</ejb>
  </module>
  <module>
    <ejb>de.danet.an.wfcore-ejbs.jar</ejb>
  </module>

  <module>
    <ejb>some.user.my.application.jar</ejb>
  </module>
</application>
```

Make sure that you have included all libraries needed by the workflow EJBs in the `lib/` directory of your J2EE application as described in Section 1.3.4, “Workflow module” [6]. This is a usable configuration if you only define workflow processes without any resources (i.e. only automatically executed activities). If resource assignment to activities is needed, some additional services have to be supplied as described below.

If you combine the workflow engine EJBs and servlets (packed as WARs) in a single EAR, it is advisable to have libraries used by both components only in the EAR (i.e. not in the WARs' `WEB-INF/lib` directory. In such a configuration, the libraries must explicitly be deployed as Java modules in `application.xml`. Note that this and other packaging aspects are not WfMOpen specific. Rather they apply to J2EE applications in general and details are bexond the scope of this manual. See one of the many books about J2EE for further informations.

1.4.3. Additional services

An issue not necessarily handled by a workflow core service is the issue of resource assignment. The OMG specification explicitly leaves this topic to a to-be-defined resource assignment facility. We have therefore chosen to introduce a simple interface to a resource assignment service in our design. This interface is described in detail in the package `de.danet.an.workflow.spis.ras` [../../spis/ras/package-summary.html].

In order to keep this interface easily implementable by any kind of architecture, access to this service is not based on JNDI and an EJB home interface. Access rather follows the established generic pattern to create a service using a factory class. In order to use the workflow component, an implementation of this service must be provided.

The workflow component includes a sample implementation of a resource assignment service. If you want to use this implementation, some additional configuration issues arise which are discussed in Section 6.1, “The sample assignment service” [65].

1.4.4. Callback module deployment

The callback module should be adapted by using the target "prepare-wfmopen-callback-module" as described in Section 1.4.1.2, “Helpers for adapting the workflow engine EJBs” [11]. This requires basically the same properties as adapting the core modules, i.e. a basic knowledge about the workflow engine configuration. In order to distinguish the callback EJBs from the ones used in the engine core and in other callback module configurations, each deployment must specify an individual additional property `@@@_JNDI_Callback_Name_Prefix_@@@`. This is used to derive the global JNDI name of the callback EJBs. To avoid a complete properties file for every callback module adaption, this property may be put in its own properties file. This properties file is then passed to the invocation of the ant target as property `token-replacements`. "Token-replacements" is a mechanism supported by the ant helpers that allows to override (and therefore also add) individual properties in the general properties file.

Also required is the specification which tool agent invocations are to be handled by the callback module. This is determined by the ant property `handler`. Its value must correspond with the value of the attribute `Handler` used for the application declaration in the process definition (see Section 4.2.5, “Defined extensions” [35]).

Chapter 2. States and state transitions of processes and activities

Dr. Christian Weidauer, Danet GmbH
Dr. Michael Lipp, Danet GmbH

2.1. States of processes and activities

Both `WfProcess` and `WfActivity` objects can assume the same six different states. The following table itemizes these states and explains their meaning for processes and activities.

Table 2.1. Meaning of execution object states

state \ object	process	activity
`open.not_running.not_started`	After creation the process is active and ready to be initialized and started.	After creation the activity is active and ready to be initialized and started when its start condition is fulfilled.
`open.running`	The process is active and executing in the workflow. The process may start new activities.	The activity is executing in the workflow. The activity is invoking the implementing tools or a sub process was started and is running.
`open.not_running.suspended`	The process is active and quiescent, but ready to execute. Its execution is temporarily paused, so that no further activities depending on this process may be started.	The execution of the activity is temporarily paused. If the activity is implemented as a sub process the process is also suspended.
`closed.aborted`	Indicates that the enactment of the process has been aborted before normal completion. The only assumption on the state of activities depending on this process is that no activity is running. Note, however, that tools may still be running if they do not support asynchronous termination.	Indicates that the enactment of the activity has been aborted before normal completion. No assumptions on the state of sub processes and tools depending on this activity are made when it enters this state.
`closed.terminated`	Indicates that enactment of the process was stopped before normal completion. It is assumed that all activities depending on this process have never been started or are completed or are terminated when it enters this state.	Indicates that enactment of the activity was stopped before normal completion. It is assumed that all sub processes and tools depending on this activity are either completed or are terminated when it enters this state.
`closed.completed`	When a process has finished its task normally in the overall workflow process it enters the completed state. It is assumed that all activities associated with the process are completed or not started when it enters this state. Further, the combination of the conditions on the incoming transitions of activities that are not started must evaluate to false.	When an activity has finished its task normally it enters the completed state. It is assumed that all tools or sub processes associated with the activity are completed when it enters this state.

2.2. State transitions of processes and activities

During the life cycle of processes and activities their states change. The following figure shows supported state transitions of `WfProcess` and `WfActivity` objects.

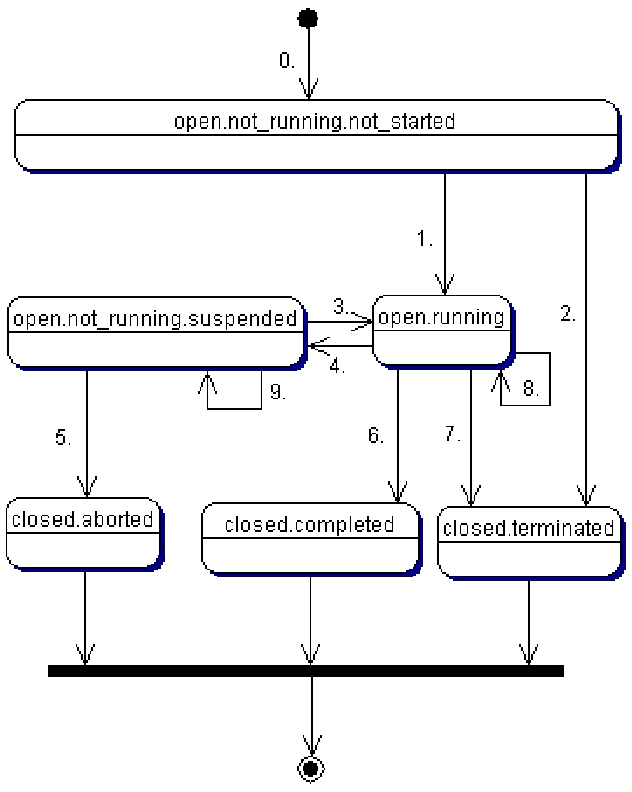


Figure 2.1. State transitions of execution objects

As can be seen in the figure, both the `WfProcess` and `WfActivity` objects start in state `open.not_running.not_started` when they are created and finish in one of three different closed states.

2.3. Triggers of state transitions

State transitions are triggered by the client API and the engine or can be an reaction to exceptions occuring when invoking tools.

The following table shows how calling a client API operation changes the state of an execution object (process or activity). The numbers used in the following tables refer to the state transitions in the above state diagram.

Note that triggering a state change will cause the workflow engine to either make the intended transition or to throw one of the exceptions declared in the API (see Section A.1.36, “Interface WfExecutionObject” [111], Section A.1.42, “Interface WfProcess” [132] and Section A.1.29, “Interface WfActivity” [98]). As a result of the transition made, however, the object's state may further be updated by the engine or other clients running in different threads. Thus querying the state of an execution object immediately after successfully triggering a state change may return a state different from the target state of the transition in the table.

Table 2.2. State transitions triggered by client API calls

object \ method	**start**	**suspend**	**resume**	**terminate**	**abort**	**complete**
WfProcess	1	4	3	2, 7	5	n. a.
WfActivity	n. a.	4	3	2, 7	5	8, 9

The "unobservable" (from the client's point of view) state transitions caused by a call to `complete` may seem strange at first. See the method's API documentation (complete() [99]) for further information about the behaviour.

Usually, state transitions of an execution object will cause further state transitions of other execution objects. There are six cases to distinguish, which are shown in the following table. The head row of the table defines the state transition of the **triggering object**. This transition may influence the triggered object (**triggering object** -> triggered object). Its potential transitions are listed in the table elements. E. g. table element (1(a)) in the second table row (**WfProcess (parent)** -> WfActivity) and fifth column (3) means: The state transition 3 of a parent process may cause a state transition 1 of its activities if the start condition of the activities is fulfilled.

Special attention has to be paid to state transitions of activities with manual start and/or finish mode. These modes are implemented by setting the activity object to state `open.not_running.suspended` instead of state `open.running` (manual start mode) or state `closed.completed` (manual finish mode). When resuming an activity object (transition 3) the engine will therefore take into account why the activity was suspended and proceed accordingly.

Table 2.3. State transitions triggered by the engine

object \ transition	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
WfProcess (parent) -> WfActivity	0	1(a)		1(a)		5, 7, 4(j)	n. a.	7	n. a.	n. a.
WfProcess (sub) -> WfActivity	n. a.	n. a.	n. a.			5	8	7	n. a.	n. a.
WfActivity -> WfProcess (parent)	n. a.	n. a.				4	6(d), 7(e)	7(c)		
WfActivity -> WfProcess (sub)		0 & 1(k)	n. a.	3	4	5	n. a.	7		n. a.
WfProcess -> WfProcess (itself)					5(b)				n. a.	n. a.
WfActivity -> WfActivity (itself)		4(f)		4(g), 6(h)	5(b)				4(g), 6(i)	

(a) If the start conditions of the activity evaluate to true.

(b) If the triggering transition 4 (to `open.not_running.suspended`) was caused by an activity state transition 5 (to `closed.aborted`). Occurs if an activity aborts (causing everything else to abort) and the process (or an activity of the process) is in state `open.running`. Then `closed.aborted` can only be reached via an intermediate `open.not_running.suspended`.

(c) If there are no more running activities or activities with start conditions evaluating to true.

(d) If there are no more running activities and no activities with start conditions evaluating to true and all closed activities have state `closed.completed`.

(e) If there are no more running activities and no activities with start conditions evaluating to true and one or more closed activities have a state other than `closed.completed`.

(f) If start mode is manual.

(g) If the activity's finish mode is manual and no work remains to be done.

(h) If activity was suspended because finish mode is manual.

(i) If the activity's finish mode is automatic and no work remains to be done.

(j) If the activity cannot be terminated.

(k) If the start mode of the activity is manual, creation and start of the sub-process will be delayed until the activity has been resumed.

Besides the triggering by the client or the engine, there are three more possible causes for state transitions:

- A failure to invoke a tool causes state transition 7 of the invoking `WfActivity` object.
- The detection of a loop causes the activities on the path to be reset, i.e. their state changes internally from `closed...` to `open.not_running.not_started` (see Section 4.3.4, “Loops” [?] for details about loops). This change of state is not recorded as an audit event, because the repeated execution of an activity is interpreted as several instances of the activity being executed one after the other. Thus the first activity instance is started, runs and completes, then the second activity instance is started, runs and completes. There is no observable change from `closed...` to `open.not_running.not_started`.
- If the preliminary choice of an activity in a deferred choice is revoked (because another activity is chosen), its state changes from `open.running` to `open.not_running.not_started` (see Section 4.3.5, “Deferred choice” [42] for details about the deferred choice). This change of state is recorded as an audit event, as it is part of the activity's execution sequence.

2.4. Exceptions

Besides events triggered by a client, an activity may also receive exceptions. Exceptions are defined in the XPDL specification as a general mechanism to signal an exceptional condition to an activity. The only specifically defined exceptions are those generated by the arrival of a deadline that has been defined for an activity. WfMOpen additionally supports handling exceptions from tool invocations, see Section 3.5.4, “Exception handling” [31] for details.

As far as state is concerned, exceptions are basically handled like the final complete call[1]. Any running tool is terminated and tools not yet invoked are skipped.

It may at first be surprising that the activity reaches `closed.completed` when an exception occurs. However, any other resulting state of an activity would have a strange effect on the final process state, as it would prohibit the process from reaching the completed state (see Section 2.1, “States of processes and activities” [15])[2]. From a process' point of view everything is all right. A deadline may have occurred and some special transitions may have been taken, but the process has still performed within its specification.

Things look a bit different, though, from the perspective of a subprocess started by an activity that receives an exception. This process is forcefully terminated by its requester (the activity that has received the exception). It therefore assumes the state terminated or aborted. In contrary to the normal state transitions, this does not cause the requesting activity to assume the same state as the activity is

[1]The impact on transitions is quite different, of course.
[2]We could have relaxed this dependency, allowing the process to reach the completed state even if an activity is terminated or aborted, provided this state was reached due to an exception. But then the state definition wouldn't be OMG compliant any more.

set to state completed by the exception processing.

Under normal circumstances, the occurrence of an exception should be derived from the transitions taken in a process. However, as a help for automated evaluations, WfMOpen has introduced the substates `closed.completed.normal` and `closed.completed.abandoned` with the latter being assigned to an activity that has been completed due to an exception.

2.5. Suspended state and deadlines

XPDL introduces deadlines, OMG knows about a "suspended" state, but neither knows about the other. This leaves the question of how the suspended state affects deadlines. We have defined the semantics as follows.

If an activity is set to state suspended, deadlines are suspended as well, i.e. no exceptions from deadlines will occur[3]. Note that "if an activity is set to state suspended" is not the same as "if an activity is in state suspended". Activities may also reach the state suspended if they have manual start or finish mode. In these cases deadlines are not suspended[4].

If the deadline has been defined as an absolute date time value (see Section 4.2.4, “Deadlines” [34] then setting an activity in the suspended state will not prolong the deadline. If the deadline is reached while the activity is suspended, the related exception will be delivered immediately when the activity is resumed.

If the deadline has been defined as a duration then setting an activity in the suspended state will prolong the deadline, i.e. the deadline will be delayed by the accumulated time that the activity has spent in the suspended state.

Dynamic deadline conditions, i.e. conditions that depend on process relevant data, will be re-evaluated every time the activity is resumed. The advantage of this behavior is that it effectively offers you a choice. If you want your expiration date to remain, you simply use only process relevant data that does not change. The disadvantage is that in order to have invariant process relevant data, you may have to make copies of several items (or, maybe better, calculate your deadline in an independent expression and keep it in some data field).

Maybe the biggest trap in re-evaluation is that the simple conversion of a duration to an absolute date "`new Date((new Date()).getTime() + duration)`" will not work. While this yields a date, which is not prolonged by the suspend state, it will be re-evaluated when the activity is resumed. Thus the deadline will probably be delayed even more than by using the duration in the first place. There is no solution to this other than saving the start time of the activity in a data item (e.g. by calling the JavaScript tool as first tool of the activity).

2.6. Debugging workflows

Workflows may be run in debugging mode. In this mode, the activities assume some additional, intermediate states. Note that the associated state transitions are neither recorded nor distributed as state change events. The debug mode has been designed to run the workflow in a way that resembles the normal execution as closely as possible. The debug mode is not a simulation. It can best be compared to running a program in a debugger, i.e. the workflow is really executed. There are, however, predefined break points and the invocation of tools may be simulated instead of actually performed.

[3]There has been some disussion if this is the proper behaviour. The reasoning is that setting the activity in the suspend state is nothing that occurs during normal workflow execution. It is a deliberate management action. This action would become farcial if it could be circumvented by a deadline.
[4]We assume that a process designer who defines start or finish mode manual together with a deadline wants that deadline to be executed to e.g. notify the administrator.

2.6.1. Enabling debug mode

Debugging mode is enabled for a specific process by calling `setDebugEnabled(true)` on the process after creating, but before starting the process (see setDebugEnabled(boolean) [209]).

Debugging can also be enabled for a process type by default using the XPDL extension mechanism as described in Section 4.2.5.2, “Extensions on Package and Process Level” [36].

2.6.2. Effect on state changes

When debug mode is enabled, the execution of activities breaks before the invocation of a tool and before completion of an activity.

Immediately before invoking a tool, the activity's state changes to `open.running.debug.invoking`. To continue the execution, the activity's state must be changed to either `open.running` or to `open.running.debug.skipping`. In the former case, the tool will be invoked as in non-debugging mode. In the latter case, tool invocation will be skipped and the activity will either assume the state `open.running.debug.invoking` again (if there are more tools to be executed by the activity) or the state `open.running.debug.completing` (if the invocation of the last tools has been skipped, see below). Of course, the process data must subsequently be modified manually to reflect the changes that would have been made by the tool invocation.

Before the activity is eventually closed, it enters one of the states `open.running.debug.terminating`, `...aborting` or `...completing` depending on the closed-state that the activity will assume. To continue execution and cause the activity to assume it proper closed state, the activity's state must be changed to `open.running`.

2.6.3. Effect on exceptions

In debugging mode, exceptions thrown by tools (i.e. calls to `abandon`, see abandon(java.lang.String) [146] will cause the activity to assume the state `open.running.debug.abandoning`. To continue the execution, it is not sufficient to provide the possibility to change the state to `open.running` as described above for the "normal" transitions to the closed state. There may be several exceptional transitions defined, and it must be possible to specify which exception should be used, else it wouldn't be possible to test the different paths during debugging.

In order to avoid polluting the interface with methods that are only used for debugging, we have defined a special sequence of method invocations that cause the execution to continue (i.e. the state will change to `closed.completed.abandoned` and the execution of subsequent activities will be triggered as specified by the transitions). You first change the state to `open.running.debug.awaiting_exception` and then call `abandon` (see abandon(java.lang.String) [146]) with the name of the exception to be processed. The preceding state change causes `abandon(String)` to behave differently from normal operation, effectively continuing execution as described above.

In order to avoid any ambiguities, we strongly recommend to execute the two operations (state change and abandoning) in a single transaction. This may be achieved by using a user transaction as described in the J2EE specification or by using the method invocation batch (see Section A.2.31, “Class MethodInvocationBatch” [193]).

```
WorkflowService wfs = ...;
MethodInvocationBatch mib = new MethodInvocationBatch(true);
mib.addInvocation (activity, "changeState",
                   new String[] {"java.lang.String"},
                   new Object[] {"open.running.debug.awaiting_exception"});
mib.addInvocation (activity, "abandon",
                   new String[] {"java.lang.String"},
                   new Object[] {exception});
MethodInvocationBatch.Result mir
    = (MethodInvocationBatch.Result)wfs.executeBatch(mib);
```

2.6.4. Effect on deadlines

Deadlines will be be started in debugging mode just as in non-debugging mode. However, nothing happens when deadlines are reached. The state of deadlines may be monitored by calling deadlines() [148]. To cause the process to proceed as if a synchronous deadline had been reached, you must first call `abandon(String)`, which causes the activity to assume the state `open.running.debug.abandoning`, and then continue execution as described for exceptions in general above (see Section 2.6.3, “Effect on exceptions” [20]). This may, of course, be done before the deadline has been reached, i.e. the effect of a deadline may be tested without actually waiting for the associated time (which may be quite long) to elapse.

In order to simulate an asynchronous deadline, the state of an activity must first be changed to `open.running.debug.forwarding_exception`. This causes a subsequent call to `abandon` to mimic the behavior of an asynchronous deadline, i.e. transitions will be triggered as specified for the given exception. The activity will automatically resume the state that it had before the change to `open.running.debug.forwarding_exception`. Of course, this sequence of method calls should also be executed in a single transaction, as described above.

Chapter 3. Using the workflow component

3.1. Component structure

The following figure shows the overall structure of Danet's workflow component.

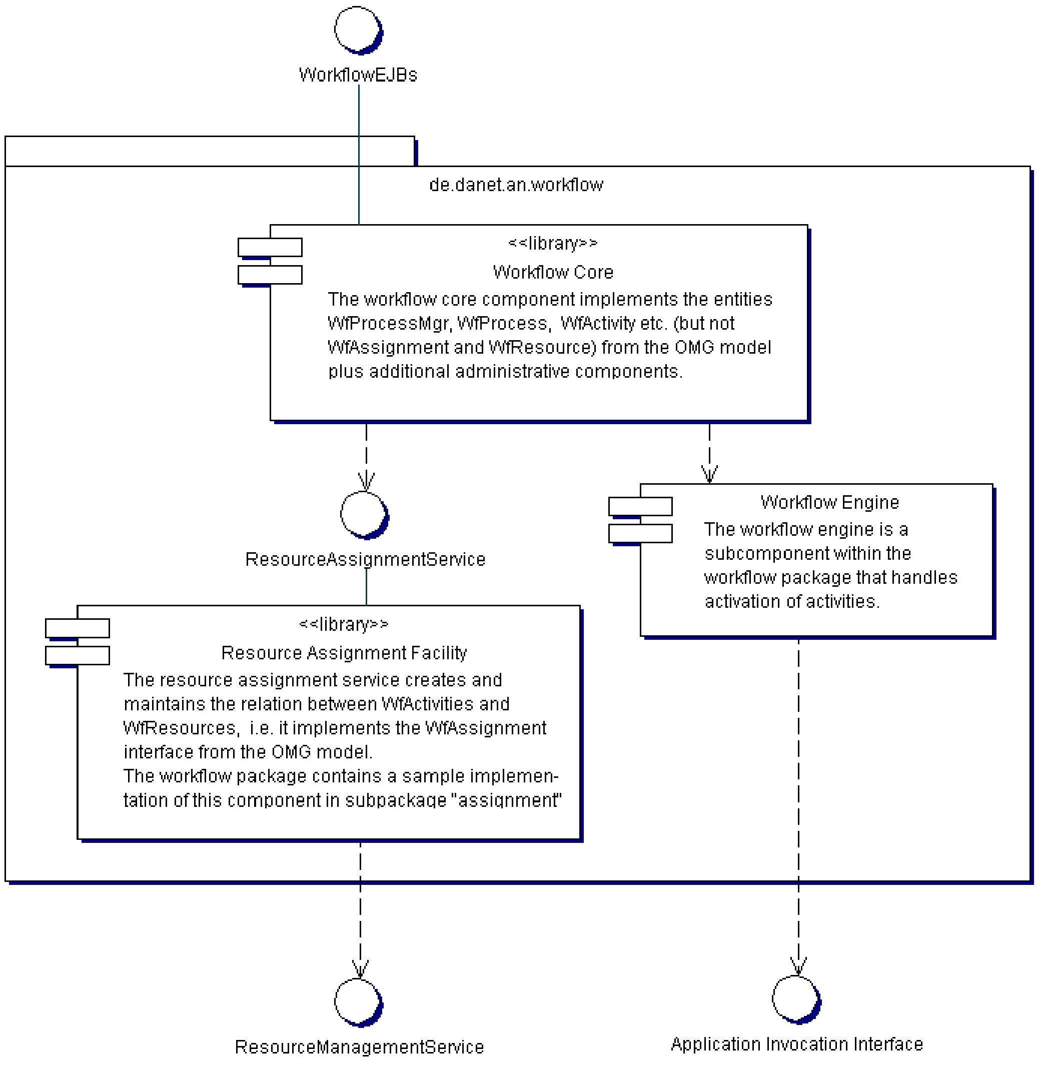

WorkflowEJBs
de.danet.an.workflow
<<library>>
Workflow Core
The workflow core component implements the entities WfProcessMgr, WfProcess, WfActivity etc. (but not WfAssignment and WfResource) from the OMG model plus additional administrative components.
ResourceAssignmentService
Workflow Engine
The workflow engine is a subcomponent within the workflow package that handles activation of activities.
<<library>>
Resource Assignment Facility
The resource assignment service creates and maintains the relation between WfActivities and WfResources, i.e. it implements the WfAssignment interface from the OMG model.
The workflow package contains a sample implementation of this component in subpackage "assignment"
ResourceManagementService
Application Invocation Interface

Figure 3.1. Structure of the workflow component

As can be seen in the figure, the workflow core component provides an EJB based client API. Helper classes on the client side hide most of the EJB invocation details, thus enabling the usage of the component without constantly keeping EJB details in mind.

The workflow component relies on the availability of a resource assignment service. As this service must be provided to the workflow component, it can be thought of as an additional API, or SPI to be precise, of the workflow component.

The distribution includes a sample implementation of such a resource assignment service which is described in detail in Chapter 6, *The sample resource assignment service* [65].

Finally the workflow package defines an API (SPI) used to invoke applications that perform activities.

The following sections describe each API in detail.

3.2. Client API

A standard workflow Java API has not been defined yet. Still, we did not want to provide users of our workflow component with a completely proprietary API. Therefore, we have taken a two stage approach.

3.2.1. Adapted OMG interfaces

As a first step, we have adapted the API specified by OMG's *Workflow Management Facility Specification, V1.2* to the Java domain. The result of this adaptation can be found in the package description of `de.danet.an.workflow.omgcore` (see de.danet.an.workflow.omgcore [81] or JavaDoc [../../omgcore/package-summary.html#package_description]). The package defines the core workflow classes such as `WfProcess`, `WfActivity` etc. A detailed description of the process used to derive the Java interfaces from the OMG specification can be found in the package description.

The OMG core workflow interfaces define a workflow model as shown in the following diagram.

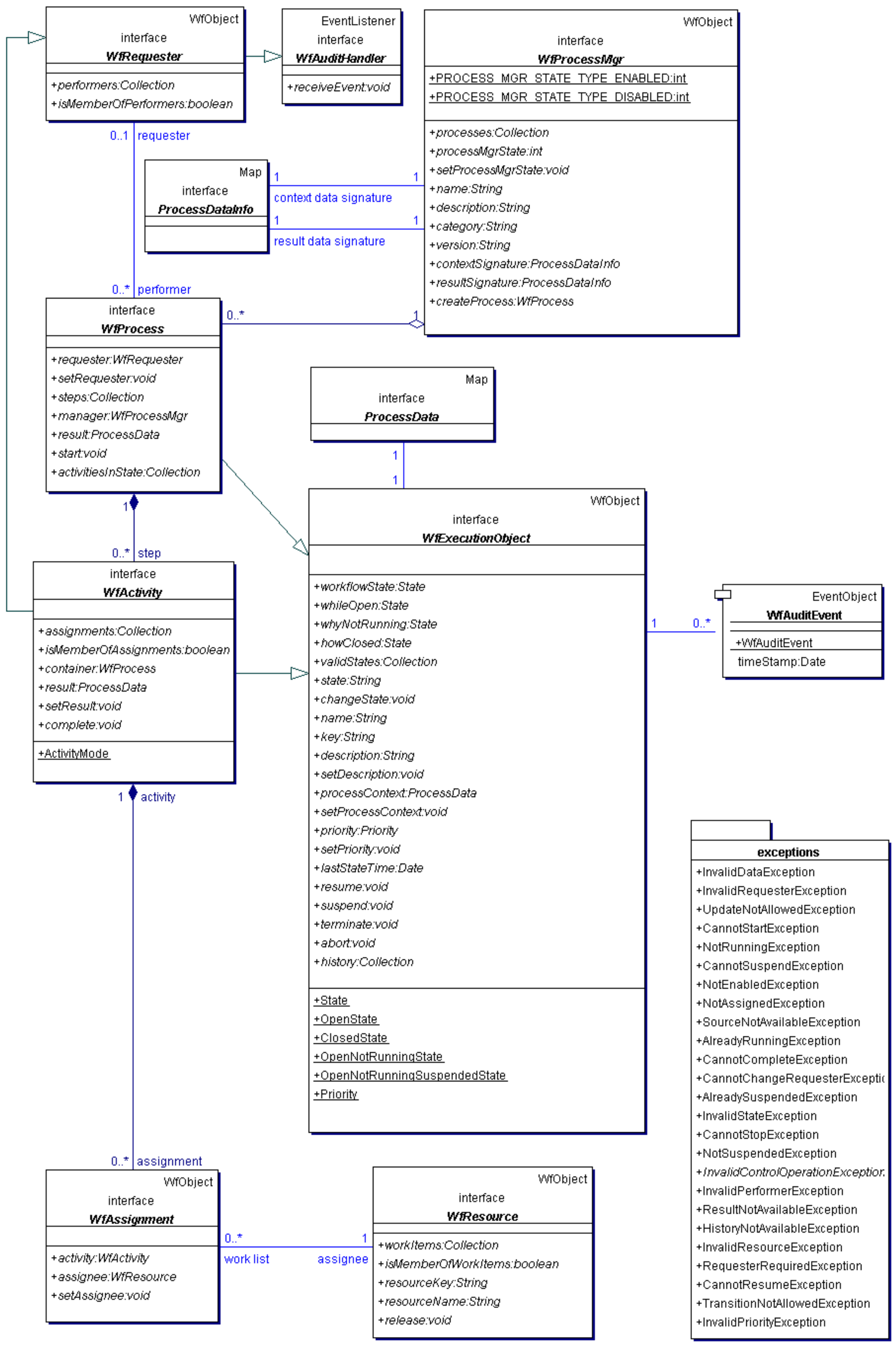


Figure 3.2. OMG core workflow model

The central classes of the model are `WfProcess` (see Section A.1.42, “Interface WfProcess” [132]) and its constituting `WfActivity` instances (see Section A.1.29, “Interface WfActivity” [98]). Processes of the same type can be created and accessed with their `WfProcessMgr` (see Section A.1.43, “Interface WfProcessMgr” [135]).

3.2.2. Extending the core interface

Examining the OMG interface closely, we found that it lacks some functions that we would like to offer. For some OMG interfaces we have in a second step therefore defined a corresponding interface that extends the OMG base interface.

We have also added some interfaces for areas that the OMG specification has purposely omitted from its scope (e.g. access to process definitions).

The starting point for using the workflow engine is the `WorkflowServiceFactory`. Use this class to create a new `WorkflowService` (see Section A.2.52, “Class WorkflowServiceFactory” [250], especially setProperty(java.lang.String, java.lang.Object) [253], and Section A.2.51, “Interface WorkflowService” [240]). The workflow service provides the methods to access the process definitions, processes etc.

Note that the extended API has been defined completely independent of J2EE. It is possible to implement this API with POJOs, though it may turn out to be difficult to provide remote access, reliable transactional behavior, security and scalability without using J2EE.

The implementation of the API provided by WfMOpen is J2EE based because we found this middleware an ideal basis for a workflow engine that can meet the demands of enterprises. The workflow service factory implementation provided by WfMOpen is described in de.danet.an.workflow.ejbs.client.StandardWorkflowServiceFactory [325]. See this description for more information about required configuration parameters.

3.3. Sample client

Holger Schlüter, Danet GmbH

This section covers a step-by-step description of the process of building a basic sample client.

The example helps you to learn how to set up your compilation and runtime environment and finally, you will be able to retrieve some information from the workflow engine.

3.3.1. Client code

Let's start with a first look at the code we need for this purpose:

```
package samples;

import java.util.Iterator;
import java.rmi.RemoteException;

import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;

import de.danet.an.workflow.api.WorkflowServiceFactory;
    // Section A.2.52, “Class WorkflowServiceFactory” [250]
import de.danet.an.workflow.api.WorkflowService;
    // Section A.2.51, “Interface WorkflowService” [240]
import de.danet.an.workflow.api.ProcessDirectory;
    // Section A.2.44, “Interface ProcessDirectory” [230]
import de.danet.an.workflow.omgcore.WfProcess;
    // Section A.1.42, “Interface WfProcess” [132]

import ProjectLoginContext; // Section 3.3.2, “Gaining access” [28]

/**
 * A sample workflow client.
 */
public class TestClient {

    private static LoginContext lctx = null;
    /**
     * Main method called when started from shell.
     * @param args arguments
     */
    public static void main(String args[]) {
        System.out.println("Started");

        // This block authenticates a user. It is needed only once
        // within a client program (see Section 3.3.2, “Gaining access” [28]
        try { // ❶
            lctx = new ProjectLoginContext();
            lctx.login();
        } catch (LoginException exc) {
            System.err.println("Login failed: " + exc.getMessage());
            System.exit(-1);
        }

        // Here starts the real work.
        WorkflowService wfs // ❷
            = WorkflowServiceFactory.newInstance().newWorkflowService();
        try {
            ProcessDirectory dir = wfs.processDirectory(); // ❸
            Iterator procs = dir.processes().iterator();  // ❹
            while (procs.hasNext()) {
                System.out.println
                    ("Proc: " + ((WfProcess)procs.next()).name()); // ❺
            }
        } catch(RemoteException exc) {
            System.err.println
                ("Process directory not accessible: " + exc.getMessage());
            System.exit(-1);
        }
    }
}
```

❶ The purpose of this block will be explained below (see Section 3.3.2, “Gaining access” [28]). It has nothing to do with the genuine workflow engine functionality.

❷ The first step in using the workflow service is to obtain a reference to a `WorkflowService` (see Section A.2.51, “Interface WorkflowService” [240]). Such a reference is obtained from the factory class `WorkflowServiceFactory` (see Section A.2.52, “Class WorkflowService-Factory” [250]).

❸ From the workflow service we can get the `ProcessDefinitionDirectory` (see Section A.2.43, “Interface ProcessDefinitionDirectory” [225]) or the `ProcessDirectory` (see Section A.2.44, “Interface ProcessDirectory” [230]). As we want to list the workflow processes in our sample clients, we get the `ProcessDirectory`.

❹ We call `processes` (see processes() [232]) to obtain the collection of known processes and iterate over them.

❺ For this example we simply output the names of the processes (see name() [115]). Of course, you may add print statements for additional attributes.

3.3.2. Running the sample client

Preparations

First of all, you have to install the database (see Section 1.2, “Preparing the database” [2]) and deploy the workflow ejbs (see Section 1.4, “Deploying the component” [9]).

If you have installed a demo application (see Appendix C, *The demo applications* [337]) you can use it as the server for the sample client.

Building environment

Next, add the following libraries to your compilation classpath:

- Danet's wfcore API library (`de.danet.an.wfcore-apis.jar`)
- Danet's AN utility library (`de.danet.an.util.jar`)
- All libraries needed by your J2EE client environment
- All libraries needed by your EJB client environment
- All third party libraries referenced in Section 1.3.7, “Additional libraries” [9]

You should now be able to compile the sample client code.

Running the client

After the successful compilation of your client, you need to add the following items to your runtime classpath:

- All libraries from the compilation classpath.
- Danet's wfcore client library (`de.danet.an.wfcore-client.jar`), adapted to your server configuration as described in Section 1.3.4, “Workflow module” [6][1]
- The directory with the preferred Log4J configuration for your client (in a file named `log4j-appl.xml`).
- The log4j library.

If you now try to start the sample client, you will get an error message, stating that the environment is unable to locate a login configuration. This is due to the fact that every access to an EJB is security controlled by the application container.

Gaining access

Access to EJBs may be secured by specifying roles that have access to methods. In the sample application, a single security role named `WfMOpenAdmin` has been configured, which a user has to adopt in order to gain full access to all application components (see the `ejb-jar.xml` in the `de.danet.an.wfcore-ejbs.jar`).

Since the J2EE specification does not define how to associate roles with a client, things get a bit vendor-specific here. Lookup your J2EE server's documentation to find out how things are handled in your environment. A popular method used by several J2EE vendors is to use JAAS based authentication. We'll explain the mechanism using the JBoss application server as an example.

Roughly, JAAS is based on a session related `LoginContext` that holds security information (including associated roles) about a `Principal` (typically a user's login name). One or more configurable "login modules" provide this information. The login modules are invoked when `LoginContext.login()` is called (see ❶ [27]). The login modules may use a callback to query credentials. If proper credentials are supplied, the login module adds information (such as roles) to the `LoginContext`.

If you have no predefined callback implementations available, you may use this example:

```
import java.io.IOException;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.UnsupportedCallbackException;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.TextOutputCallback;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
```

[1] If you use the demo application described in Appendix C, *The demo applications* [337] you can extract an adapted client library from the EAR. Go to the directory `$JBOSS_HOME/server/wfdemo/deploy` and extract the file `de.danet.an.wfcore-client.jar` from the `lib` subdirectory in the demo EAR.

```
import javax.security.auth.login.LoginContext;
import javax.security.auth.login.LoginException;
import javax.security.auth.login.FailedLoginException;

/**
 * Simple login context for unit tests.
 */
public class ProjectLoginContext extends LoginContext {

    public final static String USERNAME = "junit";

    private static class CBH implements CallbackHandler {
        public void handle (Callback[] callbacks) ❶
            throws UnsupportedCallbackException, IOException {
            for (int i = 0; i < callbacks.length; i++) {
                if (callbacks[i] instanceof TextOutputCallback) { ❷
                    // display the message according to the specified type
                    TextOutputCallback toc = (TextOutputCallback)callbacks[i
                    switch (toc.getMessageType()) {
                    case TextOutputCallback.INFORMATION:
                        System.err.println(toc.getMessage());
                        break;
                    case TextOutputCallback.ERROR:
                        System.err.println("ERROR: " + toc.getMessage());
                        break;
                    case TextOutputCallback.WARNING:
                        System.err.println("WARNING: " + toc.getMessage());
                        break;
                    default:
                        throw new IOException("Unsupported message type: " +
                                              toc.getMessageType());
                    }
                } else if (callbacks[i] instanceof NameCallback) { ❸
                    // prompt the user for a username
                    NameCallback nc = (NameCallback)callbacks[i];
                    nc.setName(USERNAME);
                } else if (callbacks[i] instanceof PasswordCallback) { ❹
                    // prompt the user for sensitive information
                    PasswordCallback pc = (PasswordCallback)callbacks[i];
                    pc.setPassword(USERNAME.toCharArray());
                } else {
                    throw new UnsupportedCallbackException
                        (callbacks[i], "Unrecognized Callback");
                }
            }
        }
    }

    public ProjectLoginContext () throws LoginException {
        super ("danetworkflow", new CBH()); ❺
    }
}
```

❶ At the first glance, things look much more complicated than they are. All we really have to do is implement the call back handler. It has a single method `handle` that has an array of callbacks to be handles as parameter.

❷ A `TextOutputCallback` is send by the login module if it want some text to be shown to the user (like "Invalid user name"). The messages are simply passed to the appropriate output channel.

❸ This callback is sent if the login module wants a user name. We used a fixed name here and simply set this as the callback result.

❹ This callback is sent if the login module wants a user credential (which need not be a string, hence it is typed as byte array). We used a fixed password (the name) here and simply set this (converted to a byte array) as the callback result.

❺ All we have to do in order to customize the standard `LoginContext` is to pass out callback handler and a security configuration name (see below) to the constructor.

The security configuration name used by the `LoginContext` must match an entry in the file that declares all security domains for JAAS. For our client (and JBoss) we need an entry like this:

```
danetworkflow {
    org.jboss.security.ClientLoginModule required;
};
```

This declares that a single login module `org.jboss.security.ClientLoginModule` should be called.

Finally we must tell JAAS that it should use our configuration file by starting the java virtual machine with this additional parameter: *`-Djava.security.auth.login.config=auth.conf`*.

The login context in our example provides credentials for the user “junit”. So we have to make sure, that the user “junit” is able to adopt the role `StaffManagementRole_0`. The described `JBoss` security configuration verifies given credentials on the basis of the Jetspeed2 user management component. Thus, to gain the proper access rights, make sure that a user name “junit” is defined within the Jetspeed2 portal and that this user is part of the administrator group (the group with the id “0”).

With these security settings, the client should now display a list with all processes currently running (or closed but not deleted) in the workflow engine.

3.4. Resource assignment SPI

The resource assignment SPI is described in detail in section de.danet.an.workflow.spis.ras [269]. It makes the resource assignment service a pluggable component, i.e. you are free to choose an assignment service that suits the needs of your business processes and your organization.

The sample resource assignment service included in the WfMOpen distribution (see Chapter 6, *The sample resource assignment service* [65]) simply supports the selection of specific humans, groups or resources. You may replace this with a component that takes in consideration skills, shift times etc.

A resource assignment service must implement a resource assignment factory and a resource assignment service as defined in Section A.4.6, “Class ResourceAssignmentServiceFactory” [279] and Section A.4.5, “Interface ResourceAssignmentService” [271]. An implementation of `ResourceAssignmentFactory` may require additional configuration properties to adapt the service instances that it creates to the specific environment. Usually, these properties are retrieved from JNDI environment entries or from a Java properties file (see Chapter 6, *The sample resource assignment service* [65] for an example).

The WfMOpen implementation guarantees that the resource assignment service factory is only used in the context of the `WorkflowEngineEJB`. Therefore, if you use JNDI environment entries to configure your factory, it is sufficient to add those entries to the `WorkflowEngineEJB`. Note that the guarantee to use the resource assignment service factory in this context only is implementation specific. Therefore the general SPI description leaves the issue of which EJBs require the entries deliberately open.

3.5. Tool invocation SPI

The tool invocation SPI provides the possibility to add "tool-plugins" to the workflow engine. Implementing tools is quite easy. The only interface that a class must implement in order to be usable as a tool is `ToolAgent` (see Section A.3.10, “Interface ToolAgent” [263]).

The tools that are bundled in the WfMOpen distribution are described in Chapter 5, *Tools* [43]. Their sources should should be consulted as examples for tool agent implementation.

3.5.1. Invocation mode

As explained in the package description of the application invocation interface (see de.danet.an.workflow.spis.aii [254]), tool agent invocations are conceptually asynchronous, i.e. the workflow engine does not expect the `invoke` method to return a value. This is a necessity for supporting tools that run for a long period of time (e.g. a tool that allows manual input of data). The process that calls `complete` on the activity may be completely different from the thread that invoked the tool agent.

Practical experience shows, however, that a lot of tools can reasonably execute completely within the invoke method. In order to simplify the implementation of this class of tools and avoid some problems with transactions (again, see de.danet.an.workflow.spis.aii [254] for details), the interface `ResultProvider` (see Section A.3.8, “Interface ResultProvider” [261]) has been defined. A tool agent implementing this interface can provide its result in a simple function call and the workflow engine handles forwarding the result to the activity and completing the activity.

3.5.2. Accessing the workflow engine context

Sometimes, tool agents need access to some workflow engine functions. While it is technically possible for tool agents to access the `WorkflowEngineEJB` (and older versions of the tools coming with WfMOpen did rely on this), this behavior is strongly discouraged. The methods that may be called by tool agents are made available in a `ToolAgentContext` (see Section A.3.11, “Interface ToolAgentContext” [265]). Tool agents that want to use this context must implement the interface `ContextRequester` (see Section A.3.4, “Interface ContextRequester” [257]) and will receive the context before method `invoke` is called[2].

3.5.3. Accessing JNDI

As tool agents run in the application server context, they can always access the application server's JNDI directory using "`new InitialContext()`". This context provides all global entries (including those restricted to local access from within the JVM, i.e. "`java:...`").

Sometimes it may be desirable to use logical names as provided for EJBs ("`java:comp/env/...`") instead of global names. This may facilitate deployment, because the mapping of the logical names used by the tool agents to global names can be administered in the same way as e.g. for mapping data sources used by EJBs.

While tool agents are not EJBs themselves, WfMOpen guarantees that tool agents run in the context of a business method of the `InvocationHelperEJB`. Tool agents can therefore additionally lookup via JNDI all entries that are bound to the "`java:comp/env/`" namespace of the `InvocationHelperEJB`. To provide a logical name for use by a tool agent, it is therefore sufficient to add the required entry to the `InvocationHelperEJB`'s section in `ejb-jar.xml`. We recommend to put entries created in this namespace for the sole use of a tool agent in a subcontext "`java:comp/env/toolagents/`*`agent name`*`/...`".

3.5.4. Exception handling

3.5.4.1. Default behaviour

As can be seen in the declaration of the `ToolAgent` interface (see Section A.3.10, “Interface ToolAgent” [263]), tool agents can throw `RemoteException`s and `CannotExecuteException`s.

In general, and especially in the J2EE environment, `RemoteException`s received when accessing a (remote) component can be interpreted as an indication that a service is temporarily unavailable. The current transaction should be rolled back and retried. To avoid having to implement this behaviour in every tool agent, the tool agent may indicate such a condition by raising a `RemoteException`. Usually, the tool agent does not create such an exception itself, rather it propagates an exception that it has received when accessing a remote component. The workflow engine does roll-back and re-invocation automatically if a `ToolAgent` raises a `RemoteException`.

[2]The methods made available in `ToolAgentContext` are based on the experience gained in implementing tools. If you feel that additional methods should be provided, please let us know.

Tools may also throw `CannotExecuteException`s during their execution. The workflow engine's default response to such an exception is to terminate the activity, and thus the process (see Section 2.1, “States of processes and activities” [15]). An exception raised by a tool is considered an indication of unexpected behaviour and it would be dangerous to continue the process after encountering an unexpected behaviour of a tool.

3.5.4.2. Overriding the default behaviour

The default behaviour may be overridden by either the tool or the application declaration. A tool can specify a mapping of Java exceptions that are reported as cause of the `CannotExecuteException` to process level exceptions (see Section 2.4, “Exceptions” [18]) by implementing the interface `ExceptionMappingProvider` (see Section A.3.5, “Interface ExceptionMappingProvider” [257]). The process definition can define such a mapping as described in Section 4.2.5.1, “Extentions of Application Declaration” [35]. Exceptions reported as cause of the`CannotExecuteException` by the tool are first matched against the mappings from the application definition and then against the mappings from the tool.

Note that there is some redundancy between exception mapping and the method `abandonActivity` made available by the workflow engine context (see Section 3.5.2, “Accessing the workflow engine context” [31] and Section A.3.11, “Interface ToolAgentContext” [265]). The main difference is that a call to `abandonActivity` cannot be redefined by the application declaration, it will always lead to the specified process level exception. Method `abandonActivity` should therefore be used if raising a specific process level exception is part of the tool's bahavioral specification and handling the condition differently is not an option.

If a tool deliberately uses an exception type as cause when constructing a `CannotExecuteException` and at the same time defines a mapping for this type of exception, it offers the user the possibility to redefine the bahavior. This is especially useful if the Java exception type has subclasses. In this case, the user of the tool has the possibility to specify special process level exceptions for the subclasses of the Java exception in the application declaration and can thus achieve differentiated behaviour.

3.5.4.3. Stopping the workflow

In addition to defining system exceptions for Java exceptions, the exception mapping may specify that the invoking activity should enter the `open.not_running.suspended.abandoning` state when the exception occurs. This will delay the actual delivery of the exception until the activity is resumed. Such a behaviour may be desired if unexpected exceptions should stop the process rather then terminate it. If e.g. combined with an exception transition that simply leads back to the activity, this offers the possibility to repeat the tool (agent) invocation after fixing the exception's cause (e.g. by modifying some data or restarting a required service).

Instead of simply resuming an activity from state `open.not_running.suspended.abandoning`, the state may first be changed to `open.not_running.suspended.clearing_exception`. This is an intermediate state that causes the activity to clear the pending exception and immediately proceed to "normal" suspended state `open.not_running.suspended.suspended`. Effectively, this causes the engine to ignore the exception, i.e. the engine behaves as if the tool invocation had been successful, and the activity had been set to the suspended state during tool invocation using the API[3]. This feature may be used to manually mend a failed tool invocation by setting the process data to the state expected after the (failed) tool invocation and resume the normal workflow.

[3]Acting as if execution had manually been suspended also applies to deadlines. If deadlines have expired while the activity has been in the suspended state caused by the exception, they will be handled as described in Section 2.5, “Suspended state and deadlines” [19]

Chapter 4. Process definitions

4.1. Managing process definitions

As an extension to the OMG API, the WfMOpen workflow engine is capable of managing an unlimited number of process definitions at a time. To gain access to a specific description, a directory component has been established, managing all packages and included process definitions (see Section A.2.43, “Interface ProcessDefinitionDirectory” [225]).

When a new process definition file is imported (via method `importProcessDefinitions`), all currently managed descriptions with the same package id as the imported package (regardless of the process' name) are removed from the process definition directory before adding any new process definition.

Note that replacing a process definition with another version (or removing a process definition) does not affect running processes. Once created, a process remains inseparably associated with the process definition that was effective when the process was created.

4.2. Process definition format

4.2.1. Base format

The import format for process definitions is XPDL which stands for "XML Process Definition Language". XPDL is a WfMC standard that is currently available as version 1.0 Final Draft.

4.2.2. XPDL and the OMG API

When using the XPDL and the OMG API together, attention has to be payed to the usage of some keywords. The most prominent among those is the usage of the attribute `Id` in XPDL vs. the usage of a key in the OMG API.

`Id`s are used in the XPDL to enable references between the different items defined. They have no significance to the OMG API (with one exception, see below). The OMG API uses *keys* to uniquely identify instances within their context. So an activity with key "a23" could e.g. be the instance of an activity that was defined in the XPDL with `Id="do_this"`.

The one exception where `Id`s have significance in our implementation of the OMG API is the process type. The OMG defines a process manager and this process manager has a "name" attribute denoting the process type. Currently, we use the `Id` from the XPDL as the process manager name (this may, however change in future versions).

It is important to maintain a clean separation between the usage of `Id` and *key* when programming against the workflow API. This is sometimes especially difficult, because keys are often labeled as "ids" in the user interface.

4.2.3. Specifying XML data

One of the most annoying problems with the current version of XPDL is the schema defined for `InitialValue` and `ActualParameter` tags. Although XPDL introduces a `SchemaType`, initial values and actual parameters may only be of type string, thus prohibiting the usage of structured data.

We have therefore defined that the content of the `InitialValue` of a `DataField` of type `SchemaType` is parsed as XML and used to initialize the data field. The XML may have multiple top nodes, i.e. a data field may be used to hold "lists"[1]. The XML must be self-contained, i.e. if you use namespaces they have to be declared within the string, namespaces of the XML surrounding

`InitialValue` are not inherited.

Starting with version 1.3.4, WfMOpen supports E4X, the XML extension for JavaScript. E4X introduces the XML "native" type (similar to e.g. the JavaScript native "Date" type) and extends the JavaScript lexical grammer to directly support XML text as initializers for new objects of this type. With E4X, parsable XML within JavaScript simply yields a new XML object. Therefore, no special provisions have to be taken to support actual parameters of `SchemaType`. An actual parameter specified as `<ActualParameter><![CDATA[<Hello/>]]></ActualParameter>` is passed to the JavaScript interpreter like any other actual parameter expression and evaluates to an XML object that is used when invoking the tool or sub-process. The XML may also have multiple top nodes. This can be expressed in E4X as e.g. `<ActualParameter><![CDATA[<><Hello/><World/></>]]></ActualParameter>`.

Note that the JavaScript expression need not be literal XML. It may be an arbitrarily complex JavaScript that ends with an expression of type XML. The use of complex scripts as actual parameters is, however, discouraged. A useful alternative to literal XML is e.g. the selection of a subtree from a process data item of `SchemaType`, such as `<ActualParameter>purchaseOrder.item.(id=5)</ActualParameter>`.

For backward compatibility with WfMOpen versions prior to 1.3.4, a special rule applies. If the actual parameter starts with the string "`<j:jelly`"[2], it is piped through the jelly interpreter before being passed to the invoked tool or sub-process. Within the jelly script, the process relevant data items are available as variables. These variables may not be modified (this is similar to the evaluation of an actual parameter of basic type using JavaScript). Setting a variable with the same name as a process data item will create a new variable that hides the process data item.

Besides the jelly core tags, the demo application makes the "xml" and "jsl" tags available by including the respective libraries. We consider these the only useful tags in the context of actual parameter evaluation. However, no special measures have been taken to enforce this subset. If other tag libraries are bundled with the application, these will be available as well. As with JavaScript expression, causing side-effects in actual parameter evaluation is, however, strongly discouraged.

The support for jelly in actual parameter evaluation is deprecated and will be dropped in WfMOpen 2.x.

4.2.4. Deadlines

WfMOpen attempts to parse a `DeadlineCondition` using the following methods:

1. Parse as a duration as defined in the XML Schema Definition (see XML Schema Part 2, "duration" [http://www.w3.org/TR/xmlschema-2/#duration]).

2. Parse as a date time specification as defined in the XML Schema Definition (see XML Schema Part 2, "dateTime" [http://www.w3.org/TR/xmlschema-2/#dateTime]).

3. Parse as value/unit pairs. Values can be arbitrary JavaScript expressions, valid unit names are "years", "yrs", "months", "mts", "hours", "hrs", "min" and "sec". There may be only one pair for each unit i.e. "5 years 3 yrs" is illegal.

   Value/unit pairs must be separated from each other with at least one whitespace character. Units may follow values immediately if the value is a number, else they must be separated by at least one whitespace.

   Any duration unit may be omitted, but the descending order of units must be preserved.

   When evaluating the JavaScript expression, process relevant data is available as read-only vari-

[1]In order to allow a string describing a not well-formed document to be parsed, an opening and closing tag `temporary-root` is added before and after the string and the corresponding node is removed from the result after parsing. This should be transparent to the user unless there is a parsing error. In this case you may find the tag being mentioned in the error message (though it will never be the cause of the problem).
[2]Note that this is really executed as a string comparison, i.e. you cannot use another prefix.

ables. The evaluation must yield a number.

4. Parse as a date time specification in the formats "d MMM yy HH:mm:ss z" and "d MMM yy HH:mm z" (see `java.text.SimpleDateFormat`). An optionally leading "EEE, " (abbreviated day) is ignored. These formats include RFC822 conformant date time specifications.

5. Evaluate as JavaScript. In this evaluation, process relevant data is available as read-only variables. The script must return a JavaScript "Date" type or a number. A result of type "Date" is interpreted as absolute time specification, a number is interpreted as seconds relative to the start time of the activity.

4.2.5. Defined extensions

Trying to define a process definition language that can be used for any workflow engine, the WfMC had to leave out some implementation specific details. The missing information has to be provided using vendor specific extentions. The following sections describes the extensions used by WfMOpen.

4.2.5.1. Extentions of Application Declaration

Within the XPDL file, a tool or application can only be declared or referenced. XPDL does not specify elements that can be used to describe the definition of the application. Usually, the binding to application implementation is handled for example by an object manager. To keep the process definition self-contained, we have chosen the straightforward approach of defining the definition details within the tool declaration. This has been achieved by defining an extended attribute named `Implementation`.

```
...
<Application Id="INCREMENT">
   <Description>Marking the current station within the transition path.
   </Description>
   <FormalParameters>
      <FormalParameter Id="counter" Mode="INOUT">
         <DataType>
            <BasicType Type="INTEGER"/>
         </DataType>
      </FormalParameter>
   </FormalParameters>
   <ExtendedAttributes>
      <ExtendedAttribute Name="Implementation">
          <vx:ToolAgent
            xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1"
            Class="de.danet.an.workflow.tools.rhino.JSExecutor"
            Execution="SYNCHR"
            XMLParamterMode="USE_W3C_DOM">
          <vx:Property Name="Script"><![CDATA[
            java.lang.System.out.println ("Incrementing counter "
               + args["counter"]);
            args["counter"] = args["counter"] + 1
           ]]></vx:Property>
           </vx:ToolAgent>
      </ExtendedAttribute>
   </ExtendedAttributes>
</Application>
```

In the example above, an application is declared with its implementation class (`de.danet.an.workflow.tools.rhino.JSExecutor`) specified as an extended attribute. The implementation class must implement `de.danet.an.workflow.spis.aii.ToolAgent` (see Section A.3.10, “Interface ToolAgent” [263]) and provide `setXXX` methods for each declared `Property` element.

In our example `de.danet.an.workflow.tools.rhino.JSExecutor` must thus implement a method `setScript`. If the content of a `<Property>` node is text, the required argument of the `setXXX` method must be of type `java.lang.String`. If the content of a `<Property>` node is another element, the argument may be either `org.jdom.Element` or `org.w3c.dom.Element` depending on the tool agent's preferred environment.

The execution type of a tool agent can be defined as `SYNCHR` or `ASYNCHR` (default) so that it may be executed synchronously or asynchronously with respect to state evaluation (see Section A.3.10, “Interface ToolAgent” [263]).

The actual parameters passed to the tool that are declared as `<SchemaType>` will be passed as `org.w3c.dom.Element` (default), `org.jdom.Element` or `de.danet.an.workflow.api.SAXEventBuffer` (see Section A.2.48, “Interface SAXEventBuffer” [236]) depending on the attribute `XMLParameterMode`. Possible values are `"USE_W3C_DOM"` (making the default explicit), `"USE_JDOM"` or `"USE_SAX"`. Note that the setting will only be effective if the tool does not implement the `XMLArgumentTypeProvider` interface (see Section A.3.12, “Interface XMLArgumentTypeProvider” [268]).

An additional attribute `Handler` (not shown above) can be used to cause invocation of the tool agent in another `InvocationHelperEJB` than the default one (usually in another EAR). See Section 1.3.5, “The optional callback module” [7] for a general description of this concept. Note that you must use the extended namespace `http://www.an.danet.de/2009/XPDL-Extensions1.1` instead of the old namespace `http://www.an.danet.de/2002/XPDL-Extensions1.0` if you want to specify the handler.

An application declaration may optionally include the definition of mappings of Java exceptions reported by the tool agent to process level exceptions (see Section 3.5.4, “Exception handling” [31]). The mapping is declared as child of the `ToolAgent` element, before the `Property` elements:

```
...
          <vx:ToolAgent ... >
            <vx:ExceptionMappings>
              <vx:ExceptionMapping JavaException="java.lang.Exception"
                                   ProcessException="Error"/>
            </vx:ExceptionMappings>
          <vx:Property Name="Script"> ...
```

If the attribute `ProcessException` is omitted, the default behaviour of terminating the activity when the exception occurs is re-established (overriding a mapping established by the tool agent implementation programatically, see Section A.3.5, “Interface ExceptionMappingProvider” [257]).

4.2.5.2. Extensions on Package and Process Level

Extensions:

`RemoveClosedProcess`	determines how the workflow engine removes a closed process. Values can be `MANUAL`, `AUTOMATIC` or `COMPLETED` [3] (default: `AUTOMATIC`). Starting with version WfMOpen version 1.3.1, the value for this extension may be specified either as attribute "`Value`" or as (textual) body of `<ExtendedAttribute>`. The latter form is considered deprecated. `COMPLETED` causes the process to be removed automatically only if its completion state is `closed.completed`. This is useful for keeping processes that have failed for debugging purposes.
`Debug`	if `True`, the process will be started in debug mode (see Section 2.6, “Debugging workflows” [19]). This can be specified on process level only. Starting with version WfMOpen version 1.3.1, the value for this extension may be specified either as attribute "`Value`" or as (textual) body of `<ExtendedAttribute>`. The latter form is considered deprecated.

[3]Added in 1.3.4

`AuditEventSelection`	The audit events reported by this process will be restricted as specified in the attribute "`Value`". The restriction applies both to the events reported to an audit handler (see receiveEvent(de.danet.an.workflow.omgcore.WfAuditEvent) [108]) and the events saved in the audit trail (see history() [113]) (see remarks at the end of this section). Valid values for the audit event selection are: `AllEvents`: All events will be reported. This is the default, i.e. equivalent to not specifying this extended attribute at all. `StateEventsOnly`: Only state change events (i.e. events of type `WfStateAuditEvent`, see Section A.1.46, “Interface WfStateAuditEvent” [143]) will be reported. `ProcessClosedEventsOnly`: Only state change events caused by a process assuming the closed state will be reported. This filter is useful if you have clients that use the notification mechanism to wait for the completion of processes. `NoEvents`: No events will be reported.
`StoreAuditEvents`	If the attribute "`Value`" is set to "`False`" the selected audit events (see above) will not be written to the event log. Consequently, they cannot be retrieved using the `history` method of the `WfExecutionObject`s created by this process (see history() [113]). The selected audit events will still be delivered to audit handlers (see remarks at the end of this section).

Example:

```
...
<package>
  ...
  <WorkflowProcesses>
    <WorkflowProcess>
      ...
      <Activities>
        ...
      </Activities>
      <ExtendedAttributes>
        <ExtendedAttribute Name="RemoveClosedProcess" Value="MANUAL"/>
        <ExtendedAttribute Name="Debug" Value="True"/>
        <ExtendedAttribute Name="AuditEventSelection" Value="AllEvents"/>
        <ExtendedAttribute Name="StoreAuditEvents" Value="True"/>
      </ExtendedAttributes>
    </WorkflowProcess>
  </WorkflowProcesses>
  <ExtendedAttributes>
    <ExtendedAttribute Name="RemoveClosedProcess" Value="AUTOMATIC"/>
  </ExtendedAttributes>
</package>
...
```

Filtering out events or disabling the audit trail causes the observable behaviour of the process at the API to deviate from the standard. It may nevertheless be a useful option as it reduces the number of database transactions significantly and thus increases throughput. Especially when using workflow processes to work off batches of datasets, a workflow specific logging provided by the invoked tools will most likely be more useful than the default audit trail.

4.2.5.3. Extensions on the Activity Level

Extensions:

`DeferredChoice`	if set to `true` the `AND`-split of an activity will be executed as deferred choice. See Section 4.3.5, “Deferred choice” [42] for an explanation of the semantics.

4.2.6. Miscellaneous

The definition and interpretation of the priority of a WfExecutionObject (e.g. a Process) differs between WfMC and OMG. According to WfMC, the priority can be any natural number starting with zero and higher numbers represent higher priorities. Conforming to the OMG specification, the priority has to be an integer between 1 to 5 with 1 being the highest priority. We have chosen to support OMG's specification which means that the priority specified in the process XPDL must be an integer value between 1 and 5 with descending priority and the result of a priority request (see priority() [115]) delivers the priority with equal semantic.

Since it is not defined how to obtain the category of the process manager from the XPDL description, we have chosen to interpret the package's Name attribute as being the category value. The same applies to the version attribute, which we have chosen to map to the element created within the process header definition.

Please note, that if you don't specify the script type for you package, it will default to "text/ecmascript" which is the most common used language for scripting within this context.

4.3. Semantics

XPDL (XML Process Definition Language) defines the syntax and grammar for the description of workflow processes that can be imported into the WfMOpen engine. Unfortunately XPDL is not precise about the interpretation of some description elements. This chapter describes our interpretation of all XPDL language elements that are important for the definition of the process flow.

Usually a workflow consists of some or all of the following elements:

- Activities that should be performed in a given order (successively and/or concurrently).
- Transitions between activities (described by "from" and "to" declarations), defining the execution sequence.
- Conditions for the transitions as criteria for the execution of the transition.

4.3.1. Start and finish of a process

Since XPDL does not define attributes for marking the first or last activity within a workflow, we have chosen to use those activities to be initially started which are not referenced within a "to" declaration of any of the workflow's transitions. We think this is an obvious approach that does not restrict the number of inital activities. Usually there will be only one entry activity but you may also define several one's which will then be started concurrently. That implies, that there has to be at least one of those activities or nothing will be started at all (in fact, a warning will report such a condition on import of the workflow description).

The workflow engine considers a running process as finished when there is nothing more to do, i.e. when after the completion of an activity there is no other activity currently running and no other activity to start (for a detailed description of process and activity states see Section 2.1, “States of processes and activities” [15]).

4.3.2. Split and Join

The simplest case of a workflow is one with just a single activity, thus defining the start and end point. As soon as a second activity is added that should run after the completion of the first, a transition between the two activities has to be defined. A transition has a "from" and a "to" attribute, referencing start and target of the connection. Once the "from" activity is completed, the "to" activity is started (assuming there are no conditions defined).

In real life, workflows do not just incorporate activities that will be started one after the other but there may be activities to be started concurrently or choices of activities depening on certain conditions.

There are typically two different situations to be dealt with when there is more than one subsequent activity:

- XOR-Split: One of many activities may be selected at runtime depending on certain conditions.
- AND-Split: Some of many activities may be selected at runtime depending on certain conditions.

In either case we have to deal with conditions describing selection criteria for the transition to be chosen. The language that is used to describe the condition can be defined via the "Script" element of the package. WfMOpen uses Javascript by default.

The workflow engine expects a boolean to be returned. That means, if more than one line of code is defined only the last one defines the condition. Prior statements may be used e.g. for debugging purposes. Note that conditions are evaluated exactly once when the "from" activity is completed.

Example:

```
...
  <Condition Type="CONDITION">
    java.lang.System.out.println ("counter is " + Counter);
    Counter &lt; 3
  </Condition>
...
```

Since XOR-Splits are used to select one of many activities although more than one of the given condition may evaluate to "true", an evaluation order has to be defined. This is done within the `TransitionRestrictions` of the activity. As soon as the first condition matches, this transition is chosen, the target activity is started and no further evaluation is performed.

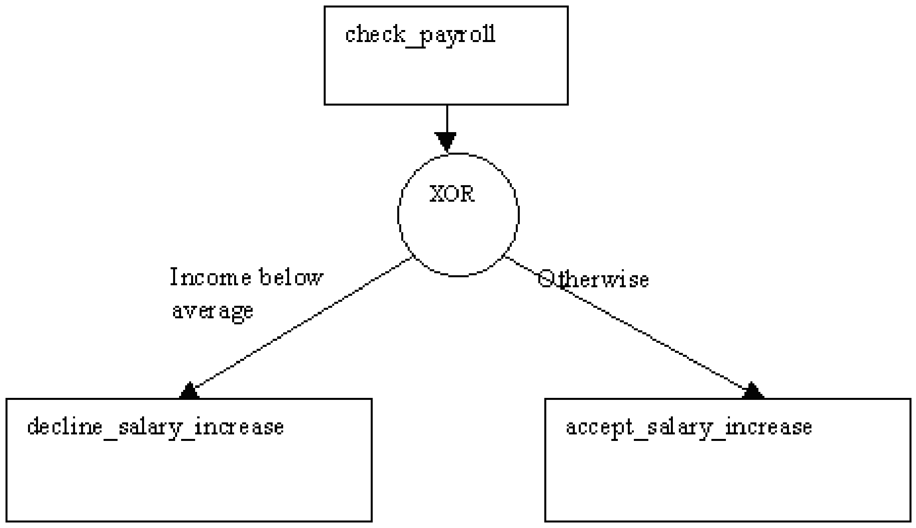


Figure 4.1. XOR-Split example

The definition of this workflow section may look like this:

```
...
<WorkflowProcess Id="XorSplitSample">
  <ProcessHeader/>

  <DataFields>
    <DataField Id="avg_payroll" IsArray="FALSE">
      <DataType>
        <BasicType Type="INTEGER"/>
      </DataType>
    </DataField>
    <DataField Id="test_payroll" IsArray="FALSE">
      <DataType>
        <BasicType Type="INTEGER"/>
      </DataType>
    </DataField>
  </DataFields>

  <Activities>
    <Activity Id="check_payroll">
      <Implementation>
        <No/>
      </Implementation>
      <TransitionRestrictions>
        <TransitionRestriction>
          <Split Type="XOR">
            <TransitionRefs>
              <TransitionRef Id="granted"/>
              <TransitionRef Id="declined"/>
            </TransitionRefs>
          </Split>
        </TransitionRestriction>
      </TransitionRestrictions>
    </Activity>
    <Activity Id="decline_salary_increase">
      <Implementation>
        <No/>
      </Implementation>
    </Activity>
    <Activity Id="accept_salary_increase">
      <Implementation>
        <No/>
      </Implementation>
    </Activity>
  </Activities>

  <Transitions>
    <Transition Id="declined" From="check_payroll" To="decline_salary_increase">
      <Condition Type="OTHERWISE"/>
    </Transition>
    <Transition Id="granted" From="check_payroll" To="accept_salary_increase">
      <Condition Type="CONDITION">
       avg_payroll &lt; test_payroll
      </Condition>
    </Transition>
  </Transitions>
</WorkflowProcess>
...
```

For AND-Splits no evaluation order is needed, since each transition with a condition evaluating to "true" is chosen and the target activity is started. Note that the workflow engine will start each of those activities in a different thread so that they are performed simultaneously.

As for splitting there a also typically two different situations to be dealt with when joining parallel activities into a common thread:

- XOR-Join: The first completed activity should trigger the joining activity. Note that the joining activity starts parallel to any predecessing activity, not yet completed.
- AND-Join: The joining activity is triggered as soon as all predecessing activities are completed. This situation may also be described as "synchronization" of acitities.

Since predecessing activities and the joining activity are connected via transitions, you may also define conditions here that will be evaluated before triggering. So, within an XOR-Join, the first completed activity with a matching condition will trigger the joining activity. Note that, in contrast to the XOR-Split, no evaluation order is needed here since it is defined implicitely by the the race condition of the activites. Once the second of the predecessing activities is completed, the engine will detect that the joining activity is (no longer) in state "not started" and thus do nothing with it.

A special case of join occurs, when a predecessing activity is at the same time a subsequent activity of the joining activity. This situation describes a loop within a workflow which will be discussed in more detail in the following section.

4.3.3. Condition evaluation

Condition evaluation for outgoing transitions takes place when an activity transitions to the `closed...` state. This is important to know in order to understand the behaviour of activities that use AND-Join.

If such an activity has two incoming transitions from e.g. activities `pre1` and `pre2` it may happen that `pre1` reaches its closed state long before `pre2`. If the transition from `pre1` has a condition that depends on some process relevant data item, the value of this data may have changed after `pre1` has reached the closed state but before `pre2` finishes. According to the definiton above, the condition will be evaluated based on the value at the time when `pre1` reaches the closed state, subsequent changes of the process relevant data will not change the result of this evaluation[4].

4.3.4. Loops

Looping within a workflow means that the workflow comes back to an activity that has already been completed before. WfMOpen supports loops in a very straightforward way and without limitations concerning start and end points of loops[5].

There is no special attribute defining an activity or transition being part of a loop. Rather the workflow engine of WfMOpen detects a loop by tracking the running workflow. As soon as a transition targets an activity that has already been marked as "been on the path", this activity is considered as entry point of a loop (the activity must, of course, have an XOR-Join mode).

If the target activity is completed, the workflow engine first resets the state of all activites on the path that lead to that activity to "not started" and the (re-)starts the target activity.

Asynchronous deadlines constitute new independant threads. Thus the activity started by the deadline has an empty path. You can therefore not loop back to the activity that defined the deadline. You can loop back to the activity started by the asynchronous deadline e.g. to have it re-triggered by a later deadline (though you'll need an extra route activity because of join conditions).

[4] WfMOpen versions up to 1.3.2 behaved differently. If an activity with AND-Join was triggered by a transition, the conditions of all other incoming transitions would be evaluated at that point in time (with evaluation errors for `OTHERWISE` and for abandoned predecessor activities). This behaviour is not an option as it can lead to unexpected results under certain circumstances. Starting with 1.3.3, WfMOpen behaves as defined above (including proper handling of `OTHERWISE` and abandoned activities).

[5] Note that you may not loop back to an entry activity, i.e. an activity that you want to be started when the process is started, because by doing so it will be no longer be detected as entry activity (remember, that the engine looks for activities that is not referenced within a "to" declaration of any transition!).

Synchronous deadlines continue processing using the same thread. Thus all activities that have been adandoned due to the deadline are predecessors of the activity started because of the deadline (there may be more than one activity that is abandoned because of the deadline if the deadline occurs on a block activity). The simplest usage of this behaviour may be a synchronous deadline that points back to the synchronous activity, thus causing a maybe different assignment of resources.

4.3.5. Deferred choice

WfMOpen supports the "deferred choice" pattern as described on the workflow patterns site [http://tmitwww.tm.tue.nl/research/patterns/deferred_choice.htm]. The pattern is implemented by defining an optional extended attribute for activities with an `AND`-split. E.g.:

```
...
<Activity Id="act1" Name="ACT1">
  <Implementation>
    <No/>
  </Implementation>
  <StartMode>
    <Automatic/>
  </StartMode>
  <FinishMode>
    <Automatic/>
  </FinishMode>
  <TransitionRestrictions>
    <TransitionRestriction>
      <Split Type="AND"/>
    </TransitionRestriction>
  </TransitionRestrictions>
  <ExtendedAttributes>
    <ExtendedAttribute Name="DeferredChoice">true</ExtendedAttribute>
  </ExtendedAttributes>
</Activity>
```

All activities following the activity with the deferred choice option set are started "preliminary". The activity (or tool started by the activity) that first performs a modification of the workflow engine's state automatically becomes the chosen activity. Modification of state includes a tool's calls to `setResult()` and `complete()`, the completion of an activity that has no implementation and the termination and abandoning of an activity. All other preliminary chosen activities are reset to the "not started" state (see Section 2.1, “States of processes and activities” [15]) after terminating any tools that are being run by these activities.

The preliminary choice implies the possibility that a tool executed by an activity has already performed some work that cannot be rolled back when the preliminary choice is revoked, i.e. in the tool's implementation of `terminate()`. A "deferred choice aware" tool may therefore prematurely (i.e. before calling `setResult()` or `complete()`) force the eventual choice to be made by calling `choose()` on the activity (see choose() [148]). An example of such a case would be a user front-end where the selection of an assigned activity (i.e. the begin of work) is already decisive for the continuation of the process. If not invoked as part of a deferred choice, the call to `choose()` simply does nothing.

Chapter 5. Tools

5.1. Overview

Tools are the "workhorses" of a workflow process. They call backend systems, retrieve data from a database, process data etc. WfMOpen comes with a set of ready-to-use tools and allows you to add your own tools using some well-defined APIs.

The tools included in the distribution are:

XForms tool	this tool can be used to build a web dialog for filling out data fields (see Section 5.2, “The XForms Tool” [43]).
JSExecutor and JSExecutor2	these tools implements an interface to the Rhino JavaScript interpreter (see Section 5.3, “JavaScript tools” [46]).
JellyTool	this tool implements an interface to the Jelly XML interpreter (see Section 5.4, “Jelly tool” [47]).
LDAP Tool	this is an extension of the Jelly tool that allows easy LDAP queries and manipulation (see Section 5.4.2, “LDAP tag library” [48]).
MailTool	this tool implements an interface to the javax mailing service (see Section 5.5, “Mail tool” [51]).
XSLTTool	this tool implements a tool that invokes a xslt processor dynamically (see Section 5.6, “XSLT tool” [52]).
Generic SOAPTool	this tool implements a tool that invokes SOAP based web services dynamically (see Section 5.7, “Generic SOAP tool” [54]).
RPC SOAPTool	this tool implements a tool that invokes web services with SOAP RPC style dynamically (see Section 5.8, “RPC SOAP tool” [55]).
WaitTool	this tool waits for a given time span, optionally in an interruptable way (see Section 5.9, “Wait tool” [56]).
MBeanInvoker	this tool invokes a method of an MBean (see Section 5.10, “MBean invocation tool” [58]).
Channel based Access	Tools that support message oriented communication with process instances (see Section 5.12, “Channel based access” [59]).

The following sections describe the predefined tools and provide some hints for implementing additional tools[1].

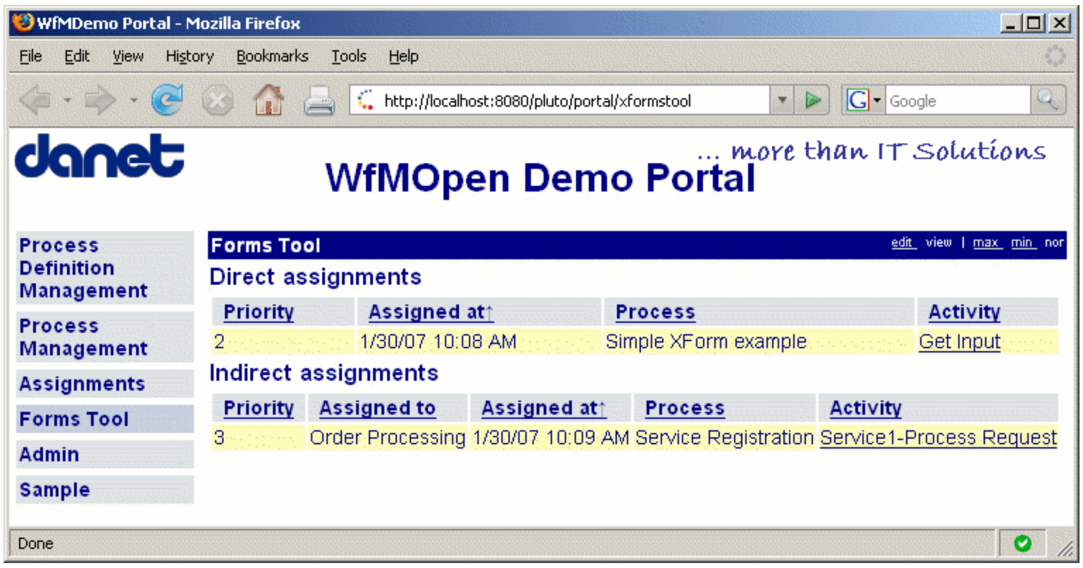
5.2. The XForms Tool

5.2.1. Usage

[1] Note that the application declarations in this section have been updated to use the new namespace `http://www.an.danet.de/2009/XPDL-Extensions1.1` that has been defined for version 2.1.2. If you do not want to specify a `Handler` attribute as described in Section 4.2.5, “Defined extensions” [35], you may still use the old namespace `http://www.an.danet.de/2002/XPDL-Extensions1.0`.

If an activity requires human interaction such as providing some data in a form, the activity may submit a form and associated initial data to the XForms tool. Then, the user associated with the activity invokes the XForms tool in order to manually complete the activity.

The XForms tool is an example of asynchronous application invocation and therefore consists of two components. The first component, the tool agent, is invoked by the workflow engine and puts information about the form to complete and the assignee in the database. The second component, the actual application, consists of a portlet. Initially, it lists all activities assigned to the currently logged in user. The display distinguishes between activities that are directly assigned to the current user and activities that are indirectly assigned, i.e. that are assigned to groups or roles the current user belongs to.


WfMDemo Portal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/pluto/portal/xformstool
Google
danet
... more than IT Solutions
WfMOpen Demo Portal
Process Definition Management
Process Management
Assignments
Forms Tool
Admin
Sample
edit view | max min nor
Done

Forms Tool

Direct assignments

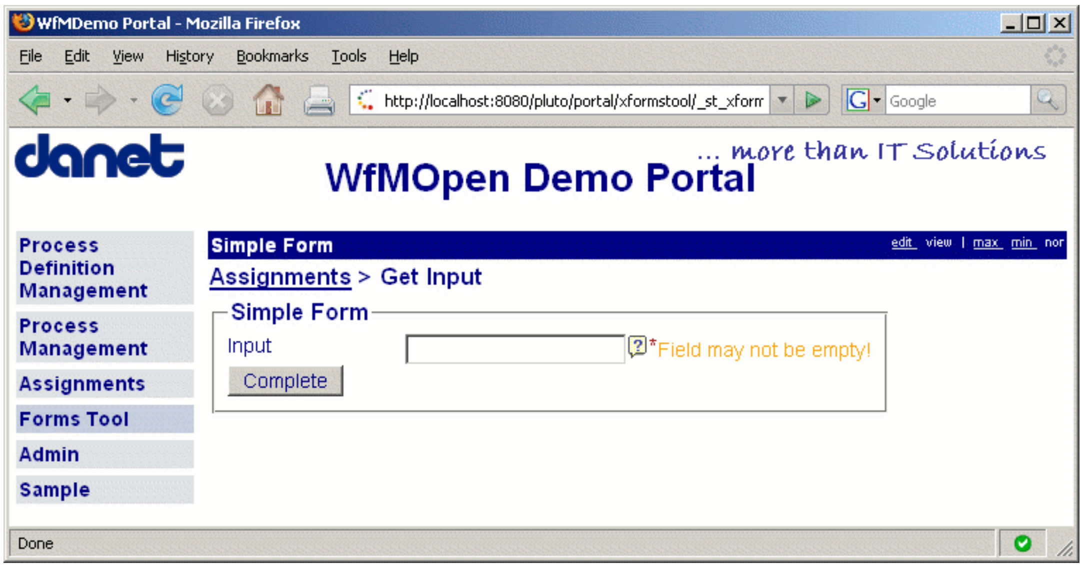
Priority	Assigned at↑	Process	Activity
2	1/30/07 10:08 AM	Simple XForm example	Get Input

Indirect assignments

Priority	Assigned to	Assigned at↑	Process	Activity
3	Order Processing	1/30/07 10:09 AM	Service Registration	Service1-Process Request

Figure 5.1. XForm task list

The web form for a specific activity is selected and displayed by selecting the link in the "Activity" column. The form may then be used to enter the required data. When the form is submitted, the entered data is used as result data of the activity and the activity is set to state "completed".


WfMDemo Portal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/pluto/portal/xformstool/_st_xform
Google
danet
... more than IT Solutions
WfMOpen Demo Portal
Process Definition Management
Process Management
Assignments
Forms Tool
Admin
Sample
edit view | max min nor
Done

Simple Form

Assignments > Get Input

Simple Form

Input ? *Field may not be empty!

Complete

Figure 5.2. XForm sample

5.2.2. Defining an XForms application in the process definition

An application of type "XFormTool" is specified in the process definition as described in Section 4.2.5.1, “Extentions of Application Declaration” [35], with the extended attribute `Implementation` referencing the class `de.danet.an.xformstool.Submitter`.

The tool agent has a mandatory property named `Form` that defines the XForm (see XForms specification [http://www.w3.org/MarkUp/Forms]) to display.

A complete example of such an application definition is shown below:

```
...
<Application Id="SimpleForm">
  <Description>
    Simple data form.
  </Description>
  <FormalParameters>
    <FormalParameter Id="input" Mode="OUT">
      <DataType><BasicType Type="STRING"/></DataType>
    </FormalParameter>
  </FormalParameters>
  <ExtendedAttributes>
    <ExtendedAttribute Name="Implementation">
      <vx:ToolAgent Class="de.danet.an.xformstool.Submitter"
        Execution="SYNCHR">
        <vx:Property Name="Form">
          <html xmlns="http://www.w3.org/1999/xhtml"
            xmlns:xforms="http://www.w3.org/2002/xforms"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
            <head>
              <title>Simple Form</title>
              <style>
                .SimpleForm .control { display:block; }
                .SimpleForm .control label {
                   display:block; float:left; width:9em; }
              </style>
              <xforms:model>
                <!-- Instance will be inserted here -->

                <!-- override/add some bindings -->
                <xforms:bind id="SimpleForm:input"
                  nodeset="ActualParameter[@name='input']"
                  constraint="string-length(.) &gt; 0"
                  required="true()"/>
              </xforms:model>
            </head>
            <body class="SimpleForm">
              <xforms:group id="C-3" appearance="minimal">
                <xforms:label id="C-4">Simple Form</xforms:label>
                <xforms:input bind="SimpleForm:input" incremental="true">
                  <xforms:label>Input </xforms:label>
                  <xforms:alert>Field may not be empty!</xforms:alert>
                  <xforms:hint>Please enter something.</xforms:hint>
                  <xforms:help id="C-7h">Use the <b>keyboard</b>.</xforms:he
                </xforms:input>
                <xforms:submit submission="action">
                  <xforms:label>Complete</xforms:label>
                </xforms:submit>
              </xforms:group>
            </body>
          </html>
        </vx:Property>
      </vx:ToolAgent>
    </ExtendedAttribute>
  </ExtendedAttributes>
</Application>
...
```

The form definition consists of XHTML and XForms elements. It must have an `<xhtml:html>` as root element. The nested `<xhtml:head>` may have a `<xhtml:title>`. If specified, it is displayed as title of the portlet when the form is selected and displayed. Any specified styles are copied to the the portal page[2]. To avoid ambiguities between forms display in different portlets on the same page, the style declarations should be made unique by starting the match expression with the application's id as a class. The HTML rendered for the form in the portlet will have an enclosing `<div>` with the application's id as class attribute. CSS styles specified with this class name in the match expression will therefore only apply to this form and no other form on the same portal page.

As last child of `<xhtml:head>`, the tool automatically generates and inserts an XForms model derived from the formal parameters. The model looks like this:

```
<xforms:model>
  <instance xmlns="">
    <ActualParameters xmlns="">
      <ActualParameter name="input"/>
    </ActualParameters>
  </instance>
  <submission xmlns="http://www.w3.org/2002/xforms"
    action="invoke:application" id="action" method="" replace="none"/>
  <xforms:bind id="SimpleForm:input"
    nodeset="ActualParameter[@name='input']"/>
</xforms:model>
```

For each formal parameter, an element `<ActualParameter>` is generated with an attribute "`name`" that equals the id of the formal parameter. In addition, a binding is generated for each formal parameter with an id derived from the combination of the application id and formal parameter id. These generated bindings may be overridden and supplemented with bindings supplied in the XForm description as shown in the example above.

Finally, the form defines the `<xhtml:body>` with the actual GUI elements. The form is processed with Chiba XForms processor (see Chiba home page [http://chiba.sourceforge.net/]). See the Chiba documentation for details about how HTML generated from the XForm definition.

5.3. JavaScript tools

The JavaScript tools can be used to execute embedded java script code (see example in Section 4.2.5.1, “Extentions of Application Declaration” [35]). The tool comes in two flavours. Both support JavaScript 1.5 with E4X. They differ in their handling of arguments.

The class `de.danet.an.workflow.tools.rhino.JSExecutor` provides an implementation that references formal parameters of the tool invocation in the script via `args["parameter_id"]`. Parameters of numeric type and of type `String` are made available as the corresponding JavaScript objects. Parameters of type `Date` and `SchemaType` are not converted. Rather the internally used Java representation (`java.util.Date` or `de.danet.an.workflow.api.SAXEventBuffer` respectively, see Section A.2.48, “Interface SAXEventBuffer” [236]) is passed to the JavaScript interpreter. Usage of this flavour of the tool is considered deprecated. However, the tool will remain part of WfMOpen.

Class `de.danet.an.workflow.tools.rhino.JSExecutor2`, available since version 1.3.4, represents the modern flavour of the tool. It provides the formal parameters declared for the tool as top-level JavaScript variables. In addition to the convertions of numeric types and strings, this version of the tool also converts parameters of type `Date` to JavaScript `Date` objects and parameters of `SchemaType` to JavaScript (E4X) XML objects.

[2]CSS styles can only be defined on the page level.

Besides the arguments, the JavaScript execution environment provides the global variable `scriptingContext`. This context defines the following methods and properties.

- The method `abandon(String)` (see Section A.3.11, “Interface ToolAgentContext” [265]). Invoking this method causes the JavaScript execution to be terminated[3] and the exeption name used as parameter to be passed to the condition evaluation for transitions to subsequent activities.
- The property `activityUniqueKey`. This property is of type `ActivityUniqueKey` (see Section A.2.11, “Class ActivityUniqueKey” [166]) and may be used to obtain some information about the invoking activity.

As the implementation of the JavaScript tool is based on Mozilla's `Rhino` implementation (see http://www.mozilla.org/rhino), you may use all features of this package (e.g. call static member functions of java classes within the JavaScript code). Example:

```
java.lang.System.out.println
          ("Waiting " + args["ms"] + "ms");
```

5.4. Jelly tool

The Jelly tool can be used to execute Jelly script code (see Jelly documentation [http://jakarta.apache.org/commons/jelly/]). A Jelly script is basically XML that consists of tags that are either copied to the output or processed by the Jelly interpreter. Processing tags may result in the generation of additional XML nodes in the output.

5.4.1. General usage

The first formal parameter of a Jelly tool declaration must be an `OUT` parameter of `SchemaType`. It receives the result, i.e. the generated XML. The other formal parameters of the tool invocation may be of arbitrary type and are directly available as variables within the Jelly script. Values assigned to variables that correspond to formal `OUT` or `INOUT` parameters in the Jelly script will change the values of the actual parameters upon tool completion.

Its SQL library makes Jelly the perfect tool for interfacing between XML data structures and an RDBMS. The library provides the same tags as the JSTL SQL tag library [http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSTL7.html]. The sample script below retrieves the names of the process definitions from the Workflow engine's database[4].

```
 ...
<Application Id="DBJelly">
  <Description>Database Jelly test</Description>
  <FormalParameters>
    <FormalParameter Id="result" Mode="OUT">
      <DataType>
        <SchemaType/>
      </DataType>
    </FormalParameter>
  </FormalParameters>
  <ExtendedAttributes>
    <ExtendedAttribute Name="Implementation">
      <vx:ToolAgent Class="de.danet.an.workflow.tools.JellyTool">
        <vx:Property Name="Script">
          <j:jelly trim="false" xmlns:j="jelly:core" xmlns:x="jelly:xml"
            xmlns:html="jelly:html" xmlns:sql="jelly:sql"
            xmlns:sqlx="jelly:de.danet.an.util.jellytags.Library">
```

[3]This behaviour differs from the behaviour defined for `ToolAgentContext.abandonActivity` (see abandonActivity(java.lang.String) [266]), which returns after passing the information to the workflow engine and allows the Java programmer to asynchronously cleanup resources.
[4]This is intended as an immediately (i.e. without setting up additional data sources) working example. Of course, you should never access the workflow engine's database directly, use the API instead!

```
            <sql:setDataSource dataSource="jdbc/WfEngine"/>

            <sqlx:query var="results">
              SELECT PackageId, ProcessId FROM ProcessDefinition
              WHERE PackageId = ?
              <sql:param value="jellytests"/>
            </sqlx:query>

            <dataSet>
              <j:forEach items="${results.rowsByIndex}" var="row">
                <row>
                  <j:forEach var="columnName"
                    items="${results.columnNames}" indexVar="i">
                    <field column="${columnName}">${row[i]}</field>
                  </j:forEach>
                </row>
              </j:forEach>
            </dataSet>
          </j:jelly>
        </vx:Property>
      </vx:ToolAgent>
    </ExtendedAttribute>
  </ExtendedAttributes>
</Application>
...
```

The example script uses a special `query` tag. This tag uses a `PreparedStatement` wrapper from the Danet utility library that handles large strings correctly even if you use Oracle as RDBMS. A similar `update` tag is available in the namespace `jelly:de.danet.an.util.jellytags.Library` as well.

Although it is possible to define "stand-alone" data sources for your RDBMS using the SQL tag library, it is much better to use the pooled connections managed by the application server. The SQL tag library supports access to these pools via JNDI, as shown in the example. Note that "`java:comp/env/`" is prepended before the data source name (so the lookup made in the sample code is "`java:comp/env/jdbc/WfEngine`"). The data source must therefore be defined as a logical name in an EJB context. All tools are invoked from a business method of the `InvocationHelperEJB`. Therefore, if you need additional data sources, you have to define them in this EJB's context (either by modifying `ejb-jar.xml` and its associated application server specific descriptor manually or by using some tool from your application server vendor).

As the tool is executed in the context of an EJB's business method, the `<transaction>` tag should not be used. The business method that invokes the tool has the container managed transaction attribute set. Thus the application server automatically wraps all database queries and updates executed during the tool invocation in a transcation.

Note that there are a lot of tag libraries available for Jelly. Some of these interface to quite complex libraries. The demo applications (see Appendix C, *The demo applications* [337]) includes only the tag libraries "jsl", "log", "sql", and "xml". Of course, you are free to re-pack the application with more libraries included.

5.4.2. LDAP tag library

Thinking about an LDAP tool, we found that a query tool can be implemented in a straight forward manner. A tool that also supports LDAP manipulation, however, would require an unmanageable number of configuration properties to support the different kinds of queries imaginable. We have therefore implemented an LDAP tag library for Jelly that enables easy query and manipulation of an LDAP directory.

5.4.2.1. `<ldap:setInitialContext>`

All query and update tags require a connection to the LDAP server. This connection can be obtained with the `<ldap:setInitialContext>`. The usage pattern is:

```
<ldap:setInitialContext
    var="ctx"
    xmlns:ldap="jelly:de.danet.an.util.jellytags.ldap.Library"
    providerUrl="ldap://localhost:389"
    securityPrincipal="cn=Manager,dc=my-domain,dc=com">
    <ldap:environmentEntry name="java.naming.security.credentials"
        value="${credential}"/>
</ldap:setInitialContext>
```

The `<ldap:setInitialContext>` tag creates a connection (actually an initial `DirContext`) and assigns it to the variable named in attribute `var`. Nested within the tag are `<ldap:environmentEntry name="..." value="..."/>` tags that specify values for the environment used to obtain the initial context (see Sun's JNDI specification for details).

For convenience, some environment entries may also be specified as attributes.

Table 5.1. Entries that can be specified as attributes

Entry name	Attribute
`java.naming.factory.initial`	`initialContextFactory`
`java.naming.provider.url`	`providerUrl`
`java.naming.security.principal`	`securityPrincipal`
`java.naming.security.credentials`	`securityCredentials`

Entry `java.naming.factory.initial` has a preset default of `com.sun.jndi.ldap.LdapCtxFactory`.

5.4.2.2. `<ldap:query>`

The `<ldap:query>` queries the LDAP server for a specific entry or a list of entries. The usage pattern is:

```
<ldap:query ldapContext="${ctx}"
   dn="ou=People,dc=my-domain,dc=com"
   filter="(uid=lipp)" attributes="cn, uid,gecos"/>
```

The attribute `ldapContext` must be specified and must refer to a previously created context. Attribute `dn` must also be given. It specifies the entry or the base for the directory search. Attribute `filter` is optional. If not specified, `dn` must refer to an entry in the directory, else `dn` is used as base for a search with `filter`. It `attributes` is specified, only the comma separated list of attributes of the entry or entries is retrieved from the server, thus saving bandwidth and CPU cycles.

The data retrieved is output by the tag as an XML structure:

```
<queryResult dn="ou=People,dc=my-domain,dc=com"
    filter="(uid=lipp)">
  <entry dn="uid=lipp,ou=People,dc=my-domain,dc=com">
    <attribute name="gecos">Michael N. Lipp</attribute>
    <attribute name="uid">lipp</attribute>
    <attribute name="cn">Michael N. Lipp</attribute>
  </entry>
</queryResult>
```

This structure may be output directly or further processes by Jelly. Here is a complete application declaration that converts the retrieved data to a user record.

```
<Application Id="LDAPQuery">
 <Description>Query fixed entry</Description>
 <FormalParameters>
  <FormalParameter Id="result" Mode="OUT">
```

```
   <DataType>
    <SchemaType/>
   </DataType>
  </FormalParameter>
 </FormalParameters>
 <ExtendedAttributes>
  <ExtendedAttribute Name="Implementation">
   <vx:ToolAgent Class="de.danet.an.workflow.tools.JellyTool">
    <vx:Property Name="Script">
     <j:jelly trim="false" xmlns:j="jelly:core" xmlns:x="jelly:xml"
      xmlns:html="jelly:html" xmlns:log="jelly:log"
      xmlns:ldap="jelly:de.danet.an.util.jellytags.ldap.Library">
      <!-- You will probably not put this in the process description.
      Use Jelly functions to obtain the password from e.g. JNDI
      environment, a properties file or whatever password store suits
      your application. -->
      <j:set var="credential" value="********"/>

      <!-- ldap:setInitialContext supports attributes
      "initialContextFactory", "providerUrl", "securityPrincipal"
      and "securityCredentials" as short-cuts.
      More entries may be set in the environment used by using nested
      "environmentEntry" tags, as shown. -->
      <ldap:setInitialContext var="ctx" providerUrl="ldap://localhost:389"
       securityPrincipal="cn=Manager,dc=my-domain,dc=com">
       <ldap:environmentEntry name="java.naming.security.credentials"
        value="${credential}"/>
      </ldap:setInitialContext>

      <!-- Execute the query and save the result document -->
      <x:parse var="response">
       <ldap:query ldapContext="${ctx}" dn="ou=People,dc=my-domain,dc=com"
        filter="(uid=lipp)" attributes="cn, uid,gecos"/>
      </x:parse>

      <!-- Output the user record -->
      <x:set var="userId" select="$response/queryResult/entry/@dn"
       single="true" asString="true"/>
      <x:set var="userName"
       select="$response/queryResult/entry/attribute[@name='cn']"
       single="true" asString="true"/>
      <user id="${userId}" name="${userName}"/>
     </j:jelly>
    </vx:Property>
   </vx:ToolAgent>
  </ExtendedAttribute>
 </ExtendedAttributes>
</Application>
```

Of course, part of the query criteria (`dn` or the value in `filter`) will usually be passed to the application as a parameter.

5.4.2.3. `<ldap:insert>`

The tag `<ldap:insert>` inserts new entries in the directory. The usage pattern is:

```
<ldap:insert ldapContext="${ctx}">
  <entry dn="uid=test,ou=People,dc=my-domain,dc=com">
    <attribute name="objectClass">top</attribute>
    <attribute name="objectClass">account</attribute>
    <attribute name="description">Just a test account</attribute>
  </entry>
</ldap:insert>
```

The tag itself has only one attribute `ldapContext` that references a previously obtained LDAP connection. The entries to be added to the directory are nested within the tag. They are formatted exactly as the results of a query. This facilitates deriving new entries from existing entries.

5.4.2.4. `<ldap:update>`

The tag `<ldap:update>` updates existing entries in the directory. The usage pattern is:

```
<ldap:update ldapContext="${ctx}" operation="replace">
  <entry dn="uid=test,ou=People,dc=my-domain,dc=com"
    operation="replace">
    <attribute name="description" operation="replace">
     Just an updated test account
    </attribute>
    <attribute name="organizationName">Testing</attribute>
  </entry>
</ldap:update>
```

The tag itself has only one mandatory attribute `ldapContext` that references a previously obtained LDAP connection. An optional attribute `operation` may be specified at the `ldap:update` tag or for the `entry` or for an `attribute`. It controls whether the attribute is replaced (`"replace"`), added (`"add"`) or removed (`"remove"`). If no operation is specified for a tag, the operation specified for the enclosing tag is performed. `ldap:update` has a default operation of `"replace"`.

5.4.2.5. `<ldap:delete>`

The tag `<ldap:delet>` deletes existing entries in the directory. The usage pattern is:

```
<ldap:delete ldapContext="${ctx}">
  <entry dn="uid=test,ou=People,dc=my-domain,dc=com"/>
</ldap:delete>
```

The tag itself has only one mandatory attribute `ldapContext` that references a previously obtained LDAP connection. The entries to be deleted are specified as nested `entry` tags. Any `attribute` tags nested within an `entry` tag are simply ignored. This allows using a query result directly as selection of entries to be deleted.

5.5. Mail tool

The mail tool builds an interface to the javax mail facility and thus can be used to send text messages to any recipient. Formal parameters are interpreted in the following order of declaration:

1. address(es) of recipient(s)
2. subject
3. message
4. address of sender

Furthermore, in case that no actual valid sender can be given as formal parameter (e.g. the data field is not set), a default sender can be declared via a property of the tool agent.

Example:

```
...
    <Application Id="MailTool2">
      <FormalParameters>
        <FormalParameter Id="recipient" Mode="IN">
          <DataType>
            <BasicType Type="STRING"/>
          </DataType>
        </FormalParameter>
        <FormalParameter Id="subject" Mode="IN">
          <DataType>
```

```
            <BasicType Type="STRING"/>
          </DataType>
        </FormalParameter>
        <FormalParameter Id="message" Mode="IN">
          <DataType>
            <BasicType Type="STRING"/>
          </DataType>
        </FormalParameter>
        <FormalParameter Id="sender" Mode="IN">
          <DataType>
            <BasicType Type="STRING"/>
          </DataType>
        </FormalParameter>
        <FormalParameter Id="status" Mode="OUT">
          <DataType>
            <BasicType Type="STRING"/>
          </DataType>
        </FormalParameter>
      </FormalParameters>
      <ExtendedAttributes>
        <ExtendedAttribute Name="Implementation">
          <vx:ToolAgent Class="de.danet.an.workflow.tools.MailTool2"
            xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1">
            <vx:Property Name="DefaultSender">test@WfMOpen.com</vx:Property>
          </vx:ToolAgent>
        </ExtendedAttribute>
      </ExtendedAttributes>
    </Application>
    ...
```

The mail provider to be used is determined by a JNDI lookup using the logical name `java:comp/env/toolagents/mailtool/Mail` in the context of the `InvocationHelperEJB` (see Section 3.5.3, “Accessing JNDI” [31]). As distributed, WfMOpen binds this logical name in the JBoss specific deployment descriptor to the global JNDI entry `java:/WfMOpenMail`. Make sure that this entry exists. For JBoss, we recommend to create a copy of `$JBOSS_HOME/server/`*`configuration`*`/deploy/mail-service.xml` (e.g. as `wfmopen-mail-service.xml`), set the entry `JndiName` within the file to `java:/WfMOpenMail`, and adjust the entry `mailhost.smtp.host` (minimal configuration).

5.6. XSLT tool

This tool can be used to transform xml content with an XSLT processor. The transformation is done using the JAXP 1.1 API, which introduces an abstract layer for the transformation process. The stylesheet to be used in the transformation may either be given by a URL or an inline definition using the property `XSLT`. The two cases are distinguished by checking whether the `XSLT` property defines a valid URL or not. If no `XSLT` property is given, the output document is the same as the input document (this allows the tools to be used as a merger or value extractor, see below). The stylesheet must produce a well formed xml-document for the tool to work correctly.

The first formal parameter containing xml content defines the content to be transformed by the tool. Additional input parameters are used to set stylesheet parameters. The result of the transformation is assigned to all formal output parameters, applying an additional mapping if defined using the `Mappings` property.

Example:

```
...
<Application Id="XSLT">
  <Description>
  </Description>
  <FormalParameters>
    <FormalParameter Id="inlist" Mode="IN">
      <DataType>
        <SchemaType/>
      </DataType>
    </FormalParameter>
    <FormalParameter Id="result" Mode="OUT">
      <DataType>
        <BasicType Type="STRING"/>
      </DataType>
    </FormalParameter>
  </FormalParameters>
  <ExtendedAttributes>
    <ExtendedAttribute Name="Implementation">
      <vx:ToolAgent Class="de.danet.an.workflow.tools.XSLTTool"
        xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1">
        <vx:Property Name="XSLT" xmlns="">
          <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               version="1.0">
           <xsl:param name="city" select="'Darmstadt'"/>
           <xsl:template match="row">
             <record>
               <painter><xsl:value-of select="@painter"/></painter>
               <title><xsl:value-of select="@title"/></title>
               <year><xsl:value-of select="@year"/></year>
               <size><xsl:value-of select="@size"/></size>
               <city><xsl:value-of select="$city"/></city>
             </record>
           </xsl:template>
           <xsl:template match="table">
             <table>
               <xsl:apply-templates select="row"/>
             </table>
           </xsl:template>
          </xsl:stylesheet>
        </vx:Property>
        <vx:Property Name="Mappings">
          <vx:OutputMappings>
            <vx:Parameter Name="result" Select="/table/record[1]/painter"/>
          </vx:OutputMappings>
        </vx:Property>
      </vx:ToolAgent>
    </ExtendedAttribute>
  </ExtendedAttributes>
</Application>
...
```

As can be seen in the example, an XPath expression can be defined for each output parameter. The XPath expression is applied to the transformation result, yielding a derived result that is then assigned to the output parameter referenced in the output mappings section. The output parameters may be of `SchemaType` or `BasicType` `STRING`, `INTEGER`, `FLOAT`, `DATETIME` or `BOOLEAN`. For types datetime, float and boolean, the XPath expression must evaluate to a string that matches the format of the corresponding XSD type. Parameters of type integer are evaluated by applying the XPath `number` function to the result of the selection and using the integer part of the returned value.

As an additonal feature, the tool supports merging XML content from different input parameters before the transformation is applied.

If the first formal parameter is of type `STRING`, its value defines the tag value for a root element of the intermediate content to be created. Any additional formal input parameters of type `SchemaType` will be added under this new root element in order of appearance.

For example, calling the tool with a first input parameter "Root" and two additional input parameters `<doc1><value1>...</value1></doc1>` and `<doc2><value2>...</value2></doc2>` results in an intermediate document `<Root><doc1><value1>...</value1></doc1><doc2><value2>...</value2></doc2></Root>` to which the transformation is applied.

5.7. Generic SOAP tool

This tool can be used to execute arbitrary SOAP requests. It must be declared with a formal `IN` parameter of `SchemaType` named `body`. The XML passed into this parameter will be literally copied into the `<soapenv:Body>` section of the SOAP request. Another formal `IN` parameter of `SchemaType` named `header` may optionally be declared and can be used to add content to the `<soapenv:Header>` section of the SOAP request. Another formal `IN` parameter of type `STRING` named `httpHeaders` may optionally be declared and can be used to specify additional HTTP headers for the request. The string must consist of "key: value" pairs separated by newline characters. Note that the value of the special HTTP header "`SOAPAction`" will automatically be surrounded by double quotes, so don't use double quotes when specifying it.

Upon tool invocation, the request will be sent to the endpoint specified as a property. Output parameters may be specified to receive the complete result of the invocation (i.e. a tree with root node `<soapenv:Envelope>`) or parts of the result. The mapping of result values to output parameters is configured as described for the XSLT tool (see Section 5.6, “XSLT tool” [52]).

Example:

```
...
<Application Id="stockQuote">
  <Description>
  </Description>
  <FormalParameters>
    <FormalParameter Id="body" Mode="IN">
      <DataType>
<SchemaType/>
      </DataType>
    </FormalParameter>
    <FormalParameter Id="result" Mode="OUT">
      <DataType>
<SchemaType/>
      </DataType>
    </FormalParameter>
  </FormalParameters>
  <ExtendedAttributes>
    <ExtendedAttribute Name="Implementation">
      <vx:ToolAgent Class="de.danet.an.workflow.tools.soap.GenericSoapClient">
        <vx:Property Name="Endpoint">
          http://localhost:8181/soapsrvdemo/StockQuoteService.jws
        </vx:Property>
        <vx:Property Name="Mappings">
          <vx:OutputMappings>
            <vx:Namespaces>
              <vx:Namespace Prefix="soapenv"
                Uri="http://schemas.xmlsoap.org/soap/envelope/"/>
            </vx:Namespaces>
            <vx:Parameter Name="result" Select="/soapenv:Envelope/*"/>
          </vx:OutputMappings>
        </vx:Property>
      </vx:ToolAgent>
    </ExtendedAttribute>
  </ExtendedAttributes>
</Application>
...
```

If you are behind a firewall and want to access web services outside your intranet, it may be necessary to tell the application server (i.e. the virtual machine running it) about the proxy for your http connection. For JBoss the information can be provided as follows (see "J2SDK Networking Properties" [http://java.sun.com/j2se/1.4.2/docs/guide/net/properties.html]): `JAVA_OPTS="... -Dhttp.proxyHost=<PROXY HOST> -Dhttp.proxyPort=<PORT> -Dhttp.nonProxyHosts=localhost|*.your.domain" ./run.sh`

If calling the web service results in a `SOAPException` that indicates that the server is unavailable, the tool raises a `CannotExecuteException` with a `java.rmi.ConnectException` as its cause. This may be mapped to a process level exception as described in Section 3.5.4, “Exception handling” [31].

5.8. RPC SOAP tool

This tool can be used to execute a method of a WSDL description that defines an RPC style SOAP service. The WSDL description may either be given by a URL or an inline definition via the property `WSDL`. The discrimination between the two cases is done by checking whether the `WSDL` property defines a valid URL or not. Formal parameters to the tool invocation define the method name (first parameter) and the appropriate parameters to the method call.

Example:

```
...
<Application Id="stockQuote">
 <Description>
 </Description>
 <FormalParameters>
  <FormalParameter Id="symbol" Mode="IN">
   <DataType>
    <BasicType Type="STRING"/>
   </DataType>
  </FormalParameter>
  <FormalParameter Id="Result" Mode="OUT">
   <DataType>
    <BasicType Type="FLOAT"/>
   </DataType>
  </FormalParameter>
 </FormalParameters>
 <ExtendedAttributes>
  <ExtendedAttribute Name="Implementation">
   <vx:ToolAgent Class="de.danet.an.workflow.tools.soapclient.SOAPClient"
    xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1">
    <vx:Property Name="WSDL">
     <wsdl:definitions
  targetNamespace="http://brian.an.danet.de:8080/soapsrvdemo/StockQuoteServi
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:apachesoap="http://xml.apache.org/xml-soap"
  xmlns:impl="http://brian.an.danet.de:8080/soapsrvdemo/StockQuoteService.jw
  xmlns:intf="http://brian.an.danet.de:8080/soapsrvdemo/StockQuoteService.jw
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
     <wsdl:message name="getQuoteResponse">
      <wsdl:part name="getQuoteReturn" type="xsd:float"/>
    </wsdl:message>
    <wsdl:message name="getQuoteRequest">
      <wsdl:part name="symbol" type="xsd:string"/>
    </wsdl:message>
    <wsdl:portType name="StockQuoteService">
      <wsdl:operation name="getQuote" parameterOrder="symbol">

        <wsdl:input message="impl:getQuoteRequest" name="getQuoteRequest"/>
        <wsdl:output message="impl:getQuoteResponse" name="getQuoteResponse"
      </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="StockQuoteServiceSoapBinding"
     type="impl:StockQuoteService">
      <wsdlsoap:binding style="rpc"
       transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="getQuote">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="getQuoteRequest">
         <wsdlsoap:body
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
          namespace="http://DefaultNamespace" use="encoded"/>
         </wsdl:input>
        <wsdl:output name="getQuoteResponse">
          <wsdlsoap:body
         encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    namespace="http://brian.an.danet.de:8080/soapsrvdemo/StockQuoteService.jws"
          use="encoded"/>
        </wsdl:output>
      </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="StockQuoteServiceService">
      <wsdl:port binding="impl:StockQuoteServiceSoapBinding"
       name="StockQuoteService">
        <wsdlsoap:address
         location="http://localhost:8080/soapsrvdemo/StockQuoteService.jws"/>
      </wsdl:port>
    </wsdl:service>
   </wsdl:definitions>
    </vx:Property>
    <vx:Property Name="Method">getQuote</vx:Property>
   </vx:ToolAgent>
  </ExtendedAttribute>
 </ExtendedAttributes>
</Application>
...
```

The hints about firewalls described in Section 5.7, “Generic SOAP tool” [54] apply to this tool as well.

When calling the web service results in a `java.rmi.RemoteException`, the tool raises a `CannotExecuteException` with the `RemoteException` as its cause. In addition, the tool establishes an exception mapping of `java.rmi.RemoteException` to a process level exception "`RemoteException`" (see Section 3.5.4, “Exception handling” [31]). This reflects the assumption that the unavailability of the web service is a condition that is to be handled at the process level and cannot be worked around by simply repeating the invocation (as is usually done with `RemoteExceptions`).

5.9. Wait tool

Sometimes, further processing in a workflow process must be delayed until some date is reached. This can be achieved with the WaitTool.

In its simplest form, the wait tool is defined as:

```
...
<Application Id="SimpleWait">
 <Description>Waits for a timer to expire</Description>
  <FormalParameters>
    <FormalParameter Id="waitUntil" Mode="IN">
      <DataType>
        <BasicType Type="DATETIME"/>
      </DataType>
    </FormalParameter>
  </FormalParameters>
  <ExtendedAttributes>
    <ExtendedAttribute Name="Implementation">
      <vx:ToolAgent Class="de.danet.an.workflow.tools.timing.WaitTool"
        xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1"/>
    </ExtendedAttribute>
  </ExtendedAttributes>
</Application>
...
```

When used in this way, the tool will simply wait until the given date/time. Note that waiting is not implemented by calling something like "sleep" for the execution thread. Rather a persistent timer is created. So it is safe to specify a date next week, next month or even in years[5]. A running wait tool will continue to run even after server restarts.

As an alternative, the tool may also be invoked with a single parameter of type `INTEGER` or `DOUBLE` instead of `DATETIME`. In this case, it waits for the specified number of seconds to pass after its invocation.

In some workflows, you need the possibility to end the waiting prematurely. For these cases another version of the tool is provided that actually consists of three tools. The first tool creates a timer, returning a timer id. The second tool waits until the timer has expired (i.e. the date/time specified on creation has been reached). And the third tool may optionally be used to terminate the timer prematurely. In order to be able to distinguish regular and premature completion easily, the wait tool is given an `OUT` parameter that is set to `EXPIRED` (regular termination) or `CANCELED` (premature termination).

Example:

```
...
<Application Id="TimerCreator">
  <Description>Creates a timer</Description>
  <FormalParameters>
    <FormalParameter Id="timerId" Mode="OUT">
      <DataType>
        <BasicType Type="INTEGER"/>
      </DataType>
    </FormalParameter>
    <FormalParameter Id="waitUntil" Mode="IN">
      <DataType>
        <BasicType Type="DATETIME"/>
      </DataType>
    </FormalParameter>
  </FormalParameters>
  <ExtendedAttributes>
    <ExtendedAttribute Name="Implementation">
      <vx:ToolAgent Class="de.danet.an.workflow.tools.timing.TimerCreator"
        xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1"/>
    </ExtendedAttribute>
  </ExtendedAttributes>
</Application>
<Application Id="TimerCanceler">
  <Description>Cancels a timer</Description>
  <FormalParameters>
    <FormalParameter Id="timerId" Mode="IN">
      <DataType>
        <BasicType Type="INTEGER"/>
      </DataType>
    </FormalParameter>
  </FormalParameters>
  <ExtendedAttributes>
    <ExtendedAttribute Name="Implementation">
      <vx:ToolAgent Class="de.danet.an.workflow.tools.timing.TimerCanceler"
        xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1"/>
    </ExtendedAttribute>
  </ExtendedAttributes>
</Application>
<Application Id="Timing">
  <Description>Waits for a timer to expire</Description>
  <FormalParameters>
    <FormalParameter Id="timerId" Mode="IN">
      <DataType>
        <BasicType Type="INTEGER"/>
      </DataType>
    </FormalParameter>
    <FormalParameter Id="status" Mode="OUT">
      <DataType>
        <BasicType Type="STRING"/>
      </DataType>
```

[5]Consider a process contolling the complete lifetime cycle of a document. After all the initial creating, reviewing, revising and using the document, it may be necessary to archive the document for another 5 years before deleting it.

```
    </FormalParameter>
  </FormalParameters>
  <ExtendedAttributes>
    <ExtendedAttribute Name="Implementation">
      <vx:ToolAgent Class="de.danet.an.workflow.tools.timing.WaitTool"
        xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1"/>
    </ExtendedAttribute>
  </ExtendedAttributes>
</Application>
...
```

5.10. MBean invocation tool

This tool supports the invocation of an MBean operation.

The tool is declared as:

```
...
<Application Id="SimpleInvoker">
  <Description>Simple MBean invoker test</Description>
  <FormalParameters>
    <FormalParameter Id="result" Mode="OUT">
      <DataType>
        <BasicType Type="STRING"/>
      </DataType>
    </FormalParameter>
  </FormalParameters>
  <ExtendedAttributes>
    <ExtendedAttribute Name="Implementation">
      <vx:ToolAgent Class="de.danet.an.workflow.tools.MBeanInvoker">
        <vx:Property Name="ObjectName">jboss:service=JNDIView</vx:Property>
        <vx:Property Name="Operation">listXML</vx:Property>
      </vx:ToolAgent>
    </ExtendedAttribute>
  </ExtendedAttributes>
</Application>
...
```

The tool looks up the MBean set in property `ObjectName` and invokes its operation `Operation`.

If the first declared formal parameter has mode `OUT` or `INOUT` the result of the operation invocation will be assigned to the actual parameter. The parameter's type must match the return type of the MBean's operation.

The second and subsequent parameters must have mode `IN` or `INOUT`. All parameters with mode `IN` or `INOUT` (i.e. including the first) are passed to the invoked operation. The parameter types are used to build the signature used for operation lookup.

5.11. EJB invocation tool

This tool supports the invocation of an EJB operation.

The tool is declared as:

```
  ...
<Application Id="EJBTool">
    <FormalParameters>
        <FormalParameter Id="JndiName" Mode="IN">
            <DataType>
                <BasicType Type="STRING"/>
            </DataType>
        </FormalParameter>
        <FormalParameter Id="HomeClass" Mode="IN">
            <DataType>
                <BasicType Type="STRING"/>
            </DataType>
        </FormalParameter>
        <FormalParameter Id="Method" Mode="IN">
            <DataType>
                <BasicType Type="STRING"/>
            </DataType>
        </FormalParameter>
        <FormalParameter Id="Num1" Mode="IN">
            <DataType>
                <BasicType Type="INTEGER"/>
            </DataType>
        </FormalParameter>
        <FormalParameter Id="Num2" Mode="IN">
            <DataType>
                <BasicType Type="INTEGER"/>
            </DataType>
        </FormalParameter>
        <FormalParameter Id="AddResult" Mode="OUT">
            <DataType>
                <BasicType Type="INTEGER"/>
            </DataType>
        </FormalParameter>
    </FormalParameters>
    <ExtendedAttributes>
        <ExtendedAttribute Name="Implementation">
            <vx:ToolAgent
              xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1"
              Class="de.danet.an.workflow.tools.EJBTool"/>
        </ExtendedAttribute>
    </ExtendedAttributes>
</Application>
...
```

The tool looks up the EJB with the JNDI name `JndiName` and the home interface `HomeClass` and invokes the operation `Method`.

If the last declared formal parameter has mode `OUT` or `INOUT` the result of the operation invocation will be assigned to the actual parameter. The parameter's type must match the return type of the MBean's operation.

The fourth and subsequent parameters must have mode `IN` or `INOUT`. All parameters with mode `IN` or `INOUT` (i.e. including the first three) are passed to the invoked operation. The parameter types are used to build the signature used for operation lookup.

5.12. Channel based access

Channel based access[6] provides channels for message oriented communication between a process instance and a client or between a process instance and its sub-processes.

Channels are named and associated with a process instance. They need not be declared. Clients can obtain access to a channel from the `WorkflowService` (see getChannel(de.danet.an.workflow.omgcore.WfProcess, java.lang.String) [246]). The channel may then be used to send messages to a receiver tool or receive messages from a sender tool (see below). Messages are simply passed from source to destination; there is no verification that a message from a client to a process (i.e. to a receiver tool) is followed by a message from the process (i.e. from a sender tool) to the client or vice versa. The communication may be initiated by any party: the process may create (or know about) a client and intially send a message to it, or a client may create or lookup a process that has started (or will start) a receiver tool and send a message to it (see Sec-

[6]"ChaBAcc" for short. We don't know how this sounds in other languages, but Germans usually tend to grin at the sound of this acronym for no specific reason.

tion 5.12.3, “Generic HTTP Access” [61] for a sample usage).

5.12.1. Receiver tool

The receiver tool receives messages from a channel. It takes an `IN` parameter of type `STRING` that specifies the channel to listen on, and an arbitrary number of additional `OUT` parameters that receive the data sent on the channel. The formal parameter names must match the keys used in the map passed to the channel by the client (see sendMessage(java.util.Map) [174]).

Normally, the receiver receives messages sent to the process it was started in. Sometimes, however, it is useful to structure a workflow process by using subflows. If the activities separated into the subflow include the usage of a receiver tool, the receiver tool listens on the wrong process, as the client does not (and should not) know about the separation in main process and subflow and keeps sending the messages to the main process.

The receiver tool therefore has a property "`ListenerIndex`" that can be used to make the receiver listen for messages sent to one of the requesting processes of the current process (i.e. processes the current process is a subflow of). If set, the property must be an integer. 0 (zero) is the top requester, i.e. the process at the end of the chain of requesting processes[7]. A positive integer specifies a process relative to the current process (e.g. "1" is the requester, "2" is the requester of the requester) while a negative integer specifies a subflow relative to the top process.

Example:

```
...
    <Application Id="TopReceiverTool">
      <Description>Receive data from a channel accessor</Description>
      <FormalParameters>
        <FormalParameter Id="channel" Mode="IN">
          <DataType>
            <BasicType Type="STRING"/>
          </DataType>
        </FormalParameter>
        <FormalParameter Id="message" Mode="OUT">
          <DataType>
            <BasicType Type="STRING"/>
          </DataType>
        </FormalParameter>
      </FormalParameters>
      <ExtendedAttributes>
        <ExtendedAttribute Name="Implementation">
          <vx:ToolAgent Class="de.danet.an.workflow.tools.chabacc.Receiver"
            xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1">
            <vx:Property Name="ListenerIndex">0</vx:Property>
          </vx:ToolAgent>
        </ExtendedAttribute>
      </ExtendedAttributes>
    </Application>
...
```

5.12.2. Sender tool

The sender tools sends messages on a channel. It takes an `IN` parameter of type `STRING` that specifies the channel to send to, and an arbitrary number of additional `IN` parameters used to pass the data to be sent to the channel. The formal parameter names are used as keys in the map returned to the client (see receiveMessage() [174]). Note that actual parameters are sent on the channel as passed to the sender tool. This implies that parameters of type `SchemaType` may be sent as SAX event buffer, W3C DOM or JDOM, depending on the value of attribute `XMLParameterMode` in the definition of the sender tool (see Section 4.2.5.1, “Extentions of Application Declaration” [35]).

Like the receiver tool, the sender tool may be configured to send messages from a requesting pro-

[7]This may also be used if the process is not a subflow at all, as a process without a requesting process is considered top of the chain.

cess instead of the process the tool was started in. This is done by setting the property "`OriginatorIndex`". The value is interpreted in the same way as described for the receiver tool above.

An additional property of the sender tool specifies whether messages sent should also be delivered to receiver tools listening on the channel used (by default, they are only delivered to clients). This may e.g. be used at the end of a process to terminate a receiver tool running "in parallel" (i.e. it has waited for some kind of optional intervention from a client during processing and is now sent some dummy message to complete the process properly). Together with the "`ListenerIndex`" and "`OriginatorIndex`" properties, this may also be used for communication between processes, e.g. a subflow informing its requester about its state.

Example:

```
...
    <Application Id="TopSenderTool">
      <Description>Send a message to a channel accessor</Description>
      <FormalParameters>
        <FormalParameter Id="channel" Mode="IN">
          <DataType>
            <BasicType Type="STRING"/>
          </DataType>
        </FormalParameter>
        <FormalParameter Id="message" Mode="IN">
          <DataType>
            <BasicType Type="STRING"/>
          </DataType>
        </FormalParameter>
      </FormalParameters>
      <ExtendedAttributes>
        <ExtendedAttribute Name="Implementation">
          <vx:ToolAgent Class="de.danet.an.workflow.tools.chabacc.Sender"
            xmlns:vx="http://www.an.danet.de/2009/XPDL-Extensions1.1">
            <vx:Property Name="LocalDelivery">True</vx:Property>
            <vx:Property Name="OriginatorIndex">0</vx:Property>
          </vx:ToolAgent>
        </ExtendedAttribute>
      </ExtendedAttributes>
    </Application>
...
```

5.12.3. Generic HTTP Access

To simplify the HTTP access to a longterm workflow process, as it may be useful for designing dialog driven applications, an "out-of-the-box" solution is provided by a generic HTTP servlet.

For an example, let us assume we'd like to design an application that enables users to subscribe and unsubscribe a service and to modify the service parameters. The user interface should be provided by HTML pages and the service will be described by a workflow process definition[8].

The components we need are:

- An entry Web page for subscribing to a new service or requesting an existing service.
- A process definition within the workflow engine, describing the actions of the service and providing HTML pages for user actions (e.g. modification of service parameters).
- A (user based) connection between the workflow engine (i.e. a service) and the client HTML pages (i.e. the browser).

[8]We use an HTML browser as example here in order to make the explanation easy to follow. Of course, this mechanism becomes more thrilling if you imagine a voice browser instead of the HTML browser and VoiceXML instead of HTML. We have used the generic servlet in such a way and built some nice workflow based voice applications with WfMOpen.

Subscribing to a new service is easy to implement, since it may simply be done by starting a new workflow process. But once the process is running, arrangements have to be made to make sure that the process performs its service duties and concurrently listens for service "update" events. The latter task can be performed using the Receiver Tool (see Section 5.12.1, “Receiver tool” [60]) and the Sender Tool (see Section 5.12.2, “Sender tool” [60]) within a concurrently running activity.

To simplify the (channel based) interaction with workflow processes via HTTP, a servlet is provided that

- starts a new process — if needed — and provides a channel to it or
- opens a channel to an already running process and
- holds the channel within the session for further use and forwards responses from the process to the client.

By using this servlet, HTTP based client applications (e.g. using HTML pages) are very easy to design. The current page sends a request to the servlet and the successor page (code) is provided by the proper workflow activity through the servlet. No further application code is needed.

The servlet described above serves as an interface between the client application and a workflow process. The communication with the process uses a message channel named "`initiator`") [9].

To identify the process to connect with, some key information has to be provided within the request. The following request parameters are treated as key values:

- `WFM_packageID` and `WFM_processID`: package ID and process ID of the associated process
- `WFM_dataItemName` and `WFM_dataItemValue`: name and value of data item (data field) that should be used as a search criteria for a process of the specified type (i.e. with the given package and process ID).

All remaining request parameters are considered to be application data and are used to initialize the process' data or send to the receiver tool.

Variations:

1. Only WFM_packageID and WFM_processID (beside application data) are provided.

   A new process of the specified type is created and initialized with the application data. A new message channel is built up to this process. The process manager name and process key are stored in the session context for later use (see variation 3 [63]).

   If no process description with the given package and process id exists, status code `400` will be returned.

2. All above-mentioned WFM* parameters are provided. A lookup is performed for the process matching the given data item criteria. If none is found, a new process of the specified type is created and initialized with the given data item and the provided application data. The process manager name and process key are stored in the session context for later use (see variation 3 [63]). If no process description with the given package and process id exists, status code `400` will be returned.

   If a matching process is found, this process is used. The provided application data is sent to the process via a built-up message channel.

   If no message channel can be opened because the process is closed, status code `410` will be returned.

[9] So you have to make sure, that the SenderTool and the ReceiverTool use a channel named `initiator` as well.

If more than one matching process is found, status code `409` will be returned.

3. WFM_packageID and WFM_processID are not provided. It is assumed that the process should be used which has already been referenced within the session (see variation 1 [62] or 2 [62]). Thus, the process manager name and process key that have been stored in the session context are used to build the connection to the process. The provided application data is sent to the process via the built-up message channel.

   If no message channel can be opened because the process ist either closed or removed, status code `410` will be returned.

   If no process information is stored in the session context, status code `400` will be returned.

4. An additional parameter WFM_waitForResponse is provided with a value of "`false`" (not equal to "`true`", to be precise). In this case, the servlet will not wait for a response from the process. It will only send data to a process or create a new process as described above and then return a response with no content.

The message received on the channel from the process (sent by the process using the sender tool, see Section 5.12.2, "Sender tool" [60]) may have one or more entries. If it has only a single entry, this entry is considered to be the data to be sent back to the HTTP client, independant of the entry's name. If the message has more than one entry, there must be one entry with key `data` containing the data to be sent back or otherwise, status `500` will be returned to the client. The data (i.e. the corresponding formal parameter of the SenderTool) may be of type `STRING` or `SchemaType`. In the latter case, the XML data is transformed to its string representation by the generic HTTP servlet before it is sent to the HTTP client. Note that the data must be provided as a SAX buffer, i.e. the sender tool defined for sending the response to the servlet must have the attribute `XMLParameterMode` set to `USE_SAX` (see Section 4.2.5.1, "Extentions of Application Declaration" [35]).

Additional entries supported in a message received by the generic HTTP servlet are:

`doctype-system` (`STRING`)	Specifies the system identifier to be used in the document type declaration of the document resulting from the conversion of response data passed as XML to its string representation.
`doctype-public` (`STRING`)	Specifies the public identifier to be used in the document type declaration of the document resulting from the conversion of response data passed as XML to its string representation.
`invalidateSession` (`BOOLEAN`)	When set to true, the session associated with the request is invalidated. This may be used to dissociate a client from its associated process (if they are linked using information from the session context as described in 3 [63]), thus forcing the creation of a new process on the next request.
`mimeType` (`STRING`)	When set, overrides automatic derivation of the response's mime type from the data to be sent. The default is to send any data of type `SchemaType` as `text/xml`. If the data is of type `STRING` and the string starts with a `"<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML"` or `"<!DOCTYPE HTML PUBLIC \"-//IETF//DTD HTML"` declaration, it is sent as `text/xml`. Otherwise the mime type ist set to `application/data`.

The generic HTTP servlet will accept `POST` requests only, as all requests lead to a modification of the server side state. There is, however, one exception to this rule.

In order to get started with a process based Web application, you must present the user an initial page that allows him to click on a link to create a new process or to fill in and submit some information that identifies an existing process. Now, where does this start page come from? Of course, it can simply be created as static HTML and provided by any Web server (and for performance reasons, this will eventually be the solution in many cases). However, even if only as proof of concept, it should be possible to specify a complete Web application — including the start page — using XP-DL.

As initial URLs (i.e. URLs that can be typed into a browser) will always perform a `GET` request, the generic HTTP servlet must support such a request for generating the start page. As this is a "get started" request, it will always create a new process, whose single purpose is to send back the content of the start page and then complete [10] (although there may be cases that allow to continue the lifespan of this process). To avoid complicated looking URLs, the generic HTTP servlet uses extra path information instead of parameters to specify the type (i.e. package id and process id) of the process to be created. The first segment in the extra path information denotes the package id, the second the process id. If the process id is omitted, it defaults to `boot`, if the package id is omitted also, it defaults to `chabacc_http_plain`. Both defaults may be altered by specifying the servlet init parameters `packagePathSegmentDefault` and `processPathSegmentDefault` respectively.

Some examples of using the generic HTTP servlet are included in the demo applications (see Appendix C, *The demo applications* [337]).

[10]Admittedly, this is less efficient than a static HTML page, but you need not set up a server for static HTML pages. It is a bit like a servlet serving static files: obviously Web servers like Apache can do it faster, but in many cases you want to avoid the effort of installing Apache or adding to its configuration only to provide a few files as part of your J2EE application.

Chapter 6. The sample resource assignment service

For assigning resources to activities, a workflow component relies on a resource assignment facility as described in Section 3.1, “Component structure” [23] and Section 3.4, “Resource assignment SPI” [30]. Such a facility is not part of the core workflow functionality.

As we felt our workflow component to be incomplete without at least a simple resource assignment service, we have included an implementation of such a service in the workflow component.

6.1. The sample assignment service

The sample assignment service implements the resource assignment API as specified by package de.danet.an.workflow.spis.ras [269]. The factory class and its configuration parameters are described in de.danet.an.workflow.assignment.StandardResourceAssignmentServiceFactory [327]. See the classes description for further information not covered in the current section.

The sample resource assignement service is bundled in the library `de.danet.an.workflow.defaultras.jar`. Besides the required classes, this jar contains the file `META-INF/services/de.danet.an.workflow.spis.ras.ResourceAssignmentServiceFactory` that defines the class `StandardResourceAssignmentServiceFactory` as an implementation of the resource assignment service. Thus the sample resource assignment service is found as described in newInstance() [280] if the jar is included in the classpath of the EAR that comprises the application. Usually, this is done by adding the jar as Java module to `application.xml`.

6.2. Provided functionality

The sample resource assignment service uses the participant type and participant name from the process definition to select a resource from a resource management service (RMS, see below). If the participant type is `HUMAN`, it looks for a user with the given name, if it is `RESOURCE_SET`, it looks for a group with the given name, and it it is `ROLE` it looks for a role with the given name.

For selecting the resource from the RMS, the sample RAS relies on the RMS to support queries of the form "M:*`user_name`*", "G:*`group_name`*" and "R:*`role_name`*" (see below).

As a special case, if the participant type is `HUMAN` and the extended attribute "`resource-selection`" exists and its value is "`!:currentUser`", then the activity is assigned to the user that has created the process. The participant definition thus looks like:

```
<Participant Id="currentUser" Name="Current User">
  <ParticipantType Type="HUMAN" />
  <ExtendedAttributes>
    <ExtendedAttribute Name="resource-selection"
                       Value="!:currentUser"/>
  </ExtendedAttributes>
</Participant>
```

As the syntax "`!:currentUser`" suggests, there had been plans to specify sophisticated selection rules here. Currently, we think that any usage of the workflow engine that needs more sophisticated assignment than selecting a person, a group or the process creator will need its own implementation of a RAS anyway.

6.3. The underlying resource management service

Thinking about resource assignment, it soon becomes obvious that a resource assignment service must know about the resources it can assign and thus must include or be based on a resource management service.

Resource management, however, is not a trivial task. As such, we cannot provide a useful resource management, even as a sample implementation, as part of the workflow component. In order to be nevertheless able to provide a sample resource assignment service, we have defined a formal interface to a resource management service in package de.danet.an.workflow.spis.rms [281]. Note that only the sample resource assignment service depends on this interface, not the workflow engine. If you decide to replace the sample resource assignment service with your own implementation, you no longer have to provide a resource management service as defined by the resource management service interface (of course, you may use the interface in your implementation of a resource assignment service if it suits your needs).

A resource management service must implement a resource management factory and a resource management service as defined in Section A.5.9, “Class ResourceManagementServiceFactory” [294] and Section A.5.8, “Interface ResourceManagementService” [291]. An implementation of `ResourceManagementFactory` may require additional configuration properties to adapt the service instances that it creates to the specific environment. Usually, these properties are retrieved from JNDI environment entries or from a Java properties file.

The WfMOpen implementation guarantees that the resource management service factory is only used in the context of the `WorkflowEngineEJB`. Therefore, if you use JNDI environment entries to configure your factory, it is sufficient to add those entries to the `WorkflowEngineEJB`. Note that the guarantee to use the resource management service factory in this context only is implementation specific. Therefore the general SPI description leaves the issue of which EJBs require the entries deliberately open.

As has been described above, any RMS that is to be used with the sample resource assignment service must support the queries "M:*`user_name`*", "G:*`group_name`*" and "R:*`role_name`*" as parameter of the `selectResources` method (see selectResources(java.lang.Object) [293]).

6.4. Provided RMS implementations

The WfMOpen distribution provides several implementations of a resource management service. These are described in the following sections.

6.4.1. Database based RMS

Unless a standards based approach is used (see below) user information is most likely kept in an RDBMS. WfMOpen therefore includes a generic implementation of a database based RMS that can be adapted to a wide range of database schemas.

The detailed description of this service and its configuration parameters can be found in de.danet.an.workflow.rmsimpls.dbrms [329].

The factory and service classes are bundled in the `de.danet.an.workflow.dbrms.jar` which can be found in `$DIST/lib/wfcore/`. Besides the required classes, this jar contains the file `META-INF/services/de.danet.an.workflow.spis.rms.ResourceManagementServiceFactory` that defines the class `de.danet.an.workflow.rmsimpls.dbrms.DatabaseRmsFactory` as an implementation of the resource management service. Thus the database based resource management service is found as described in newInstance() [295] if the jar is included in the classpath of the EAR that comprises the application. Usually, this is done by adding the jar as Java module to `application.xml`.

The service must be configured by specifying the statements for the various resource queries. This is decribed in detail in de.danet.an.workflow.rmsimpls.dbrms.DatabaseRmsFactory [329].

6.4.2. EIS based RMSes

In general, information about resources is often kept in Enterprise Information Systems (EIS). The preferred approach for accessing this kind of systems in J2EE is by using JCA. WfMOpen therefore provides a resource management system implementation that relies on a JCA datasource to obtain information about resources.

6.4.2.1. The generic EIS RMS front-end

The implementation uses a generic RMS implemantation that must be combined with a specific resource adapter. The description of this EIS RMS service and its configuration parameters can be found in de.danet.an.workflow.rmsimpls.eisrms [331].

The factory and service classes are bundled in the `de.danet.an.workflow.eisrms.jar` which can be found in `$DIST/lib/wfcore/`. Besides the required classes, this jar contains the file `META-INF/services/de.danet.an.workflow.spis.rms.ResourceManagementServiceFactory` that defines the class `de.danet.an.workflow.rmsimpls.eisrms.EisRmsFactory` as an implementation of the resource management service. Thus the EIS based resource management service is found as described in newInstance() [295] if the jar is included in the classpath of the EAR that comprises the application. Usually, this is done by adding the jar as Java module to `application.xml`.

The EIS RMS implements only the "front-end" of the resource management system. It relies on a resource adapter that implements a specific client API that is used by the EIS RMS to provide its services. While it might have been possible to use the standard CCI (Common Client Interface) as API of the resource adapter, we found that this would have caused considerable overhead without providing a significant benefit. We have therefore defined a specific application client interface (ACI) in de.danet.an.workflow.rmsimpls.eisrms.aci [332].

6.4.2.2. Resource adapter implementations

The following sections describe the resource adapter implementation for the EIS RMS that are distributed with WfMOpen. Each of these adapters can be found as `de.danet.an.workflow.*ra.rar` in the directory `$DIST/lib/wfcore/` of the distribution.

The resource adapters follow standard packaging rules and can be deployed in any JCA compliant application server.

6.4.2.2.1. Properties based resource adapter

This resource adapter obtains its information from three properties files. The first is used to describe users. Entries have the format "*`user`*=*`password`*". The password is without significance for the adapter. The format has been chosen to mach the requirements of the file based authentication module of JBoss. This allows the file to be used for both authentication and resource definition.

The second and third file describe roles and groups respectively. The format for entries is "*`user`*=*`role1`*, *`role2`*, ..." (or "*`user`*=*`group1`*, *`group2`*, ..."). Again, the file describing the roles can also be used for role management in the file based authentication module of JBoss.

The files are made known to the resource adapter in its configuration. The properties to specify for deployment are:

`PropertiesDirUrl`	The directory that hold the properties files.

`UsersPropertiesFile`	The file that defines the users.
`RolesPropertiesFile`	The file that defines the roles.
`GroupsPropertiesFile`	The file that defines the groups.

A typical deployment configuration file for JBoss is shown below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- The properties based RMS resource adaptor service configuration -->
<connection-factories>
  <no-tx-connection-factory>
    <jndi-name>DefaultWfMOpenRmsRA</jndi-name>
    <rar-name>
      de.danet.an.plutoportalapp.ear#de.danet.an.workflow.propsrmsra.rar
    </rar-name>
    <connection-definition>
      de.danet.an.workflow.rmsimpls.eisrms.aci.RmsConnectionFactory
    </connection-definition>
    <config-property name="PropertiesDirUrl" type="java.lang.String">
      ${jboss.server.config.url}
    </config-property>
    <config-property name="UsersPropertiesFile" type="java.lang.String">
      wfdemopluto-users.properties
    </config-property>
    <config-property name="RolesPropertiesFile" type="java.lang.String">
      wfdemopluto-roles.properties
    </config-property>
    <config-property name="GroupsPropertiesFile" type="java.lang.String">
      wfdemopluto-groups.properties
    </config-property>
  </no-tx-connection-factory>
</connection-factories>
```

Figure 6.1. Sample resource adapter configuration for JBoss

Note that JBoss requires that the name of the RAR is qualified with the name of the EAR that contains the RAR. You must therefore adapt the deployment configuration to the name of your EAR.

At the first glance it might seem overkill to implement the file based resource management as a resource adapter using the JCA. But if you take a closer look, you'll find that even a read-only file database is best modelled as an EIS in the J2EE architecture. And it's the easiest way to push the configuration options to your application server's management interface.

6.4.2.2.2. The LDAP based resource adapter

The most likely found, standard based resource management facility in an organization is an LDAP server. The WfMOpen distribution therefore includes an implementation of an LDAP based resource adapter for the resource management service.

All information required to access the LDAP server is made known to the resource adapter in its configuration. The properties to specify for deployment are:

`JavaNamingFactoryInitial`	The value of `java.naming.factory.initial` used when creating the initial context. Defaults to `com.sun.jndi.ldap.LdapCtxFactory`.
`JavaNamingSecurityAuthentication`	The value of `java.naming.security.authentication` used

	when creating the initial context. Defaults to `simple`.
`JavaNamingProviderUrl`	The value of `java.naming.provide.url` used when creating the initial context. Defaults to `ldap://localhost:389`.
`JavaNamingSecurityPrincipal`	The value of `java.naming.security.principal` used when creating the initial context. Optional, i.e. need not be specified if your LDAP server allows anonymous access.
`JavaNamingSecurityCredentials`	The value of `java.naming.security.credentials` used when creating the initial context. Only used when `JavaNamingSecurityPrincipal` has been set.
`UserCtxDN`	The DN of the directory entry that contains all known users.
`UserFilter`	A filter expression applied when listing all known users in the directory specified by `UserCtxDN`.
`UserSearchAttribute`	The user entry's attribute used when searching for an entry that matches the principal name or a criterion in member resource selection. Defaults to `uid`.
`UserResourceNameAttribute`	The user entry's attribute used as resource display name. Defaults to `cn`.
`UserMemberKeyAttribute`	When searching for users that are members of groups or roles, an attribute of the groups' or roles' entries is by default compared to the user entry's DN. If, however, a user member key attribute is specified, the value of this attribute is read from the user entry, and this value is used for comparision instead of the user entry's DN.
`GroupCtxDN`	The DN of the directory entry that contains all known groups.
`GroupFilter`	A filter expression applied when listing all known groups in the directory specified by `GroupCtxDN`.
`GroupSearchAttribute`	The group entry's attribute used when searching for an entry that matches a criterion in group resource selection. Defaults to `cn`.
`GroupResourceNameAttribute`	The group entry's attribute used as resource display name. Defaults to `cn`.
`GroupMemberAttribute`	When searching for users that are members of groups, either the user entry's DN or the value of an attribute from the user entry (see above) is compared to an attribute of all role entries. The role attribute used is specified with this property.
`RoleCtxDN`	The DN of the directory entry that contains all known roles.
`RoleFilter`	A filter expression applied when listing all known roles in the directory specified by `UserCtxDN`.
`RoleSearchAttribute`	The role entry's attribute used when searching for an entry that matches a criterion in role resource selection. Defaults to `cn`.
`RoleResourceNameAttribute`	The role entry's attribute used as resource display name. Defaults to `cn`.
`RoleMemberAttribute`	When searching for users that are members of roles, either the user entry's DN or the value of an attribute from the user

entry (see above) is compared to an attribute of all role entries. The role entry's attribute used is specified with this property.

A sample deployment configuration is shown below.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!-- The properties based RMS resource adaptor service configuration -->
<connection-factories>
  <no-tx-connection-factory>
    <jndi-name>DefaultWfMOpenRmsRA</jndi-name>
    <rar-name>
      de.danet.an.plutoportalapp.ear#de.danet.an.workflow.ldaprmsra.rar
    </rar-name>
    <connection-definition>
      de.danet.an.workflow.rmsimpls.eisrms.aci.RmsConnectionFactory
    </connection-definition>
    <config-property name="UserCtxDN" type="java.lang.String">
      ou=People,dc=my-domain,dc=com
    </config-property>
    <config-property name="GroupCtxDN" type="java.lang.String">
      ou=ResourceGroups,dc=my-domain,dc=com
    </config-property>
    <config-property name="RoleCtxDN" type="java.lang.String">
      ou=ResourceRoles,dc=my-domain,dc=com
    </config-property>
  </no-tx-connection-factory>
</connection-factories>
```

Figure 6.2. Sample resource adapter configuration

Note that JBoss requires that the name of the RAR is qualified with the name of the EAR that contains the RAR. You must therefore adapt the deployment configuration to the name of your EAR.

Chapter 7. Wf-XML

Dirk Schnelle, Danet GmbH
Dr. Michael Lipp, Danet GmbH

WfMOpen has full support for Wf-XML 2.0 as proposed by the WfMC to enable remote clients to access WfMOpen using SOAP requests. There are only minor deviations that are described below. This documentation is not meant to be a tutorial for Wf-XML. It is assumed that the reader is familiar with the ASAP and Wf-XML standards.

The implementation has successfully been tested with the generic ASAPClient that had been made available by the WfMC when Wf-XML was initially published. We do not know of any other means to test standard compliance and are therefore very interested in getting your feedback if you are testing interoperability.

7.1. Installation

The Wf-XML interface is packaged as a separate web application component and distributed in `$DIST/lib/wfcore/de.danet.an.workflow.wfxml.war`. In order to use the interface, this WAR must be configured and included in your EAR. The demo application (see Appendix C, *The demo applications* [337]) includes the interface, i.e. you can use the demo application to get acquainted with the interface. The base url for accessing the interface in the demo is `http://localhost:8080/wfxml`.

7.2. Accessing Wf-XML Resources

The central instance that is responsible to handle all requests is the `de.danet.an.workflow.clients.wfxml.Servlet`. The different resource types are identified through the `ReceiverKey` in the ASAP specific header. The `ResponseRequired` tag is currently ignored. A response is returned in any case.

The `ReceiverKey` is a URI with the following structure `<schema:>//<authority>/<base_path>[?<query>]` The parts up to `base_path` describe the deployment context of the servlet (e.g. `http://localhost:8080/wfxml`).

All parts that are needed to identify a specific resource are encoded in the `query` part of the URI as key-value pairs. If the query is omitted, the request is assumed to be directed at the `Service Registry`. The following table shows how specific resources are addressed with query parameters.

Table 7.1. Resource identification through the ReceiverKey

Resource	ProcessId	PackageId	ProcessKey	ActivityKey
ServiceRegistry	n/a	n/a	n/a	n/a
Factory	required	required	n/a	n/a
Instance	required	required	required	n/a
Activity	required	required	required	required

A request to e.g. the `Factory` for the process `proc` that is described in package `pkg` is made using the following `ReceiverKey` (assuming the base path above): `http://localhost:8080/wfxml?Resource=Factory&PackageId=pkg&ProcessId=proc`

Note that the values for `PackageId` and `ProcessId` must be www-form-urlencoded `UTF-8` charset identifiers.

Of course, the usual procedure is to obtain resource references by requesting listings from the parent resources, starting with the `Service Registry`. In some cases, however, the direct approach may be required (e.g. the above mentioned ASAPClient starts at the `Factory` level, i.e. you have to enter the resource reference to a factory).

The names of the states, as specified by ASAP, are enhanced to match the state names as they are used by WfMOpen. These extended state names have the ASAP prefix followed by a "." and the substate as it is returned by the OMG API.

7.3. Properties

ASAP specifies, that `SetPropertiesRq` calls need not support the modification of all properties that are given, but at least one to have any effect. Properties that are specified in the request, but can not be altered are ignored without warning.

WfMOpen supports modification of the following properties:

- ServiceRegistry
  - No modifications possible
- Factory
  - No modifications possible
- Instance
  - Description
  - ContextData
- Activity
  - Name
  - Description

7.4. Deviations from the Wf-XML 2.0 Standard

The implementation is based on the ASAP specification of OASIS and Wf-XML 2.0. It turned out that their schema definitions are erroneous. This implementation is shipped with adapted and partially corrected schema definitions.

7.4.1. ServiceRegistry

The implementation of `NewDefinitionRq` differs from the specification. If the factory exists, the process definition will be overwritten without warning. It is also possible to create multiple factories using a single request. Neither the ASAP specification nor the Wf-XML specification consider the deletion of processes. Using this approach it is possible to delete processes within a package by simply overwriting the package definition.

The implementation of `GetProperties` produces XML that fails validation, since the Wf-XML schema definition is erroneous at this point. The ASAP schema defines groups for Instance, Factory, and Observer. Wf-XML defines new groups for the ServiceRegistry and the Activity resources, which are not referenced by `GetPropertiesRs`, since this is specified in the ASAP schema, and which does not know about any extensions. As a consequence the new groups for the Service Registry and the Activity resource must not be a part of a `GetPropertiesRs` message.

The specification mentions that a `ListDefinitions` may contain filters, i.e. `Name` and

. These filters are missing in the schema definition and are not implemented. However, filtering of factories might make sense, but more in terms of the package and process ids, rather than the name, which might be ambivalent.

7.4.2. Factory

`SetDefinition` is not implemented. As a workaround, `NewDefinition` of `ServiceRegistry` can be used instead, see above.

7.4.3. Instance

`TerminateRq` is not implemented, since the specification for this request in the ASAP schema is missing. The Wf-XML specification refers to the ASAP specification for this request, but it is not mentioned there. As a workaround it is possible to use a `ChangeStateRq` instead.

The `SubscribeRq` request allows to receive notification messages, if the process state changes. The ASAP specification is not precise at this point. The implemented behavior for these critical points is described below.

- Completion of a process includes also a state changes. Observers will receive `StateChangeRq` before a `CompletedRq` . ASAP provides no hints, if the latter one is intended to be a replacement for the first one.
- The observer will be unsubscribed if the process terminates. `UnsubscribeRq` requests are ignored afterwards.

7.4.4. Activity

The implementation of `GetProperties` produces XML that fails validation caused by an error in the Wf-XML schema as it is described for the Service Registry.

For `GetProperties` the field `DueDate` contains a timestamp that matches `Long.MAX_VALUE`, since this date is currently not available using the API.

7.5. Example Client

This section provides an example of a C# client that is developed using Microsoft Visual Studio 2005[1].

WfMOpen's Wf-XML interface ships with a WSDL description that can be used to generate code for a client. This WSDL is based on the WSDL that is available from OASIS and fixes some problems that we have encountered. Visual Studio supports code generation from the supplied WSDL file with the `wsdl` tool[2]. On the Visual Studio Command Prompt enter

```
wsdl /language:cs /namespace:WfMOpen.WfXML /out:WfMOpenService.cs <base-url>
```

Replace `<base-url>` with the URL of your WfMOpen distribution. This will create a file WfMOpenService.cs that has to added to your Visual Studio solution. The error messages have their cause in the erroneous ASAP and Wf-XML schema defintions and can be ignored. Do not forget to add a reference to `System.Web.Services`. The following code will print a list of all declared processes on the console:

[1]What would be the point in using Wf-XML to access WfMOpen when you have a Java client?
[2]Unfortunately, our WSDL file does not work with the `wsimport` tool of Java 6 because this tool enforces more XML schema restrictions. I.e. it behaves correctly, but the problems cannot be fixes without major changes in the files provided by OASIS and the WfMC (these files could really use some cleanup).

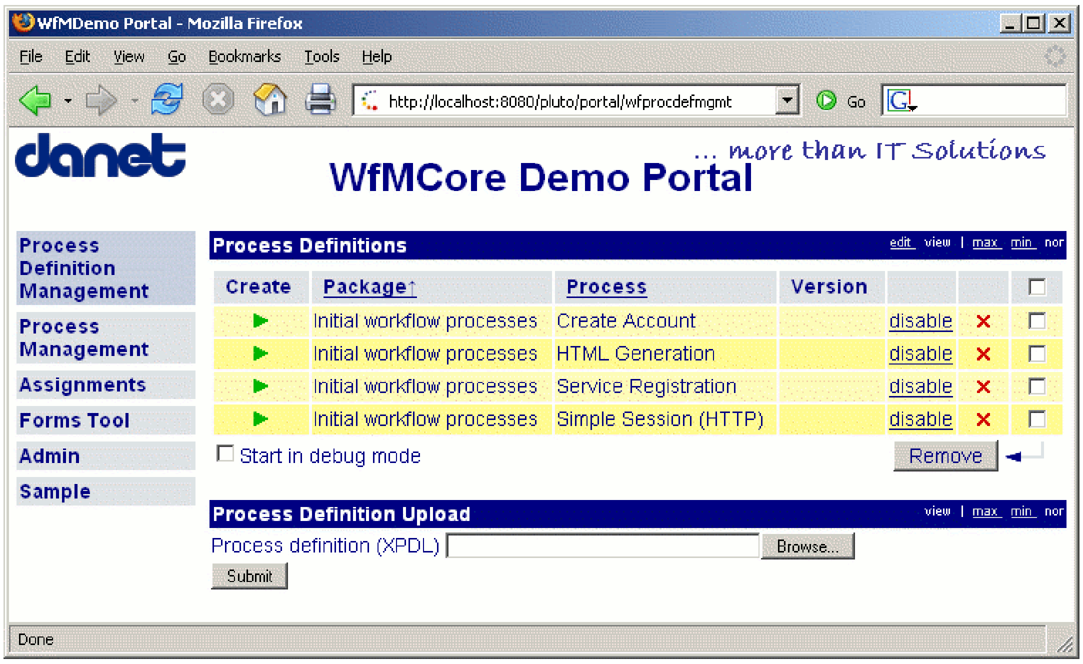
```
using System;
using System.Collections.Generic;
using System.Text;
using WfMOpen.WfXML;

namespace WfXMLTestClient
{
    class Program
    {
        static void Main(string[] args)
        {
            WfXmlServiceRegistry service = new WfXmlServiceRegistry();
            Request req = new Request();
            req.ReceiverKey = "<base-url>/wfxml/ServiceRegistry";
            service.RequestValue = req;

            DefinitionInfo[] definitions =
                service.ListDefintions("<ListDefinitionRq/>");

            for (int i = 0; i < definitions.Length; i++)
            {
                DefinitionInfo info = definitions[i];
                Console.WriteLine(info.DefinitionKey);
            }
        }
    }
}
```

Chapter 8. Management portlets

Besides the engine component, WfMOpen includes a GUI for common management tasks. The GUI is implemented as a portlet application, i.e. a set of JSR-168 compliant portlets. Portals and portlets are the state of the art (if not only) open component technology for Web based GUIs. Providing the management GUI as portlets allows you to easily integrate the portlets in your specific application GUI[1]. At the same time, a small footprint, stand-alone management GUI can be provided by using the minimal Pluto portal driver (see Apache Pluto site [http://portals.apache.org/pluto/v101/userguide/portal.html]).

8.1. Process definition portlet

The process definition portlet enables an administrator to view the available process definitions and start new processes.


WfMDemo Portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/pluto/portal/wfprocdefmgmt
Go
danet
... more than IT Solutions
WfMCore Demo Portal
Process Definition Management
Process Management
Assignments
Forms Tool
Admin
Sample
edit view | max min nor
Done

Process Definitions

Create	Package↑	Process	Version			☐
▶	Initial workflow processes	Create Account		disable	✕	☐
▶	Initial workflow processes	HTML Generation		disable	✕	☐
▶	Initial workflow processes	Service Registration		disable	✕	☐
▶	Initial workflow processes	Simple Session (HTTP)		disable	✕	☐

☐ Start in debug mode — Remove

Process Definition Upload

Process definition (XPDL) [ ] Browse...

Submit

Figure 8.1. Process definition page

To start a new process, simply click on the start icon (green triangle). If a process is to be started in debugging mode (see Section 2.6, "Debugging workflows" [19]) check the checkbox underneath the list of process definitions before starting the process. (Note that the screenshot shows both the process definition portlet and the process upload portlet described below.)

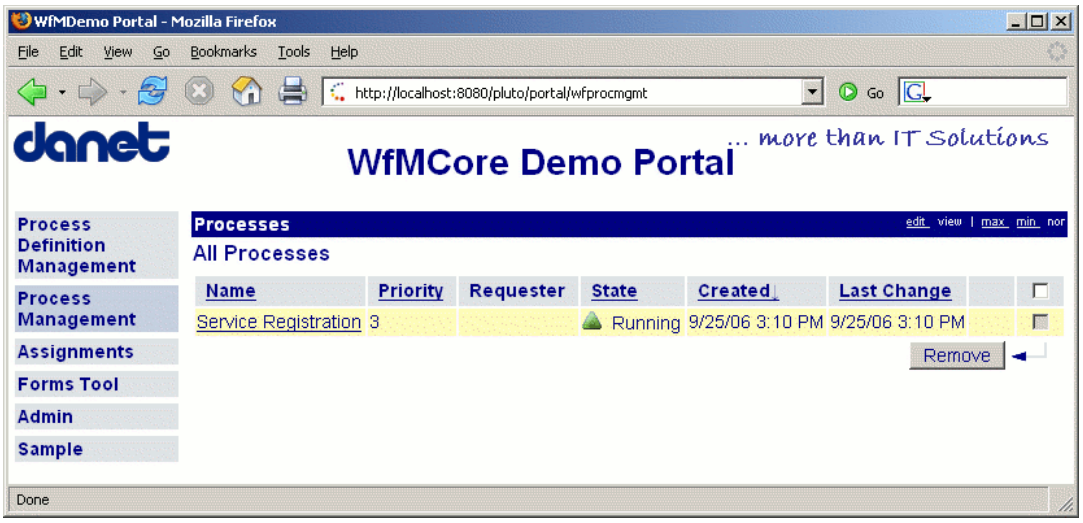
8.2. Process definition upload portlet

Also depicted in the figure above is the process definition upload. It allows the selection and upload of new process definitions.

8.3. Process portlet

The process portlet initially shows the list of processes managed by the workflow engine.

[1]Provided that it uses portal technology, of course.


WfMDemo Portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
danet
... more than IT Solutions
Process Definition Management
Process Management
Assignments
Forms Tool
Admin
Sample
edit view | max min nor
Remove
Done

http://localhost:8080/pluto/portal/wfprocmgmt

WfMCore Demo Portal

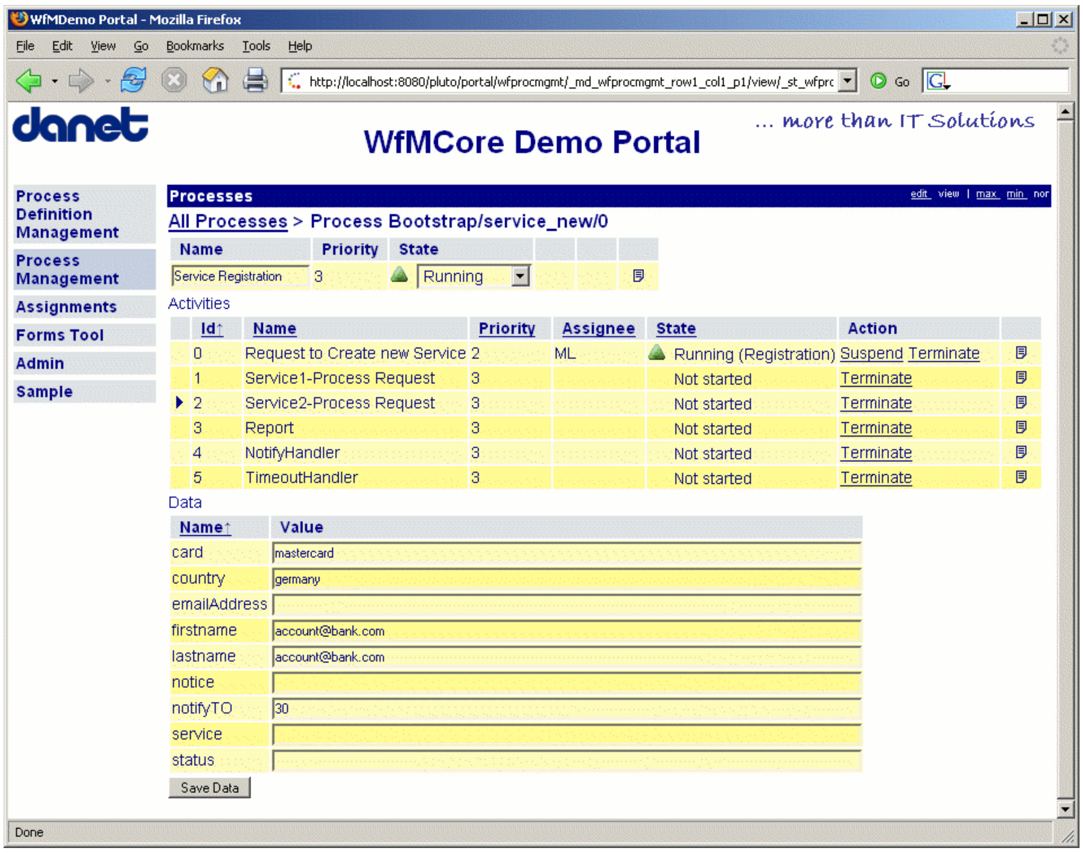
Processes

All Processes

Name	Priority	Requester	State	Created↓	Last Change	☐
Service Registration	3		Running	9/25/06 3:10 PM	9/25/06 3:10 PM	☐

Figure 8.2. Process list

Clicking on a process name navigates to a detail view of the process. Use the bread-crumb navigation at the top of the portlet to go back to the process list ("All Processes").


WfMDemo Portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
danet
... more than IT Solutions
Process Definition Management
Process Management
Assignments
Forms Tool
Admin
Sample
edit view | max min nor
Save Data
Done

http://localhost:8080/pluto/portal/wfprocmgmt/_md_wfprocmgmt_row1_col1_p1/view/_st_wfprc

WfMCore Demo Portal

Processes

All Processes > Process Bootstrap/service_new/0

Name	Priority	State
Service Registration	3	Running

Activities

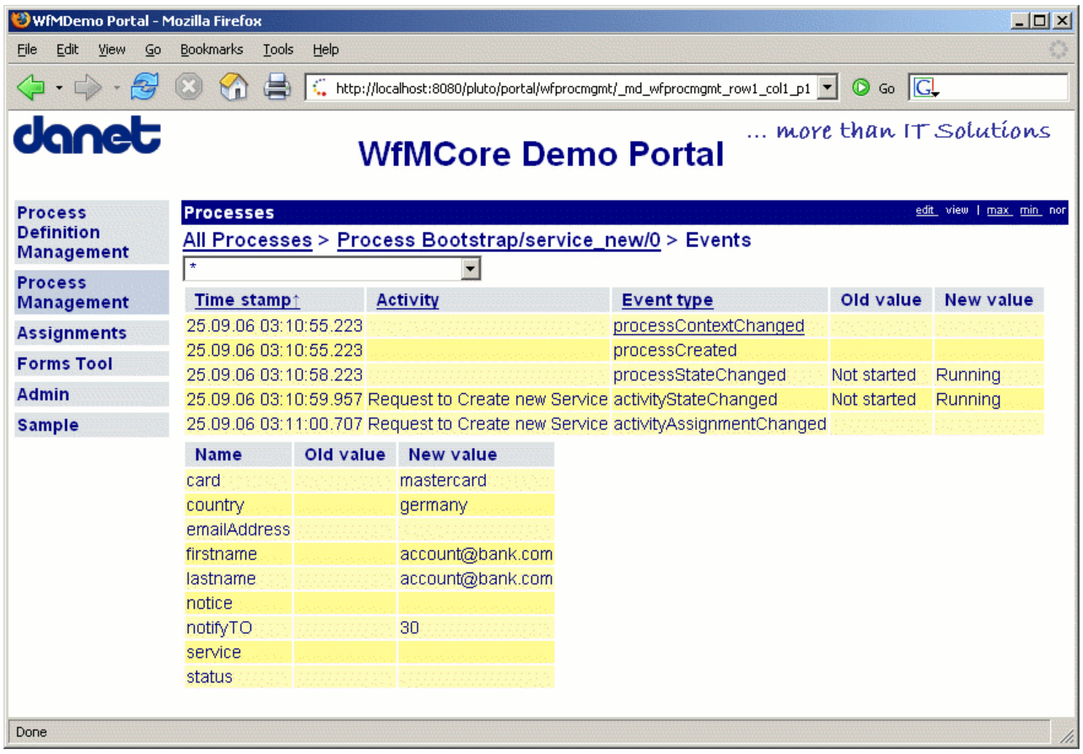
Id↑	Name	Priority	Assignee	State	Action
0	Request to Create new Service	2	ML	Running (Registration)	Suspend Terminate
1	Service1-Process Request	3		Not started	Terminate
2	Service2-Process Request	3		Not started	Terminate
3	Report	3		Not started	Terminate
4	NotifyHandler	3		Not started	Terminate
5	TimeoutHandler	3		Not started	Terminate

Data

Name↑	Value
card	mastercard
country	germany
emailAddress	
firstname	account@bank.com
lastname	account@bank.com
notice	
notifyTO	30
service	
status	

Figure 8.3. Process detail

Clicking on the small listing icon of either the process or an activity shows the events associated with the process or an activity.


WfMDemo Portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/pluto/portal/wfprocmgmt/_md_wfprocmgmt_row1_col1_p1
Go
danet
... more than IT Solutions
WfMCore Demo Portal
Process Definition Management
Process Management
Assignments
Forms Tool
Admin
Sample
Processes
edit view | max min nor
Done

All Processes > Process Bootstrap/service_new/0 > Events

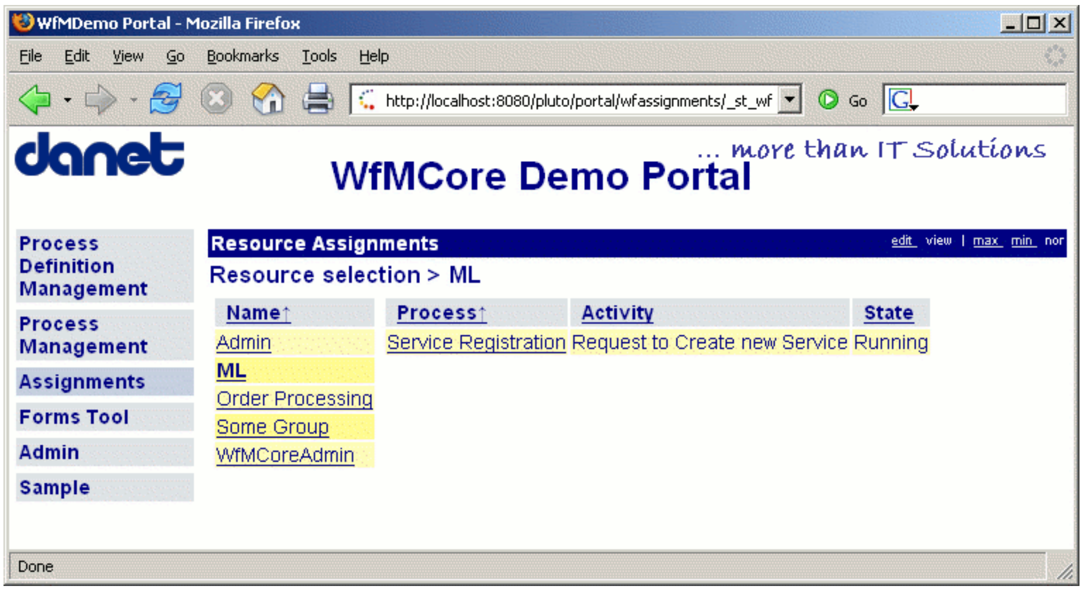
Time stamp↑	Activity	Event type	Old value	New value
25.09.06 03:10:55.223		processContextChanged		
25.09.06 03:10:55.223		processCreated		
25.09.06 03:10:58.223		processStateChanged	Not started	Running
25.09.06 03:10:59.957	Request to Create new Service	activityStateChanged	Not started	Running
25.09.06 03:11:00.707	Request to Create new Service	activityAssignmentChanged		

Name	Old value	New value
card		mastercard
country		germany
emailAddress		
firstname		account@bank.com
lastname		account@bank.com
notice		
notifyTO		30
service		
status		

Figure 8.4. Events display

For event type "processContextChanged", clicking on the event type displays the changes below the list of events.

A differently configured instance of the portlet can be displayed by selecting the "Assignments" item in the menu on the left. In this configuration, the portlet initially displays the list of known resources. Clicking on a resource name shows all assigned activities.


WfMDemo Portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/pluto/portal/wfassignments/_st_wf
Go
danet
... more than IT Solutions
WfMCore Demo Portal
Process Definition Management
Process Management
Assignments
Forms Tool
Admin
Sample
Resource Assignments
edit view | max min nor
Done

Resource selection > ML

Name↑
Admin
ML
Order Processing
Some Group
WfMCoreAdmin

Process↑	Activity	State
Service Registration	Request to Create new Service	Running

Figure 8.5. Assignments display

Clicking on a process name leads (again) to the detail view of the process. The only difference to the detail view reached from the process list is that the selected assignee is emphasized in the display.

8.4. Deploying the portlet application in a portal

8.4.1. General procedure

The portlet application is packaged as a self-contained WAR, i.e. it includes all libraries, resources etc. required to run the portlet in an JSR-168 compliant portal.

The only (obviously) missing libraries are the client libraries of the application server that runs the workflow engine. These libraries are required by the portlets to contact the workflow engine. We assume that these libraries are made available in the portal as a shared resource, not on a per-portlet application base. If the runtime environment of the portal is an application server (not just a servlet container), this is usually the case by default. If the portal runs e.g. in Tomcat, the application server's client libraries should be copied to `$CATALINA_HOME/shared/lib`.

Unless the portal and the workflow engine run in the same application server, the portlets need some information to lookup the JNDI service of the application server that runs the workflow engine. The preferred way to configure the JNDI server is adding environment entries to the portlet application's `web.xml` as described in newInstance() [251]. The propagation of the security identity in call to the workflow engine is left to the portlet container as described in the "Java Portlet Specification" (JSR-168), section PLT.20.5 "Propagation of Security Identity in EJB Calls".

8.4.2. Deploying in Jetspeed2

As distributed, the Jetspeed2 portal runs in the Tomcat servlet container environment (see Jetspeed2 Home Page [http://portals.apache.org/jetspeed-2]). It therefore requires both the installation of the application server's client libraries and a configuration update to ensure security identity propagation. Both issues are described in the following section assuming that the workflow engine is deployed in JBoss.

After installation, start the portal once and shut it down again.

Now copy `$JBOSS_HOME/client/jbossall-client.jar` to `$JETSPEED_HOME/shared/lib/`. This makes the JBoss client libraries availabe to all components run in this Tomcat environment.

In `$JETSPEED_HOME/webapps/jetspeed/WEB-INF/classes/` create a new file `login.conf` with the following contents:

```
Jetspeed {
   org.apache.jetspeed.security.impl.DefaultLoginModule required;
   org.jboss.security.ClientLoginModule required;
};
```

The line `org.jboss.security.ClientLoginModule required;` (added with respect to the default configuration[2]) causes the security credentials to be propagated to invoke EJBs.

[2]The default configuration file is "hidden" in one of the jars and overridden by the newly created file.

Chapter 9. Known bugs and limitations

9.1. Bugs

Currently none known.

9.2. Limitations

- XPDL `TypeDeclarations` are currently not supported. As a workaround, simply use `<SchemaType/>` as type specification. Note that the implementation of type declarations will only make the type information available at the API. There will be no verification of complex process relevant data set using the API against the type declarations.
- Currently only "text/javascipt", V1.5 and "text/ecmascript", 3rd Edition are supported as scripting languages.
- The XForms tool does currently not support the XForms Repeat Module as described in the XForms Recommendation.

Appendix A. The API documentation

A.1. Package de.danet.an.workflow.omgcore

This package defines the core domain of a workflow management system. It is an adaption of the OMG "Workflow Management Facility Sepcification, V1.2". We consider the interfaces and classes in this package (together with the interfaces and classes in the `extended API`) to be useable as a general Java workflow API, i.e. if there was a JSR for a Java workflow API, we think the merger of these two packages would be a good starting point.

We have tried to follow the original specification as close as possible. However, some conventions used by OMG do not fit in the Java environment and changes have been made accordingly.

The main differences are:

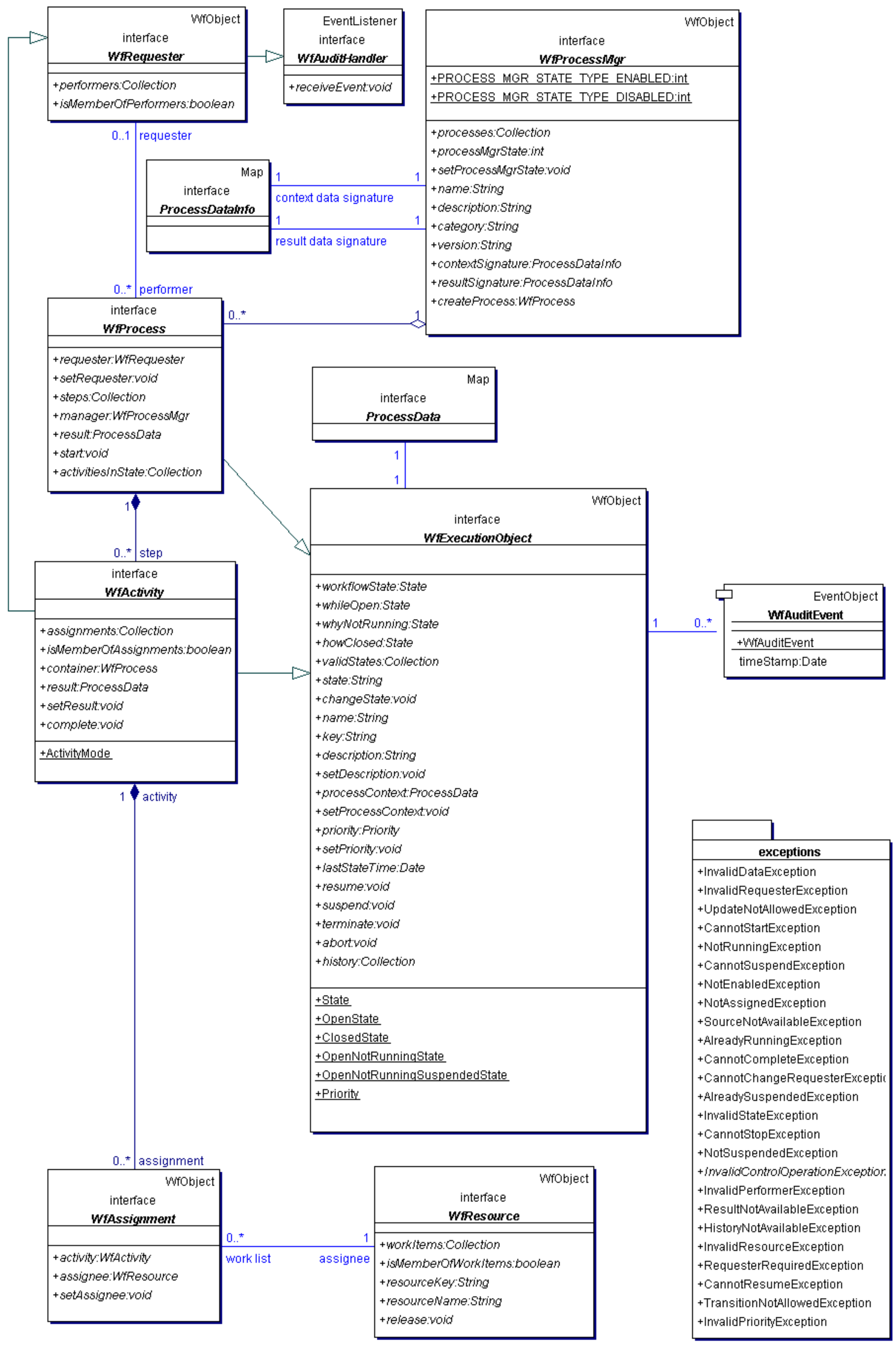
- OMG separates words in identifiers using underscores. In the Java environment, words are delimited by using a capital letter.
- The pattern for relationships has been replaced by a single access method `RELNAMEs` that returns a collection (the plural "s" has been omitted from the relation `history` of interface `WfExecutionObject` for gramatical reasons).
- The OMG model introduces a class `WfEventAudit` . While the collection of `WfEventAudit` s may be considered the result of an audit, we think that individual items are rather `WfAuditEvent` s. This conforms better to the Java naming scheme where event classes end with "...Event" and the corresponding listener interfaces with "...Listener".
- We have appended " `...Exception` " to the names of the exceptions, as is usual in Java. Additionally, we have put the exceptions in a separate sub-package to avoid naming conflicts when all `omgcore` classes are imported with a wildcard import.
- There are no enumeration types in Java. While, in general, elements of enumeration types can simply be mapped to constant definitions, we have not followed this approach in all cases.
  - In the Java environment, the states and substates of an execution object can conveniently be modelled as a `class hierarchy` . This provides type safe usage and allows to define some convenient methods for handling state.

We have, however **not** changed the following:

- The access method for attributes have not been renamed `getAttribute()` . The `getX()` / `setX(...)` pattern has been introduced by JavaBeans for the configuration of component properties, not as a general mechanism to access attributes. The properties used in the OMG model are dynamic data properties, not configuration options. Therefore, there is no reason why they should follow the `getX()` / `setX(...)` pattern

The documentation of classes and methods in this package consists mostly of an abbreviated version of the description provided by OMG's "Workflow Management Facility Specification, V1.2". Unless otherwise stated, the description from the specification applies fully, i.e. you can use the specification to obtain a detailed understanding of the functionallity provided.

The following picture shows the complete model.



A.1.1. Additional Information

Since	V1.0

A.1.2. Exception AlreadyRunningException

This exception is raised by an attempt to start a `WfProcess` that is already running.

A.1.2.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.AlreadyRunningException extends, 
    implements, java.io.Serializable {
// Public Constructors

  public AlreadyRunningException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.11, “Exception InvalidControlOperationException” [88] -> Section A.1.2, “Exception AlreadyRunningException” [83]

A.1.2.2. AlreadyRunningException(String)

```
  public AlreadyRunningException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `AlreadyRunningException` with the given message.

A.1.3. Exception AlreadySuspendedException

This exception is raised by an attempt to suspend a `WfExecutionObject` that is already suspended.

A.1.3.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.AlreadySuspendedException extends
    implements, java.io.Serializable {
// Public Constructors

  public AlreadySuspendedException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.11, “Exception InvalidControlOperationException” [88] -> Section A.1.3, “Exception AlreadySuspendedException” [83]

A.1.3.2. AlreadySuspendedException(String)

```
  public AlreadySuspendedException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `AlreadySuspendedException` with the given message.

A.1.4. Exception CannotChangeRequesterException

This exception is raised when a change of a `WfRequester` is requested, but cannot be fulfilled.

A.1.4.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.CannotChangeRequesterException extends,
    implements, java.io.Serializable {
// Public Constructors

  public CannotChangeRequesterException();

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.4, “Exception CannotChangeRequesterException” [84]

A.1.5. Exception CannotCompleteException

This exception is raised by an attempt to complete execution of a `WfExecutionObject` when it cannot be completed yet.

A.1.5.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.CannotCompleteException extends, de.dan
    implements, java.io.Serializable {
// Public Constructors

  public CannotCompleteException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.11, “Exception InvalidControlOperationException” [88] -> Section A.1.5, “Exception CannotCompleteException” [84]

A.1.5.2. CannotCompleteException(String)

```
  public CannotCompleteException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `CannotCompleteException` with the given message.

A.1.6. Exception CannotResumeException

This exception is raised by an operation on a `WfExecutionObject` that attempts to resume an object that is not in a proper condition.

A.1.6.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.CannotResumeException extends, de
    implements, java.io.Serializable {
// Public Constructors

  public CannotResumeException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.11, “Exception InvalidControlOperationException” [88] -> Section A.1.6, “Exception CannotResumeException” [85]

A.1.6.2. CannotResumeException(String)

```
  public CannotResumeException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `CannotResumeException` with the given message.

A.1.7. Exception CannotStartException

This exception is raised by an attempt to start a `WfProcess` that cannot be started yet.

A.1.7.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.CannotStartException extends, de.danet.
    implements, java.io.Serializable {
// Public Constructors

  public CannotStartException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.11, “Exception InvalidControlOperationException” [88] -> Section A.1.7, “Exception CannotStartException” [86]

A.1.7.2. CannotStartException(String)

```
  public CannotStartException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `CannotStartException` with the given message.

A.1.8. Exception CannotStopException

This exception is raised by an operation on a `WfExecutionObject` that attempts to stop an object that is not in a proper condition.

A.1.8.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.CannotStopException extends, de.danet.a
    implements, java.io.Serializable {
// Public Constructors

  public CannotStopException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.11, “Exception InvalidControlOperationException” [88] -> Section A.1.8, “Exception CannotStopException” [86]

A.1.8.2. CannotStopException(String)

```
  public CannotStopException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `CannotStopException` with the given message.

A.1.9. Exception CannotSuspendException

This exception is raised by an operation on a `WfExecutionObject` that attempts to suspend an object that is not in the proper condidition.

A.1.9.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.CannotSuspendException extends, d
    implements, java.io.Serializable {
// Public Constructors

  public CannotSuspendException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.11, “Exception InvalidControlOperationException” [88] -> Section A.1.9, “Exception CannotSuspendException” [87]

A.1.9.2. CannotSuspendException(String)

```
  public CannotSuspendException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `CannotSuspendException` with the given message.

A.1.10. Exception HistoryNotAvailableException

This exception is raised by a request for event audit history of a `WfExecutionObject` when the history is not available. For example because the implementation of the WFM facility does not support recording of history for a specific execution object.

A.1.10.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.HistoryNotAvailableException extends, j
    implements, java.io.Serializable {
// Public Constructors

  public HistoryNotAvailableException();

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.10, “Exception HistoryNotAvailableException” [87]

A.1.11. Exception InvalidControlOperationException

This exception is raised by an operation on a `WfExecutionObject` that attempts to perform an invalid control operation on that object.

A.1.11.1. Synopsis

```
 public abstract class de.danet.an.workflow.omgcore.InvalidControlOperationExcepti
    implements, java.io.Serializable {
// Public Constructors

  public InvalidControlOperationException(String msg);

}
```

Direct known subclasses : `de.danet.an.workflow.omgcore.AlreadyRunningException` , `de.danet.an.workflow.omgcore.AlreadySuspendedException` , `de.danet.an.workflow.omgcore.CannotCompleteException` , `de.danet.an.workflow.omgcore.CannotResumeException` , `de.danet.an.workflow.omgcore.CannotStartException` , `de.danet.an.workflow.omgcore.CannotStopException` , `de.danet.an.workflow.omgcore.CannotSuspendException` , `de.danet.an.workflow.omgcore.NotRunningException` , `de.danet.an.workflow.omgcore.NotSuspendedException`

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.11, “Exception InvalidControlOperationException” [88]

A.1.11.2. InvalidControlOperationException(String)

```
  public InvalidControlOperationException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `InvalidControlOperationException` with the given message.

A.1.12. Exception InvalidDataException

This exception is raised by an attempt to update the context of the result of a `WfExecutionObject` with data that does not match the signature of that object.

A.1.12.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.InvalidDataException extends, jav
    implements, java.io.Serializable {
// Public Constructors

  public InvalidDataException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.12, “Exception InvalidDataException” [89]

A.1.12.2. InvalidDataException(String)

```
  public InvalidDataException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `InvalidDataException` with the given message.

A.1.13. Exception InvalidPerformerException

This exception is raised by an attempt to signal a `WfAuditEvent` to a `WfRequester` that was not created by one of the `WfProcesses` associated with the `WfRequester` .

A.1.13.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.InvalidPerformerException extends
    implements, java.io.Serializable {
// Public Constructors

  public InvalidPerformerException();

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.13, “Exception InvalidPerformerException” [89]

A.1.14. Exception InvalidPriorityException

This exception is raised by an attempt to assign an invalid priority to a `WfExecutionObject` .

A.1.14.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.InvalidPriorityException extends, java.
    implements, java.io.Serializable {
// Public Constructors

  public InvalidPriorityException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.14, “Exception InvalidPriorityException” [90]

A.1.14.2. InvalidPriorityException(String)

```
  public InvalidPriorityException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `InvalidPriorityException` with the given message.

A.1.15. Exception InvalidRequesterException

This exception is raised when a `WfRequester` is being identified that cannot be a "parent" of instances of the process modell. When a `WfRequester` is rejected, the invoking application might decide not to register a `WfRequester` with the `WfProcess` .

A.1.15.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.InvalidRequesterException extends, java
    implements, java.io.Serializable {
// Public Constructors

  public InvalidRequesterException();
}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.15, “Exception InvalidRequesterException” [90]

A.1.16. Exception InvalidResourceException

This exception is raised by an attempt to assign or remove an invalid resource.

A.1.16.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.InvalidResourceException extends,
    implements, java.io.Serializable {
// Public Constructors

  public InvalidResourceException();

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.16, “Exception InvalidResourceException” [91]

A.1.17. Exception InvalidStateException

This exception is raised by an attempt to change the state of a `WfExecutionObject` to a state that is not defined for that object.

A.1.17.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.InvalidStateException extends, ja
    implements, java.io.Serializable {
// Public Constructors

  public InvalidStateException();


  public InvalidStateException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.17, “Exception InvalidStateException” [91]

A.1.17.2. InvalidStateException()

```
  public InvalidStateException();
```

Creates a new `InvalidStateException` .

A.1.17.3. InvalidStateException(String)

```
  public InvalidStateException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `InvalidStateException` with the given message.

A.1.18. Exception NotAssignedException

This exception is raised by an attempt to release a `WfResource` from an assignment it is not associated with.

A.1.18.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.NotAssignedException extends, java.lang
    implements, java.io.Serializable {
// Public Constructors

  public NotAssignedException();

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.18, “Exception NotAssignedException” [92]

A.1.19. Exception NotEnabledException

This exception is raised by an attempt to create a `WfProcess` using a `WfProcessMgr` that is disabled.

A.1.19.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.NotEnabledException extends, java.lang.
    implements, java.io.Serializable {
// Public Constructors

  public NotEnabledException(String msg);
}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.19, “Exception NotEnabledException” [92]

A.1.19.2. NotEnabledException(String)

```
  public NotEnabledException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new exception with the given message.

A.1.20. Exception NotRunningException

This exception is raised by an operation on a `WfExecutionObject` that attempts to perform a control operation on an object that needs to be in a running state, but is not.

A.1.20.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.NotRunningException extends, de.d
    implements, java.io.Serializable {
// Public Constructors

  public NotRunningException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.11, “Exception InvalidControlOperationException” [88] -> Section A.1.20, “Exception NotRunningException” [93]

A.1.20.2. NotRunningException(String)

```
  public NotRunningException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `NotRunningException` with the given message.

A.1.21. Exception NotSuspendedException

This exception is raised by an operation on a `WfExecutionObject` that attempts to perform a control operation on an object that needs to be in a suspended state, but is not.

A.1.21.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.NotSuspendedException extends, de.danet
    implements, java.io.Serializable {
// Public Constructors

  public NotSuspendedException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.11, “Exception InvalidControlOperationException” [88] -> Section A.1.21, “Exception NotSuspendedException” [94]

A.1.21.2. NotSuspendedException(String)

```
  public NotSuspendedException(String msg);
```

Parameters

`msg`	Description of the cause.

Creates a new `NotSuspendedException` with the given message.

A.1.22. Interface ProcessData

`ProcessData` represents context and result data of a `WfExecutionObject` . We assume that the names in a `NameValueSequence` as defined in the OMG sprecification have to be unique (it is a reasonable assumption, although the OMG specification does not state it explicitly). Therefore we simply map the `NameValueSequence` to a Map with keys of type `String` and values of type `Object` .

A.1.22.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.ProcessData extends, java.util.Map
}
```

Inheritance Path. Section A.1.22, “Interface ProcessData” [94]

A.1.23. Interface ProcessDataInfo

ProcessDataInfo describes the structure of the process data associated with a WfExecutionObject . The OMG specification defines that the data type is specified by its "IDL type represented by its string name". Porting this to Java is a bit difficult as the equivalent to this would be to support something like textual representations of Java types, i.e. Java source code.

While this issue needs further investigation, we define the following: Java primitive types are represented by the corresponding classes (e.g. boolean values are represented by a map entry with value java.lang.Boolean.class). Support of complex types depends on the workflow engine used (some may support any [serializable] Java object as process relevant data, some may support only XML as serialized format), as does the way process definitions define complex types (e.g. as XML Schema Description).

In addition to the classes representing Java primitive forms, an implementation may therefore specify additional types to describe the type information of a process data item.

A.1.23.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.ProcessDataInfo extends, java
}
```

Inheritance Path. Section A.1.23, “Interface ProcessDataInfo” [94]

A.1.24. Exception RequesterRequiredException

This exception is raised when a valid WfRequester is required by the process definition, but one is not supplied.

A.1.24.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.RequesterRequiredException extend
    implements, java.io.Serializable {
// Public Constructors

  public RequesterRequiredException();

}
```

Methods inherited from java.lang.Throwable : fillInStackTrace , getCause , getLocalizedMessage , getMessage , getStackTrace , initCause , printStackTrace , setStackTrace , toString

Methods inherited from java.lang.Object : clone , equals , finalize , getClass , hashCode , notify , notifyAll , wait

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.24, “Exception RequesterRequiredException” [95]

A.1.25. Exception ResultNotAvailableException

This exception is raised when the requested result of a WfExecutionObject is not available (yet).

A.1.25.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.ResultNotAvailableException exten
    implements, java.io.Serializable {
// Public Constructors

  public ResultNotAvailableException();
}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.25, “Exception ResultNotAvailableException” [95]

A.1.26. Exception SourceNotAvailableException

This exception is raised by the request for the source of a `WfAuditEvent` when the source is no longer available.

A.1.26.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.SourceNotAvailableException extends, ja
    implements, java.io.Serializable {
// Public Constructors

  public SourceNotAvailableException();

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.26, “Exception SourceNotAvailableException” [96]

A.1.27. Exception TransitionNotAllowedException

This exception is raised by an attempt to perform an invalid state transistion of a `WfExecutionObject` .

A.1.27.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.TransitionNotAllowedException extends, 
    implements, java.io.Serializable {
// Public Constructors

  public TransitionNotAllowedException(WfExecutionObject.State from,
                                       WfExecutionObject.State to);


  public TransitionNotAllowedException(WfExecutionObject.State from,
                                       WfExecutionObject.State to,
                                       String msg);


  public TransitionNotAllowedException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.27, “Exception TransitionNotAllowedException” [96]

A.1.27.2. TransitionNotAllowedException(String)

```
public TransitionNotAllowedException(String msg);
```

Parameters

`msg`	the given message.

Creates a new `TransitionNotAllowedException` with the given message.

A.1.27.3. TransitionNotAllowedException(WfExecutionObject.State, WfExecutionObject.State)

```
public TransitionNotAllowedException(WfExecutionObject.State from,
                                     WfExecutionObject.State to);
```

Parameters

`from`	current state
`to`	new state

Creates a new `TransitionNotAllowedException` .

A.1.27.4. TransitionNotAllowedException(WfExecutionObject.State, WfExecutionObject.State, String)

```
public TransitionNotAllowedException(WfExecutionObject.State from,
                                     WfExecutionObject.State to,
                                     String msg);
```

Parameters

`from`	current state
`to`	new state
`msg`	description of the cause.

Creates a new `TransitionNotAllowedException` with the given message.

A.1.28. Exception UpdateNotAllowedException

This exception is raised when it is not allowed to update the process context.

A.1.28.1. Synopsis

```
 public class de.danet.an.workflow.omgcore.UpdateNotAllowedException extends, java
    implements, java.io.Serializable {
// Public Constructors

  public UpdateNotAllowedException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.1.28, “Exception UpdateNotAllowedException” [98]

A.1.28.2. UpdateNotAllowedException(String)

```
  public UpdateNotAllowedException(String msg);
```

Parameters

`msg`	description of the cause.

Creates a new Exception with the given message.

A.1.29. Interface WfActivity

`WfActivity` is a step in a process that is associated, as part of an aggregation, with a single `WfProcess` . It represents a request for work in the context of the containing `WfProcess` .

A.1.29.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfActivity extends, de.danet.an.wor
// Public Methods

  public java.util.Collection assignments()
    throws RemoteException;


  public void complete()
    throws RemoteException, CannotCompleteException;


  public WfProcess container()
    throws RemoteException;


  public boolean isMemberOfAssignments(WfAssignment member)
    throws RemoteException;
  public ProcessData result()
    throws RemoteException, ResultNotAvailableException;


  public void setResult(ProcessData result)
    throws RemoteException, InvalidDataException;

}
```

Inheritance Path. Section A.1.29, “Interface WfActivity” [98]

A.1.29.2. assignments()

```
  public java.util.Collection assignments()
    throws RemoteException;
```

Parameters

return	the collection of all the WfAssignment.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns all the `WfAssignment` associated with a `WfActivity` .

A.1.29.3. complete()

```
  public void complete()
    throws RemoteException, CannotCompleteException;
```

Exceptions

`RemoteException`	if a system-level error occurs.
`CannotCompleteException`	if the activity cannot be completed yet.

This method is used by an application to signal the completion of an activity. Note that this does not necessarily imply that the activity's state changes to `closed.completed` .

XPDL allows an activity to be implemented by several tools that are executed in sequence. Only if `complete` is called after the last tool has been started will the workflow engine change the activity's state to `closed.completed` . Else the activity will remain in the `open.running` state and the next tool will be started. If `complete` is called while the activity's state has been set to `open.not_running.suspended` the next tool will not be started until the activity is resumed.

The `extended API` provides methods for finding out which tool is currently being executed.

A.1.29.4. container()

```
  public WfProcess container()
    throws RemoteException;
```

Parameters

return	the process.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the `WfProcess` that this activity is a part of.

A.1.29.5. isMemberOfAssignments(WfAssignment)

```
public boolean isMemberOfAssignments(WfAssignment member)
  throws RemoteException;
```

Parameters

`member`	the assignment in question.
return	true if the assignment is among the assignments of this activity.

Exceptions

`RemoteException`	if a system-level error occurs.

Check if the given assignment is among the assignments of this activity.

A.1.29.6. result()

```
public ProcessData result()
  throws RemoteException, ResultNotAvailableException;
```

Parameters

return	the process data as result

Exceptions

`RemoteException`	if a system-level error occurs.
`ResultNotAvailableException`	if accessing to the result of an activity is not supported or the result data are not available yet.

Returns the result produced by the realization of the work request represented by an activity.

A.1.29.7. setResult(ProcessData)

```
  public void setResult(ProcessData result)
    throws RemoteException, InvalidDataException;
```

Parameters

`result`	the result data.

Exceptions

`RemoteException`	if a system-level error occurs.
`InvalidDataException`	if the data do not match the signature of the activity or when an invalid attempt is made to update the results of an activity; lack of access rights might be one of those reasons.

Passes result data back to the workflow process. The data item names must be the names of formal IN or OUT parameters of the currently invoked tool or subflow.

A.1.30. Interface WfAssignment

A `WfAssignment` links `WfActivity` objects to `WfResource` objects. These links represent real assignements for enacting the activity.

A.1.30.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfAssignment extends, de.dane
// Public Methods

  public WfActivity activity()
    throws RemoteException;


  public WfResource assignee()
    throws RemoteException;


  public void setAssignee(WfResource newValue)
    throws RemoteException, InvalidResourceException;

}
```

Inheritance Path. Section A.1.30, “Interface WfAssignment” [101]

A.1.30.2. activity()

```
  public WfActivity activity()
    throws RemoteException;
```

Parameters

return	the associated activity.

Exceptions

`RemoteException`	if a system-level error occurs.

A `WfAssignment` is associated with one `WfActivity` ; the association is established when the assignment is created as part of the resource selection process for the activity. This method returns the associated activity.

A.1.30.3. assignee()

```
public WfResource assignee()
  throws RemoteException;
```

Parameters

return	the associated resource.

Exceptions

`RemoteException`	if a system-level error occurs.

A `WfAssignment` is associated with one `WfResource` ; the association is established when the assignment is created as part of the resource selection process for the activity. This method returns the associated resource.

A.1.30.4. setAssignee(WfResource)

```
public void setAssignee(WfResource newValue)
  throws RemoteException, InvalidResourceException;
```

Parameters

`newValue`	the new resource.

Exceptions

`InvalidResourceException`	if an attempt is made to assign an invalid resource to the assignment.
`RemoteException`	if a system-level error occurs.

A `WfAssignment` is associated with one `WfResource` ; the association is established when the assignment is created as part of the resource selection process for the activity; this method can be used to reassign the assignment to another resource at a later point in time.

A.1.31. Interface WfAssignmentAuditEvent

A `WfAssignmentAuditEvent` provides an audit record of assignment change information for either the status of an assignment change for a `WfActivity` or when an exisiting assignment is re-assigned to another resource.

A.1.31.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfAssignmentAuditEvent extend
// Public Methods

  public String newResourceKey();


  public String newResourceName();


  public String oldResourceKey();


  public String oldResourceName();

}
```

Inheritance Path. Section A.1.31, “Interface WfAssignmentAuditEvent” [103]

A.1.31.2. newResourceKey()

```
  public String newResourceKey();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `newResourceKey` .

A.1.31.3. newResourceName()

```
  public String newResourceName();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `newResourceName` .

A.1.31.4. oldResourceKey()

```
  public String oldResourceKey();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `oldResourceKey` .

A.1.31.5. oldResourceName()

```
  public String oldResourceName();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `oldResourceName` .

A.1.32. Interface WfAuditEvent

A `WfAuditEvent` provides an audit record of workflow event information.

It provides information on the source of the event and contains specific event data. Workflow events include state changes, change of a resource assignment, and data changes. Workflow events are persistent and can be accessed navigating the history relationship of a `WfExecutionObject` .

Workflow audit event objects are not part of the persistent state of their source workflow object.

A.1.32.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfAuditEvent {
// Public Static Fields

  public static final String ACTIVITY_ASSIGNMENT_CHANGED  = activityAssignmentChan

  public static final String ACTIVITY_CONTEXT_CHANGED  = activityContextChanged;

  public static final String ACTIVITY_RESULT_CHANGED  = activityResultChanged;

  public static final String ACTIVITY_STATE_CHANGED  = activityStateChanged;

  public static final String PROCESS_CONTEXT_CHANGED  = processContextChanged;

  public static final String PROCESS_CREATED  = processCreated;

  public static final String PROCESS_STATE_CHANGED  = processStateChanged;

// Public Methods

  public String activityKey();

  public String activityName();

  public String eventType();
  public String processKey();

  public String processMgrName();

  public String processMgrVersion();

  public String processName();

  public WfExecutionObject source()
    throws SourceNotAvailableException;

  public java.util.Date timeStamp();

}
```

Inheritance Path. Section A.1.32, “Interface WfAuditEvent” [104]

A.1.32.2. ACTIVITY_ASSIGNMENT_CHANGED

```
  public static final String ACTIVITY_ASSIGNMENT_CHANGED  = activityAssignme
```

An identifier for the event type "activityAssignmentChanged"

A.1.32.3. ACTIVITY_CONTEXT_CHANGED

```
  public static final String ACTIVITY_CONTEXT_CHANGED  = activityContextChan
```

An identifier for the event type "activityContextChanged"

A.1.32.4. ACTIVITY_RESULT_CHANGED

```
  public static final String ACTIVITY_RESULT_CHANGED  = activityResultChange
```

An identifier for the event type "activityResultChanged"

A.1.32.5. ACTIVITY_STATE_CHANGED

```
  public static final String ACTIVITY_STATE_CHANGED  = activityStateChanged;
```

An identifier for the event type "activityStateChanged"

A.1.32.6. PROCESS_CONTEXT_CHANGED

```
  public static final String PROCESS_CONTEXT_CHANGED  = processContextChange
```

An identifier for the event type "processContextChanged"

A.1.32.7. PROCESS_CREATED

```
  public static final String PROCESS_CREATED  = processCreated;
```

An identifier for the event type "processCreated"

A.1.32.8. PROCESS_STATE_CHANGED

```
public static final String PROCESS_STATE_CHANGED  = processStateChanged;
```

An identifier for the event type "processStateChanged"

A.1.32.9. activityKey()

```
public String activityKey();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `activityKey` .

A.1.32.10. activityName()

```
public String activityName();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `activityName` .

A.1.32.11. eventType()

```
public String eventType();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `eventType` .

A.1.32.12. processKey()

```
public String processKey();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `processKey` .

A.1.32.13. processMgrName()

```
public String processMgrName();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `processMgrName` .

A.1.32.14. processMgrVersion()

```
public String processMgrVersion();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `processMgrVersion` .

A.1.32.15. processName()

```
public String processName();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `processName` .

A.1.32.16. source()

```
public WfExecutionObject source()
  throws SourceNotAvailableException;
```

Parameters

return	the current value of the attribute.

Exceptions

`SourceNotAvailableException`	if the source is not available.

Returns the current value of the attribute `source` . The source of the event is the `WfExceutionObject` associated to the event, i.e. that triggered the event.

A.1.32.17. timeStamp()

```
public java.util.Date timeStamp();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `timeStamp` .

A.1.33. Interface WfAuditHandler

The listener interface for receiving an event from a process.

A.1.33.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfAuditHandler extends, java.util.E
// Public Methods

  public void receiveEvent(WfAuditEvent e)
    throws InvalidPerformerException, RemoteException;

}
```

Inheritance Path. Section A.1.33, “Interface WfAuditHandler” [108]

A.1.33.2. receiveEvent(WfAuditEvent)

```
public void receiveEvent(WfAuditEvent e)
  throws InvalidPerformerException, RemoteException;
```

Parameters

`e`	the event.

Exceptions

`InvalidPerformerException`	thrown by the derived `WfRequester` if it receives an event from a process that is not among its performers.
`RemoteException`	if a system-level error occurs.

Called by the workflow engine if an event occurs.

A.1.34. Interface WfCreateProcessAuditEvent

A `WfCreateProcessAuditEvent` provides an audit record with information related to the creation of a process. If the process is created as a sub-process of another process that is synchronized with the main process via a `WfActivity` requester, information on the requester is recorded.

A.1.34.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfCreateProcessAuditEvent exte
// Public Methods

  public String pActivityKey();


  public String pProcessKey();


  public String pProcessMgrName();


  public String pProcessMgrVersion();


  public String pProcessName();

}
```

Inheritance Path. Section A.1.34, “Interface WfCreateProcessAuditEvent” [108]

A.1.34.2. pActivityKey()

```
  public String pActivityKey();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `pActivityKey` .

A.1.34.3. pProcessKey()

```
  public String pProcessKey();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `pProcessKey` .

A.1.34.4. pProcessMgrName()

```
  public String pProcessMgrName();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `pProcessMgrName` .

A.1.34.5. pProcessMgrVersion()

```
  public String pProcessMgrVersion();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `pProcessMgrVersion` .

A.1.34.6. pProcessName()

```
  public String pProcessName();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `pProcessName` .

A.1.35. Interface WfDataAuditEvent

A `WfDataAuditEvent` provides an audit record of either context changes of a `WfExecu-tionObject` or result changes of a `WfActivity` .

A.1.35.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfDataAuditEvent extends, de.danet.
// Public Methods

  public ProcessData newData();


  public ProcessData oldData();

}
```

Inheritance Path. Section A.1.35, “Interface WfDataAuditEvent” [110]

A.1.35.2. newData()

```
  public ProcessData newData();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `newData` .

A.1.35.3. oldData()

```
  public ProcessData oldData();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `oldData` .

A.1.36. Interface WfExecutionObject

`WfExecutionObject` is an abstract base interface that defines common attributes, states, and operations for `WfProcess` and `WfActivity` .

A.1.36.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfExecutionObject extends, de
// Public Methods

  public void abort()
    throws RemoteException, CannotStopException, NotRunningException;


  public void changeState(String newState)
    throws RemoteException, InvalidStateException, TransitionNotAllowedExcep


  public String description()
    throws RemoteException;


  public java.util.Collection history()
    throws RemoteException, HistoryNotAvailableException;


  public WfExecutionObject.State howClosed()
    throws RemoteException;


  public String key()
    throws RemoteException;


  public java.util.Date lastStateTime()
    throws RemoteException;


  public String name()
    throws RemoteException;


  public int priority()
    throws RemoteException;


  public ProcessData processContext()
    throws RemoteException;


  public void resume()
    throws RemoteException, CannotResumeException, NotRunningException, NotS
  public void setDescription(String newValue)
    throws RemoteException;


  public void setName(String newValue)
    throws RemoteException;


  public void setPriority(int newValue)
    throws RemoteException, InvalidPriorityException, UpdateNotAllowedException;


  public void setProcessContext(ProcessData newValue)
    throws RemoteException, InvalidDataException, UpdateNotAllowedException;


  public String state()
    throws RemoteException;


  public void suspend()
    throws RemoteException, CannotSuspendException, NotRunningException, AlreadySu


  public void terminate()
    throws RemoteException, CannotStopException, NotRunningException;


  public java.util.Collection validStates()
    throws RemoteException;


  public WfExecutionObject.State whileOpen()
    throws RemoteException;


  public WfExecutionObject.State whyNotRunning()
    throws RemoteException;


  public WfExecutionObject.State workflowState()
    throws RemoteException;

}
```

Inheritance Path. Section A.1.36, “Interface WfExecutionObject” [111]

A.1.36.2. abort()

```
  public void abort()
    throws RemoteException, CannotStopException, NotRunningException;
```

Exceptions

`RemoteException`	if a system-level error occurs.
`CannotStopException`	when the execution object cannot be aborted.
`NotRunningException`	when the object is not running.

Requests enactment of a suspended execution object to be aborted before its normal completion. The state is set to ClosedState.ABORTED.

A.1.36.3. changeState(String)

```
public void changeState(String newState)
  throws RemoteException, InvalidStateException, TransitionNotAllowedExcep
```

Parameters

`newState`	State to change to.

Exceptions

`InvalidStateException`	If `newState` is an invalid state for the execution object.
`TransitionNotAllowedEx-ception`	If the transition from the current state to `newState` is not al-lowed.
`RemoteException`	If a communication error occurred.

Updates the current state of the execution object. As a result the state of execution objects associated with this execution object might be updated, too.

A.1.36.4. description()

```
public String description()
  throws RemoteException;
```

Parameters

return	a string value of the description.

Exceptions

`RemoteException`	If a communication error occurred.

Get the description of this `WfExecutionObject` .

A.1.36.5. history()

```
public java.util.Collection history()
  throws RemoteException, HistoryNotAvailableException;
```

Parameters

return	the collection of all `WfAuditEvent` items.

Exceptions

`RemoteException`	if a system-level error occurs.
`HistoryNotAvailableException`	if any audit event item available.

Return all `WfAuditEvent` items associated with this execution object.

A.1.36.6. howClosed()

```
public WfExecutionObject.State howClosed()
  throws RemoteException;
```

Parameters

return	the state as `State` object.

Exceptions

`RemoteException`	If a communication error occurred.

Returns the workflow state for closed execution objects.

A.1.36.7. key()

```
public String key()
  throws RemoteException;
```

Parameters

return	string of the identifier.

Exceptions

`RemoteException`	If a communication error occurred.

Identifier of the execution object.

A.1.36.8. lastStateTime()

```
public java.util.Date lastStateTime()
  throws RemoteException;
```

Parameters

return	value of the time.

Exceptions

`RemoteException`	if a system-level error occurs.

Return the time the state of the WfExecutionObject was changed.

A.1.36.9. name()

```
public String name()
  throws RemoteException;
```

Parameters

return	string of the descriptive identifier.

Exceptions

`RemoteException`	If a communication error occurred.

Return human readable, descriptive identifier of the execution object.

A.1.36.10. priority()

```
public int priority()
  throws RemoteException;
```

Parameters

return	the value of the priority.

Exceptions

`RemoteException`	if a system-level error occurs.

Return the priority of this `WfExecutionObject` .

A.1.36.11. processContext()

```
public ProcessData processContext()
  throws RemoteException;
```

Parameters

return	the process relevant data that define the context of the execu-

tion object.

Exceptions

`RemoteException`	If a communication error occurred.

Return the context of this `WfExecutionObject` .

A.1.36.12. resume()

```
public void resume()
  throws RemoteException, CannotResumeException, NotRunningException, NotSuspend
```

Exceptions

`RemoteException`	if a system-level error occurs.
`CannotResumeException`	when the execution object cannot be resumed. For example, resuming a WfActivity might not be allowed when the containing WfProcess is suspended.
`NotRunningException`	when the object is not running.
`NotSuspendedException`	when the object is not suspended.

Requests enactment of a suspended execution object to be resumed. The state is set to OpenState.Running (or a substate) from NotRunningState.SUSPENDED.

A.1.36.13. setDescription(String)

```
public void setDescription(String newValue)
  throws RemoteException;
```

Parameters

`newValue`	the description of this `WfExecutionObject` .

Exceptions

`RemoteException`	If a communication error occurred.

Set the description of this `WfExecutionObject` .

A.1.36.14. setName(String)

```
public void setName(String newValue)
  throws RemoteException;
```

Parameters

`newValue`	the description of this `WfExecutionObject` .

Exceptions

`RemoteException`	If a communication error occurred.

Set the name of this `WfExecutionObject` .

A.1.36.15. setPriority(int)

```
public void setPriority(int newValue)
  throws RemoteException, InvalidPriorityException, UpdateNotAllowedExcept
```

Parameters

`newValue`	new priority to set

Exceptions

`RemoteException`	if a system-level error occurs.
`InvalidPriorityException`	when the specified priority is out of range.
`UpdateNotAllowedException`	when the priority cannot be updated.

Update the priority of this `WfExecutionObject` .

A.1.36.16. setProcessContext(ProcessData)

```
public void setProcessContext(ProcessData newValue)
  throws RemoteException, InvalidDataException, UpdateNotAllowedException;
```

Parameters

`newValue`	process relevant data that define the context of the execution object.

Exceptions

`RemoteException`	if a system-level error occurs.
`InvalidDataException`	when new process data does not match the signature of this `WfExecutionObject` .

`UpdateNotAllowedException`	raised when the implementation of the WfM Facility or the specific workflow process does not allow an update of the context.

Set the context of this `WfExecutionObject` .

A.1.36.17. state()

```
public String state()
  throws RemoteException;
```

Parameters

return	the current state.

Exceptions

`RemoteException`	If a communication error occurred.

Gets the current state of the object.

A.1.36.18. suspend()

```
public void suspend()
  throws RemoteException, CannotSuspendException, NotRunningException, AlreadySu
```

Exceptions

`RemoteException`	if a system-level error occurs.
`CannotSuspendException`	when the execution object cannot be suspended. For example, an implementation of the WfM Facility might not support suspension of a `WfActivity` .
`NotRunningException`	when the object is not running.
`AlreadySuspendedException`	when the object is already suspended.

Requests enactment of an execution object to be suspended. The state is set to NotRunningState.SUSPENDED (or one of its substates).

A.1.36.19. terminate()

```
public void terminate()
  throws RemoteException, CannotStopException, NotRunningException;
```

Exceptions

`RemoteException`	if a system-level error occurs.

`CannotStopException`	when the execution object cannot be aborted.
`NotRunningException`	when the object is not running.

Requests enactment of an execution object to be terminated before its normal completion.

A.1.36.20. validStates()

```
public java.util.Collection validStates()
  throws RemoteException;
```

Parameters

return	A collection of all the valid states.

Exceptions

`RemoteException`	If a communication error occurred.

Returns a list of all the valid states that can be reached from the current state.

A.1.36.21. whileOpen()

```
public WfExecutionObject.State whileOpen()
  throws RemoteException;
```

Parameters

return	the state as `State` object.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the workflow state for open execution objects.

A.1.36.22. whyNotRunning()

```
public WfExecutionObject.State whyNotRunning()
  throws RemoteException;
```

Parameters

return	the state as `State` object.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the workflow state for open, not running execution objects.

A.1.36.23. workflowState()

```
public WfExecutionObject.State workflowState()
  throws RemoteException;
```

Parameters

return	the current `state` .

Exceptions

`RemoteException`	if a system-level error occurs.

Return the current state.

A.1.37. Class WfExecutionObject.ClosedState

This class defines the sub-states of State.CLOSED of a `WfExecutionObject` as returned by `howClosedState()` .

A.1.37.1. Synopsis

```
 public static class de.danet.an.workflow.omgcore.WfExecutionObject.ClosedState ex
   implements, java.io.Serializable {
// Public Static Fields

  public static final WfExecutionObject.ClosedState ABORTED ;


  public static final WfExecutionObject.ClosedState COMPLETED ;


  public static final WfExecutionObject.ClosedState TERMINATED ;

// Protected Constructors

  protected WfExecutionObject.ClosedState(String text);

// Public Methods

  public WfExecutionObject.State getParent();


  public WfExecutionObject.State howClosedState();


  public WfExecutionObject.State whileOpenState();
  public WfExecutionObject.State whyNotRunningState();


  public WfExecutionObject.State workflowState();

}
```

Direct known subclasses : `de.danet.an.workflow.api.Activity.ClosedCompletedState`

Methods inherited from de.danet.an.workflow.omgcore.WfExecutionObject.State : `fromString` , `getParent` , `howClosedState` , `isSameOrSubState` , `registerState` , `textRepresentation` , `toString` , `whileOpenState` , `whyNotRunningState` , `workflowState`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Fields inherited from de.danet.an.workflow.omgcore.WfExecutionObject.State : `CLOSED` , `OPEN`

Inheritance Path. java.lang.Object-> Section A.1.40, “Class WfExecutionObject.State” [128] -> Section A.1.37, “Class WfExecutionObject.ClosedState” [120]

A.1.37.2. WfExecutionObject.ClosedState(String)

```
  protected WfExecutionObject.ClosedState(String text);
```

Parameters

`text`	Textual representation of the state

Default constructor.

A.1.37.3. ABORTED

```
  public static final WfExecutionObject.ClosedState ABORTED ;
```

Indicates that the enactment of the execution object has been aborted before normal execution. No assumptions on the state of execution objects depending on this execution object are made when enters this state.

A.1.37.4. COMPLETED

```
  public static final WfExecutionObject.ClosedState COMPLETED ;
```

When an execution object has finished its task in the overall workflow process it enters the completed state; it is assumed that all execution objects associated with that execution object are completed when it enters this state.

A.1.37.5. TERMINATED

```
  public static final WfExecutionObject.ClosedState TERMINATED ;
```

Indicates that enactment of the execution object was stopped before normal completion. It is assumed that all execution objects depending on this execution object (i.e., WfActivities contained in a WfProcess or a WfProcess implementing a WfActivity) are either completed or are terminated when it enters this state.

A.1.37.6. getParent()

```
public WfExecutionObject.State getParent();
```

Parameters

return	parent in the state hierachy

Returns the parent in the state hierachy if all states defined in this class or `null`, if this states are at the top level of the hierachy.

A.1.37.7. howClosedState()

```
public WfExecutionObject.State howClosedState();
```

Parameters

return	the sub-state of closed state

Returns the workflow substate for closed execution objects.

A.1.37.8. whileOpenState()

```
public WfExecutionObject.State whileOpenState();
```

Parameters

return	the sub-state of open state

Returns the workflow substate for open execution objects.

A.1.37.9. whyNotRunningState()

```
public WfExecutionObject.State whyNotRunningState();
```

Parameters

return	the sub-state of not-running state

Returns the workflow substate for open, not running execution objects.

A.1.37.10. workflowState()

```
  public WfExecutionObject.State workflowState();
```

Parameters

return	the workflow state

Returns the workflow state, i.e. the parent.

A.1.38. Class WfExecutionObject.NotRunningState

This class defines the sub-states of OpenState.NOT_RUNNING of a `WfExecutionObject` as returned by `whyNotRunningState()` .

A.1.38.1. Synopsis

```
 public static class de.danet.an.workflow.omgcore.WfExecutionObject.NotRunni
    implements, java.io.Serializable {
// Public Static Fields

  public static final WfExecutionObject.NotRunningState NOT_STARTED ;


  public static final WfExecutionObject.NotRunningState SUSPENDED ;

// Protected Constructors

  protected WfExecutionObject.NotRunningState(String text);

// Public Methods

  public WfExecutionObject.State getParent();


  public WfExecutionObject.State howClosedState();


  public WfExecutionObject.State whileOpenState();


  public WfExecutionObject.State whyNotRunningState();


  public WfExecutionObject.State workflowState();

}
```

Methods inherited from de.danet.an.workflow.omgcore.WfExecutionObject.OpenState : `getParent` , `howClosedState` , `whileOpenState` , `whyNotRunningState` , `workflowState`

Methods inherited from de.danet.an.workflow.omgcore.WfExecutionObject.State : `fromString` , `isSameOrSubState` , `registerState` , `textRepresentation` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Fields inherited from de.danet.an.workflow.omgcore.WfExecutionObject.OpenState : `NOT_RUNNING` , `RUNNING`

Fields inherited from de.danet.an.workflow.omgcore.WfExecutionObject.State : `CLOSED` , `OPEN`

Inheritance Path. java.lang.Object-> Section A.1.40, “Class WfExecutionObject.State” [128] -> Section A.1.39, “Class WfExecutionObject.OpenState” [125] -> Section A.1.38, “Class WfExecutionObject.NotRunningState” [123]

A.1.38.2. WfExecutionObject.NotRunningState(String)

```
protected WfExecutionObject.NotRunningState(String text);
```

Parameters

`text`	Textual representation of the state

Default constructor.

A.1.38.3. NOT_STARTED

```
public static final WfExecutionObject.NotRunningState NOT_STARTED ;
```

Provides a state after creation where the object is active and ready to be initialized and started.

A.1.38.4. SUSPENDED

```
public static final WfExecutionObject.NotRunningState SUSPENDED ;
```

Provides a state to temporarily pause the execution of the object. When an execution object is suspended, no execution objects depending on this object may be started.

A.1.38.5. getParent()

```
public WfExecutionObject.State getParent();
```

Parameters

return	parent in the state hierachy

Returns the parent in the state hierachy if all states defined in this class or `null`, if this states are at the top level of the hierachy.

A.1.38.6. howClosedState()

```
public WfExecutionObject.State howClosedState();
```

Parameters

return	the sub-state of closed state

Returns the workflow substate for closed execution objects.

A.1.38.7. whileOpenState()

```
public WfExecutionObject.State whileOpenState();
```

Parameters

return	the sub-state of open state

Returns the workflow substate for open execution objects.

A.1.38.8. whyNotRunningState()

```
public WfExecutionObject.State whyNotRunningState();
```

Parameters

return	the sub-state of not-running state

Returns the workflow substate for open, not running execution objects.

A.1.38.9. workflowState()

```
public WfExecutionObject.State workflowState();
```

Parameters

return	the workflow state

Returns the workflow state, i.e. the grandparent.

A.1.39. Class WfExecutionObject.OpenState

This class defines the sub-states of State.OPEN of a `WfExecutionObject` as returned by `whileOpenState()` .

A.1.39.1. Synopsis

```
 public static class de.danet.an.workflow.omgcore.WfExecutionObject.OpenStat
    implements, java.io.Serializable {
// Public Static Fields

  public static final WfExecutionObject.OpenState NOT_RUNNING ;


  public static final WfExecutionObject.OpenState RUNNING ;

// Protected Constructors

  protected WfExecutionObject.OpenState(String text);

// Public Methods
```

```
  public WfExecutionObject.State getParent();

  public WfExecutionObject.State howClosedState();

  public WfExecutionObject.State whileOpenState();

  public WfExecutionObject.State whyNotRunningState();

  public WfExecutionObject.State workflowState();

}
```

Direct known subclasses : `de.danet.an.workflow.omgcore.WfExecutionObject.NotRunningState`

Methods inherited from de.danet.an.workflow.omgcore.WfExecutionObject.State : `fromString` , `getParent` , `howClosedState` , `isSameOrSubState` , `registerState` , `textRepresentation` , `toString` , `whileOpenState` , `whyNotRunningState` , `workflowState`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Fields inherited from de.danet.an.workflow.omgcore.WfExecutionObject.State : `CLOSED` , `OPEN`

Inheritance Path. java.lang.Object-> Section A.1.40, “Class WfExecutionObject.State” [128] -> Section A.1.39, “Class WfExecutionObject.OpenState” [125]

A.1.39.2. WfExecutionObject.OpenState(String)

```
  protected WfExecutionObject.OpenState(String text);
```

Parameters

`text`	Textual representation of the state

Default constructor.

A.1.39.3. NOT_RUNNING

```
  public static final WfExecutionObject.OpenState NOT_RUNNING ;
```

Object is active and quiescent, but ready to execute.

A.1.39.4. RUNNING

```
  public static final WfExecutionObject.OpenState RUNNING ;
```

The object is active and executing in the workflow.

A.1.39.5. getParent()

```
public WfExecutionObject.State getParent();
```

Parameters

return	parent in the state hierachy

Returns the parent in the state hierachy if all states defined in this class or `null`, if this states are at the top level of the hierachy.

A.1.39.6. howClosedState()

```
public WfExecutionObject.State howClosedState();
```

Parameters

return	the sub-state of closed state

Returns the workflow substate for closed execution objects.

A.1.39.7. whileOpenState()

```
public WfExecutionObject.State whileOpenState();
```

Parameters

return	the sub-state of open state

Returns the workflow substate for open execution objects.

A.1.39.8. whyNotRunningState()

```
public WfExecutionObject.State whyNotRunningState();
```

Parameters

return	the sub-state of not-running state

Returns the workflow substate for open, not running execution objects.

A.1.39.9. workflowState()

```
public WfExecutionObject.State workflowState();
```

Parameters

return	the workflow state

Returns the workflow state, i.e. the parent.

A.1.40. Class WfExecutionObject.State

This class defines the top-hierachy possible states of a `WfExecutionObject` as returned by `workflowState()` .

A.1.40.1. Synopsis

```
 public static class de.danet.an.workflow.omgcore.WfExecutionObject.Stateimplement
// Public Static Fields

  public static final WfExecutionObject.State CLOSED ;


  public static final WfExecutionObject.State OPEN ;

// Protected Constructors

  protected WfExecutionObject.State(String text);

// Public Static Methods

  public static WfExecutionObject.State fromString(String text)
    throws InvalidStateException;

// Public Methods

  public WfExecutionObject.State getParent();


  public WfExecutionObject.State howClosedState();


  public boolean isSameOrSubState(WfExecutionObject.State s);


  public String toString();


  public WfExecutionObject.State whileOpenState();


  public WfExecutionObject.State whyNotRunningState();


  public WfExecutionObject.State workflowState();

// Protected Methods

  protected static void registerState(WfExecutionObject.State state);


  protected final String textRepresentation();

}
```

Direct known subclasses : `de.danet.an.workflow.omgcore.WfExecutionObject.ClosedState` , `de.danet.an.workflow.omgcore.WfExecutionObject.OpenState`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.1.40, “Class WfExecutionObject.State” [128]

A.1.40.2. WfExecutionObject.State(String)

```
protected WfExecutionObject.State(String text);
```

Parameters

`text`	Textual representation of the state

Default constructor.

A.1.40.3. CLOSED

```
public static final WfExecutionObject.State CLOSED ;
```

Reflects that the object is finished and inactive.

A.1.40.4. OPEN

```
public static final WfExecutionObject.State OPEN ;
```

To reflect that the object is active and not finished.

A.1.40.5. fromString(String)

```
public static WfExecutionObject.State fromString(String text)
  throws InvalidStateException;
```

Parameters

`text`	state name to search
return	state object

Exceptions

`InvalidStateException`	if `text` is not a valid state name.

Get a state by name.

A.1.40.6. getParent()

```
public WfExecutionObject.State getParent();
```

Parameters

return	parent in the state hierachy

Returns the parent in the state hierachy if all states defined in this class or `null`, if this states are at the top level of the hierachy.

A.1.40.7. howClosedState()

```
public WfExecutionObject.State howClosedState();
```

Parameters

return	the state as `State` object.

Returns the workflow substate for closed execution objects.

A.1.40.8. isSameOrSubState(WfExecutionObject.State)

```
public boolean isSameOrSubState(WfExecutionObject.State s);
```

Parameters

`s`	State to compare
return	`true` if same or sub-state `false` else.

Checks if this state is the same state as the given state or a substate of the given state.

A.1.40.9. registerState(WfExecutionObject.State)

```
protected static void registerState(WfExecutionObject.State state);
```

Parameters

`state`	the `State` to be registered.

Register state for `fromString` evaluation.

Additionally introduced sub-states must call this method during their initialization if the textual representation is to be recognized in the `fromString` method. It is up to the implementor of a derived state to assure that the sub-state is initialized before the from String method is called.

A.1.40.10. textRepresentation()

```
protected final String textRepresentation();
```

Parameters

return	the textual representation

Return the textual representation of the state.

A.1.40.11. toString()

```
public String toString();
```

Parameters

return	string representation of the state.

Returns the string representation of the state.

A.1.40.12. whileOpenState()

```
public WfExecutionObject.State whileOpenState();
```

Parameters

return	the state as `State` object.

Returns the workflow substate for open execution objects.

A.1.40.13. whyNotRunningState()

```
public WfExecutionObject.State whyNotRunningState();
```

Parameters

return	the state as `State` object.

Returns the workflow substate for open, not running execution objects.

A.1.40.14. workflowState()

```
public WfExecutionObject.State workflowState();
```

Parameters

return	the state as `State` object.

Returns the workflow state, i.e. this object.

A.1.41. Interface WfObject

A tagging interface that all omgcore interfaces must extend.

A.1.41.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfObject {
}
```

Inheritance Path. Section A.1.41, “Interface WfObject” [131]

A.1.42. Interface WfProcess

`WfProcess` is the performer of a workflow request. All workflow objects that perform work implement this interface. This interface allows work to proceed asynchronously while being monitored and controlled.

A.1.42.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfProcess extends, de.danet.an.work
// Public Methods

  public java.util.Collection activitiesInState(String state)
    throws RemoteException, InvalidStateException;


  public WfProcessMgr manager()
    throws RemoteException;


  public WfRequester requester()
    throws RemoteException;


  public ProcessData result()
    throws RemoteException, ResultNotAvailableException;


  public void setRequester(WfRequester requester)
    throws RemoteException, CannotChangeRequesterException;


  public void start()
    throws RemoteException, CannotStartException, AlreadyRunningException;


  public java.util.Collection steps()
    throws RemoteException;

}
```

Inheritance Path. Section A.1.42, “Interface WfProcess” [132]

A.1.42.2. activitiesInState(String)

```
  public java.util.Collection activitiesInState(String state)
    throws RemoteException, InvalidStateException;
```

Parameters

`state`	the given state.

return	the collection of all WfActivities.

Exceptions

`RemoteException`	if a system-level error occurs.
`InvalidStateException`	if an invalid state has been specified.

Return all `WfActivity` objects that are in a certain state.

A.1.42.3. manager()

```
public WfProcessMgr manager()
  throws RemoteException;
```

Parameters

return	the associated WfProcessMgr.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the `WfProcessMgr` associated with the `WfProcess` .

A.1.42.4. requester()

```
public WfRequester requester()
  throws RemoteException;
```

Parameters

return	the associated `WfRequester` .

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the `WfRequester` associated with this `WfProcess` .

A.1.42.5. result()

```
public ProcessData result()
  throws RemoteException, ResultNotAvailableException;
```

Parameters

return	process data representing intermediate result.

Exceptions

`RemoteException`	if a system-level error occurs.
`ResultNotAvailableException`	when the result cannot be obtained yet.

Returns the result produced by the WfProcess. In general the result is undefined until the process completes, but some processes may produce intermediate results.

A.1.42.6. setRequester(WfRequester)

```
public void setRequester(WfRequester requester)
  throws RemoteException, CannotChangeRequesterException;
```

Parameters

`requester`	new WfRequester.

Exceptions

`RemoteException`	if a system-level error occurs.
`CannotChangeRequesterException`	if ressignment of the process is not supported.

Reassigns the `WfProcess` to another `WfRequester` .

A.1.42.7. start()

```
public void start()
  throws RemoteException, CannotStartException, AlreadyRunningException;
```

Exceptions

`RemoteException`	if a system-level error occurs.
`CannotStartException`	when the process cannot be started (e.g., because it is not properly initialized).
`AlreadyRunningException`	when the process has already been started.

Initiate enactment of a WfProcess.

A.1.42.8. steps()

```
  public java.util.Collection steps()
    throws RemoteException;
```

Parameters

return	the collection of all the WfActivities.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns all WfActivities associated with this `WfProcess` .

A.1.43. Interface WfProcessMgr

A `WfProcessMgr` represents a template for a specific workflow process; it is used to create instances of a workflow process. Logically it is the factory and locator for `WfProcess` instances.

A.1.43.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfProcessMgr extends, de.dane
// Public Static Fields

  public static final int DISABLED  = 0;


  public static final int ENABLED  = 1;

// Public Methods

  public String category()
    throws RemoteException;


  public ProcessDataInfo contextSignature()
    throws RemoteException;


  public WfProcess createProcess(WfRequester requester)
    throws RemoteException, NotEnabledException, InvalidRequesterException, 


  public String description()
    throws RemoteException;


  public String name()
    throws RemoteException;


  public int processMgrState()
    throws RemoteException;


  public java.util.Collection processes()
    throws RemoteException;


  public ProcessDataInfo resultSignature()
```

```
    throws RemoteException;


  public void setProcessMgrState(int newState)
    throws RemoteException, TransitionNotAllowedException;


  public String version()
    throws RemoteException;

}
```

Inheritance Path. Section A.1.43, “Interface WfProcessMgr” [135]

A.1.43.2. DISABLED

```
public static final int DISABLED  = 0;
```

Indicates that creation of workflow processes is disabled.

A.1.43.3. ENABLED

```
public static final int ENABLED  = 1;
```

Indicates that creation of workflow processes is enabled.

A.1.43.4. category()

```
public String category()
  throws RemoteException;
```

Parameters

return	the string of category.

Exceptions

`RemoteException`	if a system-level error occurs.

Return the category of a process manager used for classification of process types. It is set when the process manager is initialized and cannot be modified.

A.1.43.5. contextSignature()

```
public ProcessDataInfo contextSignature()
  throws RemoteException;
```

Parameters

return	the process meta information.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the meta information that defines how to set the context of an instance.

A.1.43.6. createProcess(WfRequester)

```
public WfProcess createProcess(WfRequester requester)
  throws RemoteException, NotEnabledException, InvalidRequesterException, 
```

Parameters

`requester`	the requester to create process.
return	the created process.

Exceptions

`RemoteException`	if a system-level error occurs.
`NotEnabledException`	when the process manager is disabled.
`InvalidRequesterException`	when the process definition requires a `WfRequester` and an invalid `WfRequester` is supplied in the parameter.
`RequesterRequiredException`	when a `WfRequester` is being identified that cannot be a parent of instances of the process.

Create instances of a process and link its requester.

A.1.43.7. description()

```
public String description()
  throws RemoteException;
```

Parameters

return	the description.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the description of the process manager.

A.1.43.8. name()

```
  public String name()
    throws RemoteException;
```

Parameters

return	the name.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the name of the process manager.

A.1.43.9. processes()

```
  public java.util.Collection processes()
    throws RemoteException;
```

Parameters

return	a Collection object of `WfProcess` .

Exceptions

`RemoteException`	if a system-level error occurs.

Returns a collection with `WfProcess` objects from this process manager.

A.1.43.10. processMgrState()

```
  public int processMgrState()
    throws RemoteException;
```

Parameters

return	WfProcessMgr.ENABLED if creation of workflow processes is enabled, otherwise WfProcessMgr.DISABLED.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the state of the `WfProcessMgr` .

A.1.43.11. resultSignature()

```
public ProcessDataInfo resultSignature()
  throws RemoteException;
```

Parameters

return	the process meta information.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the meta information that specifies how instances will return results.

A.1.43.12. setProcessMgrState(int)

```
public void setProcessMgrState(int newState)
  throws RemoteException, TransitionNotAllowedException;
```

Parameters

`newState`	the new state.

Exceptions

`RemoteException`	if a system-level error occurs.
`TransitionNotAllowedException`	if the transition is not allowed

Set the new state of this process manager.

A.1.43.13. version()

```
public String version()
  throws RemoteException;
```

Parameters

return	the version.

Exceptions

`RemoteException`	if a system-level error occurs.

Return the version attribute of a process manager used to distinguish between different versions of a process model.

A.1.44. Interface WfRequester

`WfRequester` is the interface that has a direct concern with the execution and results of a workflow process. It represents the request for some work to be done. Its performer, a `WfProcess` is expected to handle its request and communicate significant status changes; in particular to inform the requester when it has completed performing the request work. The support of `WfRequester` s in a workflow engine implementation is complicated because the `receiveEvent` method reverses the client server relationship and because an object from the application space must be stored by the server.

Implementations of `WfRequester` are therefore subject to the following restrictions:

- The implementation must implement <code>java.io.Serializable</code>. Make sure not to use any attributes that are not serializable.
- In order for deserialization to work, the implementation's class file must be in the classpath of the application server.
- The implementation must provide proper <code>equals</code> and <code>hashCode</code> methods.

Most applications will simply use the requester provided as `DefaultRequester` .

A.1.44.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfRequester extends, de.danet.an.wo
// Public Methods

  public boolean isMemberOfPerformers(WfProcess member)
    throws RemoteException;


  public java.util.Collection performers()
    throws RemoteException;

}
```

Inheritance Path. Section A.1.44, “Interface WfRequester” [140]

A.1.44.2. isMemberOfPerformers(WfProcess)

```
  public boolean isMemberOfPerformers(WfProcess member)
    throws RemoteException;
```

Parameters

`member`	the process in question.
return	`true` if the `process` is among the performers of this requester.

Exceptions

`RemoteException`	if a system-level error occurs.

Check if the given process is among the performers of this requester.

A.1.44.3. performers()

```
  public java.util.Collection performers()
    throws RemoteException;
```

Parameters

return	A collection of associated performers.

Exceptions

`RemoteException`	if a system-level error occurs.

Return all performers associated with this requester.

A.1.45. Interface WfResource

`WfResource` is an abstraction that represents a person or thing that will potentially accept an assignment to an activity.

A.1.45.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfResource extends, de.danet.a
// Public Methods

  public boolean isMemberOfWorkItems(WfAssignment member)
    throws RemoteException;


  public void release(WfAssignment fromAssignment,
                      String releaseInfo)
    throws RemoteException, NotAssignedException;


  public String resourceKey()
    throws RemoteException;


  public String resourceName()
    throws RemoteException;


  public java.util.Collection workItems()
    throws RemoteException;

}
```

Inheritance Path. Section A.1.45, “Interface WfResource” [141]

A.1.45.2. isMemberOfWorkItems(WfAssignment)

```
public boolean isMemberOfWorkItems(WfAssignment member)
  throws RemoteException;
```

Parameters

`member`	the given `WfAssignment` .
return	`true` if the association exists.

Exceptions

`RemoteException`	if a system-level error occurs.

Checks if a given `WfAssignment` is associated with this resource.

A.1.45.3. release(WfAssignment, String)

```
public void release(WfAssignment fromAssignment,
                    String releaseInfo)
  throws RemoteException, NotAssignedException;
```

Parameters

`fromAssignment`	the specific assignment.
`releaseInfo`	specifies additional information on the reason for realizing the resource as input.

Exceptions

`NotAssignedException`	if the resource is not associated with the given assignment.
`RemoteException`	if a system-level error occurs.

Signals to the resource that it is no longer needed for a specific assignment. It is assumed that this operation is invoked when an assignment is deleted or when an assignment is reassigned to another resource.

(The description from the OMG specification is a bit unclear about the nature of this method. Is it used to *signal* the release, or does it *cause* the release, i.e. does it update the storage of assignments. Due to the fact that this method throws `NotAssignedException` , which is possible only if this method interfaces with the assignment storage, we have opted for the latter.)

A.1.45.4. resourceKey()

```
public String resourceKey()
  throws RemoteException;
```

Parameters

return	the resource key.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the resource key. The resource key identifies a resource within a given business domain. It is assumed that resources are defined in the same business domain as the workflow process they are associated with.

The key is set when the object is initialized; modification of the key can be done in the context of a resource management facility.

A.1.45.5. resourceName()

```
public String resourceName()
  throws RemoteException;
```

Parameters

return	the name of the resource.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns a human readable, descriptive name of the resource.

A.1.45.6. workItems()

```
public java.util.Collection workItems()
  throws RemoteException;
```

Parameters

return	the associated `WfAssignments` s.

Exceptions

`RemoteException`	if a system-level error occurs.

This method returns the `WfAssignments` s associated with a resource.

A.1.46. Interface WfStateAuditEvent

A `WfStateAuditEvent` provides an audit record of information for a `WfExecutionObject` 's state change.

A.1.46.1. Synopsis

```
 public interface de.danet.an.workflow.omgcore.WfStateAuditEvent extends, de.danet
// Public Methods

  public String newState();


  public String oldState();

}
```

Inheritance Path. Section A.1.46, "Interface WfStateAuditEvent" [143]

A.1.46.2. newState()

```
  public String newState();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `newState` .

A.1.46.3. oldState()

```
  public String oldState();
```

Parameters

return	the current value of the attribute.

Returns the current value of the attribute `oldState` .

A.2. Package de.danet.an.workflow.api

This package defines the the workflow API provided by Danet's workflow component. The API extends the `core API` derived from the OMG "Workflow Management Facility Sepcification, V1.2". We consider the interfaces and classes in this package (together with the interfaces and classes in the core API) to be useable as a general Java workflow API, i.e. if there was a JSR for a Java workflow API, we think the merger of these two packages would be a good starting point.

Trying to implement workflow clients using the OMG interface, we found that it lacks some functions that are either absolutely necessary or nice to have. For some OMG interfaces we have therefore defined a corresponding interface that extends the OMG base interface.

We have also added some interface for areas that the OMG specification has purposely omitted from its scope (e.g. access to process definitions).

Finally, the OMG specification has left it open how to access the root objects of type `WfProcessMgr` . We have therefore defined a `WorkflowService` and its corresponding `factory` that provides this access (among some other useful function).

A.2.1. Additional Information

Since	V1.0

A.2.2. Interface Activity

Interface `Activity` adds some functions to the `OMG activity` .

A.2.2.1. Synopsis

```
 public interface de.danet.an.workflow.api.Activity extends, de.danet.an.wor
// Public Methods

  public void abandon(String exceptionName)
    throws RemoteException, TransitionNotAllowedException;


  public Activity.Info activityInfo()
    throws RemoteException;


  public String blockActivity()
    throws RemoteException;


  public void changeAssignment(de.danet.an.workflow.omgcore.WfResource oldRe
                               de.danet.an.workflow.omgcore.WfResource newRe
    throws RemoteException, InvalidResourceException, AlreadyAssignedExcepti


  public boolean choose()
    throws RemoteException, TransitionNotAllowedException;


  public Activity.DeadlineInfo[] deadlines()
    throws RemoteException;


  public Activity.Implementation executor()
    throws RemoteException;


  public de.danet.an.workflow.omgcore.WfResource getResource(de.danet.an.wor
    throws RemoteException;


  public String[] handledExceptions()
    throws RemoteException;


  public Activity.Implementation[] implementation()
    throws RemoteException;


  public Activity.JoinAndSplitMode joinMode()
    throws RemoteException;


  public java.util.List nextActivities()
    throws RemoteException;


  public String performer()
    throws RemoteException;
  public void removeAssignment(de.danet.an.workflow.omgcore.WfResource resource)
    throws RemoteException, InvalidResourceException, NotAssignedException;


  public Activity.JoinAndSplitMode splitMode()
    throws RemoteException;


  public ActivityUniqueKey uniqueKey()
    throws RemoteException;

}
```

Inheritance Path. Section A.2.2, “Interface Activity” [145]

A.2.2.2. abandon(String)

```
  public void abandon(String exceptionName)
    throws RemoteException, TransitionNotAllowedException;
```

Parameters

`exceptionName`	the name of the exception

Exceptions

`RemoteException`	if a system-level error occurs
`TransitionNotAllowedException`	if the activity is not executing a tool

Force the completion of this activity, leaving it in state "closed.completed.abandoned". Does nothing if the activity is in state "closed" already or has not been started ("open.not_running.not_started").

An abandoned activity is considered to be completed under exceptional circumstances. Therefore only transitions with conditions of type `EXCEPTION` or `DEFAULTEXCEPTION` are considered when evaluating the set of subsequent activities. The argument is the name of the exception which may be used to distinguish different exceptions in transition evaluation (see XPDL).

This method should be used with care. In general, exceptions have a different level of abstraction in a workflow process description than in Java programming. The author of a workflow process should not have to know about e.g. a "SAXException". But he may know what to do in case of a "ResultInvalid" exception (though this kind of problem should only arise during development anyway).

This method may only be called during tool execution. Note that calling this method does not terminate tool execution, i.e. the method will return. A tool agent should, however, not try to do anything with the activity any more after calling this method.

A.2.2.3. activityInfo()

```
  public Activity.Info activityInfo()
    throws RemoteException;
```

Parameters

return	the resulting `Activity.Info` value

Exceptions

`RemoteException`	if a system-level error occurs

This method returns all available information about the activity in a single operation.

A.2.2.4. blockActivity()

```
public String blockActivity()
  throws RemoteException;
```

Parameters

return	an identification of the block activity that caused this activity to be instantiated or `null` if this activity was not instantiated as part of an activity set

Exceptions

`RemoteException`	if a system-level error occurs

Returns the key of the "parent" block activity. All activities implicitly created by a block activity share the same block activity key.

Note that there need not be an activity with the returned key, as an activity set is actually a template describing how to implement block activities. The information obtained can mainly be used to group all activities that have been instantiated as part of an activity set.

A.2.2.5. changeAssignment(WfResource, WfResource)

```
public void changeAssignment(de.danet.an.workflow.omgcore.WfResource oldRe
                             de.danet.an.workflow.omgcore.WfResource newRe
  throws RemoteException, InvalidResourceException, AlreadyAssignedExcepti
```

Parameters

`oldResource`	the resource that has its assignment removed
`newResource`	the resource to be assigned

Exceptions

`RemoteException`	if a system-level error occurs
`InvalidResourceException`	if the resource is invalid. As the environment is a concurrent

	multi user environment, `WfResource` objects may become invalid.
`AlreadyAssignedException`	if the assignment already exists
`NotAssignedException`	if there is no assignment to the old resource

Change an assignment for enacting the activity. This method calls the corresponding method of the resource assignment service and creates the appropriate audit event.

This method is intended to be used by resource assignment systems for implementing `WfAssignment.setAssignee` . Resource assignment systems are responsible for implementing `WfAssignment` and could therefore perform the reassignment directly; this would, however, leave the generation of notifications unexecuted.

Clients should not use this method but rather call `WfAssignment.setAssignee` .

A.2.2.6. choose()

```
public boolean choose()
  throws RemoteException, TransitionNotAllowedException;
```

Parameters

return	`true` if the activity could be made the effectively chosen one

Exceptions

`RemoteException`	if a system-level error occurs
`TransitionNotAllowedException`	if the activity is neither running nor suspended

Makes this activity the chosen one in a set of activities started by an AND split with the "deferred choice" option set. All other activities in the set are reset to their initial state.

If the activity does not participate in a deferred choice, this method does nothing and returns `true` .

A.2.2.7. deadlines()

```
public Activity.DeadlineInfo[] deadlines()
  throws RemoteException;
```

Parameters

return	the deadlines

Exceptions

`RemoteException`	if a system-level error occurs

Returns the deadlines defined for this activity.

A.2.2.8. executor()

```
public Activity.Implementation executor()
  throws RemoteException;
```

Parameters

return	current executor or `null` if no executor running

Exceptions

`RemoteException`	if a system-level error occurs

Returns the current executor.

A.2.2.9. getResource(WfAssignment)

```
public de.danet.an.workflow.omgcore.WfResource getResource(de.danet.an.wor
  throws RemoteException;
```

Parameters

`asnmnt`	the assignment
return	the resource

Exceptions

`RemoteException`	if a system-level error occurs.

Since	1.3.4

Get the resource associated with an Assignment. The method calls the corresponding method of the resource assignment service. This method is intended to be used by resource assignment systems for implementing `WfAssignment.assignee` .

Clients should not use this method but rather call `WfAssignment.assignee` .

A.2.2.10. handledExceptions()

```
public String[] handledExceptions()
  throws RemoteException;
```

Parameters

return	handled exceptions

Exceptions

`RemoteException`	if a system-level error occurs

Returns the names of the exceptions handled by this activity.

A.2.2.11. implementation()

```
public Activity.Implementation[] implementation()
  throws RemoteException;
```

Parameters

return	an array of `Implementation` }s or `null` if no implementation is defined

Exceptions

`RemoteException`	if a system-level error occurs

Returns the implementation of the activity as `Implementation` s.

A.2.2.12. joinMode()

```
public Activity.JoinAndSplitMode joinMode()
  throws RemoteException;
```

Parameters

return	join mode

Exceptions

`RemoteException`	if a system-level error occurs

Returns the join mode.

A.2.2.13. nextActivities()

```
public java.util.List nextActivities()
  throws RemoteException;
```

Parameters

return	the list of `Activity` objects.

Exceptions

`RemoteException`	if a system-level error occurs

Returns the list of activities that may follow this activity, i.e. to which transitions exist.

A.2.2.14. performer()

```
public String performer()
  throws RemoteException;
```

Parameters

return	performer as string

Exceptions

`RemoteException`	if a system-level error occurs

Returns the performer as string.

A.2.2.15. removeAssignment(WfResource)

```
public void removeAssignment(de.danet.an.workflow.omgcore.WfResource resou
  throws RemoteException, InvalidResourceException, NotAssignedException;
```

Parameters

`resource`	the resource whose assignment is to be canceled

Exceptions

`RemoteException`	if a system-level error occurs
`InvalidResourceException`	if the resource is invalid. As the environment is a concurrent multi user environment, `WfResource` objects may become invalid.
`NotAssignedException`	if there is no such assignment

Removes an assignment for enacting the activity. This method calls the corresponding method of the resource assignment service and creates the appropriate audit event.

This method is intended to be used by resource management systems for implementing

`WfResource.release` .

Clients should not use this method but rather call `WfResource.release` .

A.2.2.16. splitMode()

```
  public Activity.JoinAndSplitMode splitMode()
    throws RemoteException;
```

Parameters

return	split mode

Exceptions

`RemoteException`	if a system-level error occurs

Returns the split mode.

A.2.2.17. uniqueKey()

```
  public ActivityUniqueKey uniqueKey()
    throws RemoteException;
```

Parameters

return	value of uniqueKey

Exceptions

`RemoteException`	if a system-level error occurs

Return a unique key for the activity. (Note that the OMG interface defines the key returned by the `key()` method as unique within the scope of the containing process only.)

A.2.3. Class Activity.ClosedCompletedState

This class defines the sub-states of ClosedState.COMPLETED of a `WfExecutionObject` . These substates are an extention of the predefined omg states.

A.2.3.1. Synopsis

```
 public static class de.danet.an.workflow.api.Activity.ClosedCompletedState extend
    implements, java.io.Serializable {
// Public Static Fields

  public static final Activity.ClosedCompletedState ABANDONED ;


  public static final Activity.ClosedCompletedState NORMAL ;
```

```
// Protected Constructors

  protected Activity.ClosedCompletedState(String text);

// Public Methods

  public de.danet.an.workflow.omgcore.WfExecutionObject.State getParent();

  public de.danet.an.workflow.omgcore.WfExecutionObject.State howClosedState

  public de.danet.an.workflow.omgcore.WfExecutionObject.State whileOpenState

  public de.danet.an.workflow.omgcore.WfExecutionObject.State whyNotRunningS

  public de.danet.an.workflow.omgcore.WfExecutionObject.State workflowState(

}
```

Methods inherited from de.danet.an.workflow.omgcore.WfExecutionObject.ClosedState : `getParent` , `howClosedState` , `whileOpenState` , `whyNotRunningState` , `work-flowState`

Methods inherited from de.danet.an.workflow.omgcore.WfExecutionObject.State : `from-String` , `isSameOrSubState` , `registerState` , `textRepresentation` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Fields inherited from de.danet.an.workflow.omgcore.WfExecutionObject.ClosedState : `ABOR-TED` , `COMPLETED` , `TERMINATED`

Fields inherited from de.danet.an.workflow.omgcore.WfExecutionObject.State : `CLOSED` , `OPEN`

Inheritance Path. java.lang.Object-> Section A.1.40, “Class WfExecutionObject.State” [128] -> Section A.1.37, “Class WfExecutionObject.ClosedState” [120] -> Section A.2.3, “Class Activity.ClosedCompletedState” [152]

A.2.3.2. Activity.ClosedCompletedState(String)

```
  protected Activity.ClosedCompletedState(String text);
```

Parameters

`text`	Textual representation of the state

Default constructor.

A.2.3.3. ABANDONED

```
  public static final Activity.ClosedCompletedState ABANDONED ;
```

Provides a state indicating that the activity was completed by an exception.

A.2.3.4. NORMAL

```
public static final Activity.ClosedCompletedState NORMAL ;
```

Provides a state indicating that the activity was completed normally.

A.2.3.5. getParent()

```
public de.danet.an.workflow.omgcore.WfExecutionObject.State getParent();
```

Parameters

return	parent in the state hierachy

Returns the parent in the state hierachy if all states defined in this class or `null`, if this states are at the top level of the hierachy.

A.2.3.6. howClosedState()

```
public de.danet.an.workflow.omgcore.WfExecutionObject.State howClosedState();
```

Parameters

return	the closed state.

Returns the workflow substate for closed execution objects.

A.2.3.7. whileOpenState()

```
public de.danet.an.workflow.omgcore.WfExecutionObject.State whileOpenState();
```

Parameters

return	the open state.

Returns the workflow substate for open execution objects.

A.2.3.8. whyNotRunningState()

```
public de.danet.an.workflow.omgcore.WfExecutionObject.State whyNotRunningState()
```

Parameters

return	the why not running state.

Returns the workflow substate for open, not running execution objects.

A.2.3.9. workflowState()

```
  public de.danet.an.workflow.omgcore.WfExecutionObject.State workflowState(
```

Parameters

return	the workflow state.

Returns the workflow state, i.e. the grandparent.

A.2.4. Class Activity.DeadlineInfo

Class `DeadlineInfo` describes all properties of a deadline.

A.2.4.1. Synopsis

```
 public static class de.danet.an.workflow.api.Activity.DeadlineInfoimplement
// Public Static Fields

  public static final int ASYNCHR  = 1;


  public static final int STATE_ACTIVE  = 2;


  public static final int STATE_CANCELED  = 3;


  public static final int STATE_INITIAL  = 0;


  public static final int STATE_REACHED  = 1;


  public static final int SYNCHR  = 2;

// Public Constructors

  public Activity.DeadlineInfo(int executionMode,
                               String exceptionName,
                               String condition,
                               int state);

// Public Methods

  public String getCondition();


  public String getExceptionName();


  public int getExecutionMode();


  public int getState();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.4, “Class Activity.DeadlineInfo” [155]

A.2.4.2. Activity.DeadlineInfo(int, String, String, int)

```
public Activity.DeadlineInfo(int executionMode,
                             String exceptionName,
                             String condition,
                             int state);
```

Parameters

`executionMode`	the execution mode
`exceptionName`	the exception to be thrown
`condition`	the condition
`state`	the current state

Creates a new `DeadlineInfo` instance with the given values.

A.2.4.3. ASYNCHR

```
public static final int ASYNCHR  = 1;
```

Denotes asynchronous execution of the deadline.

A.2.4.4. STATE_ACTIVE

```
public static final int STATE_ACTIVE  = 2;
```

Deadline is running.

A.2.4.5. STATE_CANCELED

```
public static final int STATE_CANCELED  = 3;
```

Deadline has been canceled.

A.2.4.6. STATE_INITIAL

```
public static final int STATE_INITIAL  = 0;
```

Deadline is in initial state.

A.2.4.7. STATE_REACHED

```
public static final int STATE_REACHED  = 1;
```

Deadline has been reached.

A.2.4.8. SYNCHR

```
public static final int SYNCHR  = 2;
```

Denotes synchronous execution of the deadline.

A.2.4.9. getCondition()

```
public String getCondition();
```

Parameters

return	value of Condition.

Get the value of Condition.

A.2.4.10. getExceptionName()

```
public String getExceptionName();
```

Parameters

return	value of ExceptionName.

Get the value of ExceptionName.

A.2.4.11. getExecutionMode()

```
public int getExecutionMode();
```

Parameters

return	value of execution.

Get the value of execution, one of `ASYNCHR` or `SYNCHR` .

A.2.4.12. getState()

```
public int getState();
```

Parameters

return	value of state.

Get the value of state, one of `STATE_INITIAL` , `STATE_REACHED` or `STATE_CANCELED` .

A.2.5. Interface Activity.Implementation

The super interface of possible activity implementation descriptions.

A.2.5.1. Synopsis

```
 public static interface de.danet.an.workflow.api.Activity.Implementation extends,
}
```

Inheritance Path. Section A.2.5, “Interface Activity.Implementation” [157]

A.2.6. Class Activity.Info

Class `Info` combines various informational attributes about an activity in a single structure for efficient retrieval.

A.2.6.1. Synopsis

```
 public static class de.danet.an.workflow.api.Activity.Infoimplements, java.io.Ser
// Public Constructors

  public Activity.Info(ActivityUniqueKey key,
                       String actName,
                       String actDesc,
                       int actPrio,
                       java.util.Date actLastTime,
                       String procName,
                       String procDesc);

// Public Methods

  public String description();


  public java.util.Date lastStateTime();


  public String name();


  public int priority();


  public String processDescription();


  public String processName();


  public ActivityUniqueKey uniqueKey();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.6, “Class Activity.Info” [158]

A.2.6.2. Activity.Info(ActivityUniqueKey, String, String, int, Date, String, String)

```
  public Activity.Info(ActivityUniqueKey key,
                       String actName,
                       String actDesc,
                       int actPrio,
                       java.util.Date actLastTime,
                       String procName,
                       String procDesc);
```

Parameters

`key`	the unique key.
`actName`	the activity name.
`actDesc`	the activity description.
`actPrio`	the priority of the activity.
`actLastTime`	the date of the last state change.
`procName`	the name of the containing process.
`procDesc`	the description of the containing process.

Creates a new `Info` instance with the given values.

A.2.6.3. description()

```
public String description();
```

Parameters

return	value of activityDescription.

Get the value of activityDescription.

A.2.6.4. lastStateTime()

```
public java.util.Date lastStateTime();
```

Parameters

return	value of lastStateTime.

Get the value of lastStateTime.

A.2.6.5. name()

```
public String name();
```

Parameters

return	value of activityName.

Get the value of activityName.

A.2.6.6. priority()

```
public int priority();
```

Parameters

return	value of activityPriority.

Get the value of activityPriority.

A.2.6.7. processDescription()

```
public String processDescription();
```

Parameters

return	value of processDescription.

Get the value of processDescription.

A.2.6.8. processName()

```
public String processName();
```

Parameters

return	value of processName.

Get the value of processName.

A.2.6.9. uniqueKey()

```
public ActivityUniqueKey uniqueKey();
```

Parameters

return	value of uniqueKey.

Get the value of uniqueKey.

A.2.7. Class Activity.JoinAndSplitMode

This class defines the join and split modes for an activity.

A.2.7.1. Synopsis

```
 public static class de.danet.an.workflow.api.Activity.JoinAndSplitModeimple
// Public Static Fields

  public static final Activity.JoinAndSplitMode AND ;


  public static final Activity.JoinAndSplitMode XOR ;

// Protected Constructors

  protected Activity.JoinAndSplitMode(String text);

// Public Static Methods

  public static Activity.JoinAndSplitMode fromString(String text)
    throws IllegalArgumentException;

// Public Methods

  public final boolean isAND();


  public final boolean isXOR();


  public final String toString();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.7, “Class Activity.JoinAndSplitMode” [160]

A.2.7.2. Activity.JoinAndSplitMode(String)

```
  protected Activity.JoinAndSplitMode(String text);
```

Parameters

`text`	textual representation of the mode.

Default constructor.

A.2.7.3. AND

```
  public static final Activity.JoinAndSplitMode AND ;
```

AND join or split.

A.2.7.4. XOR

```
  public static final Activity.JoinAndSplitMode XOR ;
```

XOR join or split.

A.2.7.5. fromString(String)

```
public static Activity.JoinAndSplitMode fromString(String text)
  throws IllegalArgumentException;
```

Parameters

`text`	mode name.
return	mode object

Exceptions

`IllegalArgumentException`	if `text` is not a valid mode name.

Get the join/split mode by name.

A.2.7.6. isAND()

```
public final boolean isAND();
```

Parameters

return	`true` if the mode is "AND".

Checks if the mode is "AND".

A.2.7.7. isXOR()

```
public final boolean isXOR();
```

Parameters

return	`true` if the mode is "XOR".

Checks if the mode is "XOR".

A.2.7.8. toString()

```
public final String toString();
```

Parameters

return	mode as text

Returns the mode as text.

A.2.8. Class Activity.StartFinishMode

This class defines the values for start and finish mode for an `Activity` .

A.2.8.1. Synopsis

```
 public static class de.danet.an.workflow.api.Activity.StartFinishModeimplem
// Public Static Fields

  public static final Activity.StartFinishMode AUTOMATIC ;


  public static final Activity.StartFinishMode MANUAL ;

// Protected Constructors

  protected Activity.StartFinishMode(String text);

// Public Static Methods

  public static Activity.StartFinishMode fromString(String mode);

// Public Methods

  public String toString();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.8, “Class Activity.StartFinishMode” [163]

A.2.8.2. Activity.StartFinishMode(String)

```
  protected Activity.StartFinishMode(String text);
```

Parameters

`text`	textual representation of the activity mode.

Default constructor.

A.2.8.3. AUTOMATIC

```
  public static final Activity.StartFinishMode AUTOMATIC ;
```

Triggered implicitly by the system.

A.2.8.4. MANUAL

```
  public static final Activity.StartFinishMode MANUAL ;
```

Triggered explicitly by the user.

A.2.8.5. fromString(String)

```
  public static Activity.StartFinishMode fromString(String mode);
```

Parameters

`mode`	the mode to convert.
return	the result.

Convert a string to a `StartFinishMode` .

A.2.8.6. toString()

```
  public String toString();
```

Parameters

return	mode as text

Returns the mode as text.

A.2.9. Interface Activity.SubFlowImplementation

This interface describes the implementation of an activity by a sub flow.

A.2.9.1. Synopsis

```
 public static interface de.danet.an.workflow.api.Activity.SubFlowImplementation e
// Public Static Fields

  public static final int ASYNCHR  = 1;

  public static final int SYNCHR  = 2;

// Public Methods

  public int execution();

  public String packageId();

  public String processId();

  public String processKey();

}
```

Inheritance Path. Section A.2.9, “Interface Activity.SubFlowImplementation” [164]

A.2.9.2. ASYNCHR

```
public static final int ASYNCHR  = 1;
```

Denotes asynchronous execution of subflow.

A.2.9.3. SYNCHR

```
public static final int SYNCHR  = 2;
```

Denotes synchronous execution of subflow.

A.2.9.4. execution()

```
public int execution();
```

Parameters

return	either `ASYNCHR` or `SYNCHR` .

Return the execution mode.

A.2.9.5. packageId()

```
public String packageId();
```

Parameters

return	value of package id.

Return the package id of the subflow. Can be used together with the `processId` to lookup the process definition using `ProcessDefinitionDirectory.lookupProcessDefinition` .

A.2.9.6. processId()

```
public String processId();
```

Parameters

return	value of process id.

Return the process id of the subflow. Can be used together with the `packageId` to lookup the process definition using `ProcessDefinitionDirectory.lookupProcessDefinition` .

A.2.9.7. processKey()

```
public String processKey();
```

Parameters

return	the key or `null` if no process is running.

Return the key of the invoked process if it has been started.

A.2.10. Interface Activity.ToolImplementation

This interface describes the implementation of an activity by a tool.

A.2.10.1. Synopsis

```
 public static interface de.danet.an.workflow.api.Activity.ToolImplementation exte
// Public Methods

  public String description();


  public String id();

}
```

Inheritance Path. Section A.2.10, “Interface Activity.ToolImplementation” [166]

A.2.10.2. description()

```
  public String description();
```

Parameters

return	value of description.

Return the implementation description.

A.2.10.3. id()

```
  public String id();
```

Parameters

return	value of id.

Return the tool id. The id can be mapped to a tool definition by `ProcessDefinition.applicationById` .

A.2.11. Class ActivityUniqueKey

This class implements a unique activity key. The OMG interface defines the key returned by the `key()` method as unique within the scope of the containing process only. The key of a process in turn is unique only among the processes with a common process manager.

This class therefore combines the activity key, the process key and the process manager name to a unique activity key.

A.2.11.1. Synopsis

```
 public class de.danet.an.workflow.api.ActivityUniqueKeyimplements, java.io.S
// Public Constructors

  public ActivityUniqueKey(de.danet.an.workflow.omgcore.WfActivity activity)
    throws RemoteException;


  public ActivityUniqueKey(String managerName,
                           String processKey,
                           String activityKey);

// Public Methods

  public String activityKey();


  public boolean equals(Object other);


  public int hashCode();


  public String managerName();


  public String processKey();


  public String toString();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.11, “Class ActivityUniqueKey” [166]

A.2.11.2. ActivityUniqueKey(String, String, String)

```
  public ActivityUniqueKey(String managerName,
                           String processKey,
                           String activityKey);
```

Parameters

`managerName`	the process manager name.
`processKey`	the process key.
`activityKey`	the activity key.

Creates an instance of `ActivityUniqueKey` from the given partial keys.

A.2.11.3. ActivityUniqueKey(WfActivity)

```
public ActivityUniqueKey(de.danet.an.workflow.omgcore.WfActivity activity)
  throws RemoteException;
```

Parameters

`activity`	the `WfActivity` .

Exceptions

`RemoteException`	if a system-level error occurs.

Creates an instance of `ActivityUniqueKey` for the given activity.

A.2.11.4. activityKey()

```
public String activityKey();
```

Parameters

return	the activity key.

Return the activity key.

A.2.11.5. equals(Object)

```
public boolean equals(Object other);
```

Parameters

`other`	a `ActivityUniqueKey` value
return	`true` if objects are equal.

Two `ActivityUniqueKey` s are equal, if all attributes are equal.

A.2.11.6. hashCode()

```
public int hashCode();
```

Parameters

return	the hash code.

Calculate a hash code for a `ActivityUniqueKey` object.

A.2.11.7. managerName()

```
  public String managerName();
```

Parameters

return	the process manager name.

Return the process manager name.

A.2.11.8. processKey()

```
  public String processKey();
```

Parameters

return	the process key.

Return the process key.

A.2.11.9. toString()

```
  public String toString();
```

Parameters

return	a string representation.

Generate a string representation for debugging purposes.

A.2.12. Exception AlreadyAssignedException

This class provides ...

A.2.12.1. Synopsis

```
 public class de.danet.an.workflow.api.AlreadyAssignedException extends, jav
    implements, java.io.Serializable {
// Public Constructors

  public AlreadyAssignedException();


  public AlreadyAssignedException(String message);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.2.12, “Exception AlreadyAssignedException” [169]

A.2.12.2. AlreadyAssignedException()

```
public AlreadyAssignedException();
```

Creates an instance of `AlreadyAssignedException` with all attributes initialized to default values.

A.2.12.3. AlreadyAssignedException(String)

```
public AlreadyAssignedException(String message);
```

Parameters

`message`	the message

Creates an instance of `AlreadyAssignedException` with the given message.

A.2.13. Interface Application

This interface defines methods to access the definition of an application that participates in a workflow process.

A.2.13.1. Synopsis

```
 public interface de.danet.an.workflow.api.Application {
// Public Methods

  public String description();


  public String id();

}
```

Inheritance Path. Section A.2.13, “Interface Application” [170]

A.2.13.2. description()

```
public String description();
```

Parameters

return	The description as specified in the process definition.

Returns the description of the application.

A.2.13.3. id()

```
  public String id();
```

Parameters

return	The id as specified in the process definition.

Returns the id of the Application.

A.2.14. Interface Batch

This interface must be implemented by classes that can be run as batch.

A.2.14.1. Synopsis

```
 public interface de.danet.an.workflow.api.Batch {
// Public Methods

  public Object execute(Batch.Context ctx)
    throws InvocationTargetException;

}
```

Inheritance Path. Section A.2.14, “Interface Batch” [171]

A.2.14.2. execute(Batch.Context)

```
  public Object execute(Batch.Context ctx)
    throws InvocationTargetException;
```

Parameters

`ctx`	the execution context
return	the result as defined by the implementing class

Exceptions

`InvocationTargetException`	wraps exceptions as defined by the implementing class

Execute the batch.

A.2.15. Interface Batch.Context

This interface specifies a simple contract between a batch and its execution environment.

A.2.15.1. Synopsis

```
 public static interface de.danet.an.workflow.api.Batch.Context {
// Public Methods

  public boolean isRollbackOnly();
}
```

Inheritance Path. Section A.2.15, “Interface Batch.Context” [171]

A.2.15.2. isRollbackOnly()

```
  public boolean isRollbackOnly();
```

Parameters

return	`true` if the current transaction will be rolled back.

Indicates if the transaction the batch is running in will eventually be rolled back. Continuing the execution of a batch if this method returns `true` is pointless.

A.2.16. Exception CannotRemoveException

This exception is raised by an attempt to remove a `WfProcess` that is still in progress.

A.2.16.1. Synopsis

```
 public class de.danet.an.workflow.api.CannotRemoveException extends, java.lang.Ex
    implements, java.io.Serializable {
// Public Constructors

  public CannotRemoveException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.2.16, “Exception CannotRemoveException” [172]

A.2.16.2. CannotRemoveException(String)

```
  public CannotRemoveException(String msg);
```

Parameters

`msg`	an informational message describing the problem

Creates a new `CannotRemoveException` with the given message.

A.2.17. Interface Channel

This interface defines a named connection with a process that can be used to receive messages from activities and send messages to activities.

Note that messages sent from the workflow engine to clients on a channel may be lost when no client has opened the channel.

A.2.17.1. Synopsis

```
 public interface de.danet.an.workflow.api.Channel extends, de.danet.an.work
// Public Methods

  public String name()
    throws RemoteException;


  public Process process()
    throws InvalidKeyException, RemoteException;


  public java.util.Map receiveMessage()
    throws RemoteException;


  public java.util.Map receiveMessage(long timeout)
    throws RemoteException;


  public void sendMessage(java.util.Map msg)
    throws InvalidKeyException, InvalidDataException, RemoteException;

}
```

Inheritance Path. Section A.2.17, “Interface Channel” [172]

A.2.17.2. name()

```
  public String name()
    throws RemoteException;
```

Parameters

return	the name

Exceptions

`RemoteException`	if a system-level error occurs

Return the channel name.

A.2.17.3. process()

```
  public Process process()
    throws InvalidKeyException, RemoteException;
```

Parameters

return	the process

Exceptions

`InvalidKeyException`	if the process no longer exists
`RemoteException`	if a system-level error occurs

Return the process this channel belongs to.

A.2.17.4. receiveMessage()

```
public java.util.Map receiveMessage()
  throws RemoteException;
```

Parameters

return	the message or `null` if the process has been closed or removed

Exceptions

`RemoteException`	if a system-level error occurs

Wait for the next message from the process on this channel. The message consists of all `IN` or `IN-OUT` parameters of the sender tool, stored in the `Map` object by formal parameter name.

A.2.17.5. receiveMessage(long)

```
public java.util.Map receiveMessage(long timeout)
  throws RemoteException;
```

Parameters

`timeout`	the timeout value in milliseconds. A timeout of zero never expires.
return	the message or `null` if the process has been closed or removed or the timeout expires

Exceptions

`RemoteException`	if a system-level error occurs

Wait for the next message from the process on this channel within the specified timeout interval. The message consists of all `IN` or `INOUT` parameters of the sender tool, stored in the `Map` object by formal parameter name.

A.2.17.6. sendMessage(Map)

```
  public void sendMessage(java.util.Map msg)
    throws InvalidKeyException, InvalidDataException, RemoteException;
```

Parameters

`msg`	the message

Exceptions

`InvalidKeyException`	if the process no longer exists
`InvalidDataException`	if the message contains invalid data, i.e. entries that do not match the name of a formal parameter
`RemoteException`	if a system-level error occurs

Send a message on this channel to the process. The message will be received by an active or subsequently activated receiver tool listening on this channel. If two or more receiver tools listen on the same channel concurrently, the message delivery (only to one or to every receiver) is undefined.

The message sent is mapped to the formal `OUT` parameters of the receiver tool by matching the parameter names with the data entry names in the message.

A.2.18. Interface Configuration

Interface `Configuration` . Gives access to the configuration-methods.

A.2.18.1. Synopsis

```
 public interface de.danet.an.workflow.api.Configuration extends, java.io.Se
// Public Methods

  public String workflowEngineInstanceKey()
    throws RemoteException;

}
```

Inheritance Path. Section A.2.18, “Interface Configuration” [175]

A.2.18.2. workflowEngineInstanceKey()

```
  public String workflowEngineInstanceKey()
    throws RemoteException;
```

Parameters

return	the unique instance key

Every workflow engine has a globally unique key. This key may be used in environments with several workflow engines to uniquely identify an instance. Note that this key should be bound to the data (state) of an engine, not to a deployment address. If an engine is moved to a different machine, but continues using the same data (i.e. all running processes remain the same etc.) then it should still provide the same instance key.

A.2.19. Class DefaultProcessData

This class extends `HashMap` in order to provide a default implementation of `ProcessData` .

A.2.19.1. Synopsis

```
 public class de.danet.an.workflow.api.DefaultProcessData extends, java.util.HashMa
    implements, de.danet.an.workflow.omgcore.ProcessData, java.io.Serializable {
// Public Constructors

  public DefaultProcessData();


  public DefaultProcessData(java.util.Map procData);

// Public Methods

  public String toString();

}
```

Methods inherited from java.util.HashMap : `clear` , `clone` , `containsKey` , `containsValue` , `entrySet` , `get` , `isEmpty` , `keySet` , `put` , `putAll` , `remove` , `size` , `values`

Methods inherited from java.util.AbstractMap : `equals` , `hashCode` , `toString`

Methods inherited from java.lang.Object : `finalize` , `getClass` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.util.AbstractMap-> java.util.HashMap-> Section A.2.19, “Class DefaultProcessData” [176]

A.2.19.2. DefaultProcessData()

```
  public DefaultProcessData();
```

Creates an empty `DefaultProcessData` .

A.2.19.3. DefaultProcessData(Map)

```
  public DefaultProcessData(java.util.Map procData);
```

Parameters

`procData`	the process data that are to be placed in this `DefaultProcessData` .

Creates a `DefaultProcessData` with the same mapping as the given map.

A.2.19.4. toString()

```
  public String toString();
```

Parameters

return	string representation

Generate a string representation for debugging purposes.

A.2.20. Class DefaultRequester

This class provides an implementation of a `WfRequester` . It class may be used directly if the events that are usually delivered to a requester are of no interest.

If events are to be processed, `DefaultRequester` must be subclassed with `receiveEvent` overridden with the event handling code.

As an alternative to subclassing, a handler may be passed to the `constructor` . Events will then be forwarded to the handler. This is convenient in situation where the implementation of `WfAuditHandler` already exists and a subclass would only do the forwarding. Note that the reference to the handler is `transient` .

A.2.20.1. Synopsis

```
 public class de.danet.an.workflow.api.DefaultRequesterimplements, de.danet.a
// Public Constructors

  public DefaultRequester(WorkflowService wfs)
    throws RemoteException;


  public DefaultRequester(WorkflowService wfs,
                          de.danet.an.workflow.omgcore.WfAuditHandler hdlr)
    throws RemoteException;

// Protected Constructors

  protected DefaultRequester(WorkflowService wfs,
                             boolean register)
    throws RemoteException;

// Public Methods

  public boolean equals(Object obj);


  public int hashCode();


  public boolean isMemberOfPerformers(de.danet.an.workflow.omgcore.WfProcess
    throws RemoteException;


  public java.util.Collection performers()
    throws RemoteException;


  public void receiveEvent(de.danet.an.workflow.omgcore.WfAuditEvent wfAudit
    throws InvalidPerformerException, RemoteException;


  public String toString();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.20, “Class DefaultRequester” [177]

A.2.20.2. DefaultRequester(WorkflowService)

```
public DefaultRequester(WorkflowService wfs)
  throws RemoteException;
```

Parameters

`wfs`	the workflow service.

Exceptions

`RemoteException`	if thrown during `requester registration` .

Creates a `DefaultRequester` for use with the given workflow service. The created requester will not receive events until registered with a `WorkflowService` .

A.2.20.3. DefaultRequester(WorkflowService, boolean)

```
protected DefaultRequester(WorkflowService wfs,
                           boolean register)
  throws RemoteException;
```

Parameters

`wfs`	the workflow service.
`register`	if `true` the requester will be registered at the workflow service.

Exceptions

`RemoteException`	if thrown during `requester registration` .

Creates a `DefaultRequester` for use with the given workflow service. The created requester is automatically registered at the given `WorkflowService` .

A.2.20.4. DefaultRequester(WorkflowService, WfAuditHandler)

```
public DefaultRequester(WorkflowService wfs,
                        de.danet.an.workflow.omgcore.WfAuditHandler hdlr)
  throws RemoteException;
```

Parameters

`wfs`	the workflow service.
`hdlr`	the event handler.

Exceptions

`RemoteException`	if thrown during `requester registration` .

Creates a `DefaultRequester` for use with the given workflow service. The created requester will forward received events to the given handler.

A.2.20.5. equals(Object)

```
public boolean equals(Object obj);
```

Parameters

`obj`	the other object.
return	`true` if the objects are equal.

Indicates whether some other object is "equal to" this one.

A.2.20.6. hashCode()

```
public int hashCode();
```

Parameters

return	the hash code

Returns a hash code value for the object.

A.2.20.7. isMemberOfPerformers(WfProcess)

```
public boolean isMemberOfPerformers(de.danet.an.workflow.omgcore.WfProcess
  throws RemoteException;
```

Specified by: Method isMemberOfPerformers in interface WfRequester

Parameters

`member`	the process in question.
return	`true` if the `process` is among the performers of this requester.

Exceptions

`RemoteException`	if a system-level error occurs.

Description copied from interface: <link linkend="METHOD-DE.DANET.AN.WORKFLOW.OMGCORE.WFREQUESTER.ISMEMBEROFPERFORMERS-DE.DANET.AN.WORKFLOW.OMGCORE.WFPROCESS-">isMemberOfPerformers</link>

Check if the given process is among the performers of this requester.

A.2.20.8. performers()

```
  public java.util.Collection performers()
    throws RemoteException;
```

Specified by: Method performers in interface WfRequester

Parameters

return	A collection of associated performers.

Exceptions

`RemoteException`	if a system-level error occurs.

Description copied from interface: <link linkend="METHOD-DE.DANET.AN.WORKFLOW.OMGCORE.WFREQUESTER.PERFORMERS--">performers</link>

Return all performers associated with this requester.

A.2.20.9. toString()

```
  public String toString();
```

Parameters

return	the string representation.

Return a string representation for debugging purposes.

A.2.21. Interface EventSubscriber

An `EventSubscriber` represents a connection to the workflow engine's event queue. From its creation with `WorkflowService.createEventSubscriber` until its destruction with `WorkflowService.release` all events from the workflow engine are delivered to the `EventSubscriber` . `EventSubscriber` s should be released using `WorkflowService.release` when no longer needed as they may consume considerable resources.

A.2.21.1. Synopsis

```
 public interface de.danet.an.workflow.api.EventSubscriber extends, de.danet.an.wo
// Public Methods

  public de.danet.an.workflow.omgcore.WfAuditEvent receive()
```

```
    throws IOException;

  public de.danet.an.workflow.omgcore.WfAuditEvent receive(long timeout)
    throws IOException;

  public de.danet.an.workflow.omgcore.WfAuditEvent receiveNoWait()
    throws IOException;

  public void setEventHandler(de.danet.an.workflow.omgcore.WfAuditHandler ha
    throws IOException;

}
```

Inheritance Path. Section A.2.21, “Interface EventSubscriber” [180]

A.2.21.2. receive()

```
  public de.danet.an.workflow.omgcore.WfAuditEvent receive()
    throws IOException;
```

Parameters

return	the audit event

Exceptions

`IOException`	if an error occurs

Receives the next audit event. This method blocks until the next event is received.

A.2.21.3. receive(long)

```
  public de.danet.an.workflow.omgcore.WfAuditEvent receive(long timeout)
    throws IOException;
```

Parameters

`timeout`	the timeout value (in milliseconds). If 0, the method blocks until a message is received.
return	the audit event or `null` if the timeout expires

Exceptions

`IOException`	if an error occurs

Receives the next audit event that arrives within the specified timeout interval.

A.2.21.4. receiveNoWait()

```
public de.danet.an.workflow.omgcore.WfAuditEvent receiveNoWait()
  throws IOException;
```

Parameters

return	the audit event or `null` if no event is available

Exceptions

`IOException`	if an error occurs

Receives the next audit event if one is immediately available.

A.2.21.5. setEventHandler(WfAuditHandler)

```
public void setEventHandler(de.danet.an.workflow.omgcore.WfAuditHandler handler)
  throws IOException;
```

Parameters

`handler`	the event handler

Exceptions

`IOException`	if an error occurs

Sets a handler for received events that is automatically invoked. If a handler has been set, the `re-ceive` methods may not be called.

A.2.22. Interface ExecutionObject

Interface `ExecutionObject` extends the `OMG execution object` with additional methods that allow the type-safe `state class` to be used to query and set state. The OMG API provides string based methods only to enable vendors to define additional sub-states. The type-safe equival-ent is to define new subclasses of the state classes.

A.2.22.1. Synopsis

```
 public interface de.danet.an.workflow.api.ExecutionObject extends, de.danet.an.wo
// Public Methods

  public void changeState(de.danet.an.workflow.omgcore.WfExecutionObject.State newS
    throws RemoteException, InvalidStateException, TransitionNotAllowedException;


  public boolean debugEnabled()
    throws RemoteException;
  public de.danet.an.workflow.omgcore.WfExecutionObject.State typedState()
    throws RemoteException;

}
```

Inheritance Path. Section A.2.22, “Interface ExecutionObject” [182]

A.2.22.2. changeState(WfExecutionObject.State)

```
  public void changeState(de.danet.an.workflow.omgcore.WfExecutionObject.Sta
    throws RemoteException, InvalidStateException, TransitionNotAllowedExcep
```

Parameters

`newState`	state to change to.

Exceptions

`InvalidStateException`	if `newState` is an invalid state for the execution object.
`TransitionNotAllowedException`	if the transition from the current state to `newState` is not allowed.
`RemoteException`	if a system-level error occurs.

Type-safe equivalent to `WfExecutionObject.changeState()` .

A.2.22.3. debugEnabled()

```
  public boolean debugEnabled()
    throws RemoteException;
```

Parameters

return	`true` if the execution object is in debugging mode

Exceptions

`RemoteException`	if a system-level error occurs

Checks if the execution object is in debugging mode.

A.2.22.4. typedState()

```
  public de.danet.an.workflow.omgcore.WfExecutionObject.State typedState()
    throws RemoteException;
```

Parameters

return	the state.

Exceptions

`RemoteException`	if a system-level error occurs.

Type-safe equivalent to `WfExecutionObject.state()` .

A.2.23. Interface ExternalReference

This interface represents an `ExternalReference` as specified by XPDL.

A.2.23.1. Synopsis

```
 public interface de.danet.an.workflow.api.ExternalReference extends, java.io.Seri
// Public Methods

  public java.net.URI location();


  public String namespace();


  public String xref();

}
```

Since	1.1

Inheritance Path. Section A.2.23, “Interface ExternalReference” [184]

A.2.23.2. location()

```
  public java.net.URI location();
```

Parameters

return	location information

Returns the location information of the external document.

A.2.23.3. namespace()

```
  public String namespace();
```

Parameters

return	namespace information

Returns the scope in which the entity is defined or `null` if no such information is supplied.

A.2.23.4. xref()

```
  public String xref();
```

Parameters

return	identity of entity

Returns the identity of the entity within the external document or `null` if no such information is supplied.

A.2.24. Error FactoryConfigurationError

This exception is thrown by the `newInstance` method of `WorkflowServiceFactory` .

A.2.24.1. Synopsis

```
 public class de.danet.an.workflow.api.FactoryConfigurationError extends, ja
// Public Constructors

  public FactoryConfigurationError();


  public FactoryConfigurationError(String msg);


  public FactoryConfigurationError(String msg,
                                   Throwable cause);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Error-> Section A.2.24, “Error FactoryConfigurationError” [185]

A.2.24.2. FactoryConfigurationError()

```
  public FactoryConfigurationError();
```

Creates a new exception.

A.2.24.3. FactoryConfigurationError(String)

```
  public FactoryConfigurationError(String msg);
```

Parameters

`msg`	the detail message.

Creates a new exception with the given message.

A.2.24.4. FactoryConfigurationError(String, Throwable)

```
  public FactoryConfigurationError(String msg,
                                   Throwable cause);
```

Parameters

`msg`	the detail message.
`cause`	the cause.

Creates a new exception with the given message and cause.

A.2.25. Class FormalParameter

This class provides a description of a formal parameter as used for workflow processes and workflow applications.

A.2.25.1. Synopsis

```
 public class de.danet.an.workflow.api.FormalParameterimplements, java.io.Serializ
// Public Constructors

  public FormalParameter(String newId,
                         String newIndex,
                         FormalParameter.Mode newMode,
                         Object newType);

// Public Methods

  public boolean equals(Object obj);


  public int hashCode();


  public String id();


  public String index();


  public FormalParameter.Mode mode();


  public String toString();


  public Object type();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.25, “Class FormalParameter” [186]

A.2.25.2. FormalParameter(String, String, FormalParameter.Mode, Object)

```
public FormalParameter(String newId,
                       String newIndex,
                       FormalParameter.Mode newMode,
                       Object newType);
```

Parameters

`newId`	identifier of the formal parameter in String
`newIndex`	index of the formal parameter in String. In the 1.iteration if the new Index is not null, a warning will be generated.
`newMode`	mode of this formal parameter
`newType`	type of this formal parameter

Creates a new `FormalParameter` .

A.2.25.3. equals(Object)

```
public boolean equals(Object obj);
```

Parameters

`obj`	the object to compare with.
return	`true` if the objects are equal.

Compare two formal parameter objects.

A.2.25.4. hashCode()

```
public int hashCode();
```

Parameters

return	the hash code.

Evaluate hash code.

A.2.25.5. id()

```
public String id();
```

Parameters

return	a String representing the id value

Return the id of the formal parameter.

A.2.25.6. index()

```
public String index();
```

Parameters

return	a String representing the index value

Return the index of the formal parameter.

A.2.25.7. mode()

```
public FormalParameter.Mode mode();
```

Parameters

return	the `FormalParameter.Mode` of the formal parameter.

Return the mode of the formal parameter.

A.2.25.8. toString()

```
public String toString();
```

Parameters

return	the result.

Create string representation for debugging purposes.

A.2.25.9. type()

```
public Object type();
```

Parameters

return	the type of the formal parameter.

Return the type of the formal parameter. Types are represented as defined for `Process-Mgr.contextSignature` .

A.2.26. Class FormalParameter.Mode

Defines a class for representing priorities in a type save way.

A.2.26.1. Synopsis

```
 public static final class de.danet.an.workflow.api.FormalParameter.Modeimpl
// Public Static Fields

  public static final FormalParameter.Mode IN ;


  public static final FormalParameter.Mode INOUT ;


  public static final FormalParameter.Mode OUT ;

// Public Static Methods

  public static FormalParameter.Mode fromString(String text)
    throws IllegalArgumentException;

// Public Methods

  public final String toString();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.26, “Class FormalParameter.Mode” [188]

A.2.26.2. IN

```
  public static final FormalParameter.Mode IN ;
```

Mode as Input Parameter.

A.2.26.3. INOUT

```
  public static final FormalParameter.Mode INOUT ;
```

Mode as input and output parameter.

A.2.26.4. OUT

```
  public static final FormalParameter.Mode OUT ;
```

Mode as Output Parameter.

A.2.26.5. fromString(String)

```
  public static FormalParameter.Mode fromString(String text)
    throws IllegalArgumentException;
```

Parameters

`text`	mode name to search

return	mode object

Exceptions

`IllegalArgumentException`	if `text` is not a valid mode name.

Get a Mode by name.

A.2.26.6. toString()

```
  public final String toString();
```

Parameters

return	mode as text

Returns the mode as text.

A.2.27. Interface GroupResource

This interface extends the interface `WfResource` for resources that are groups. The distinction of resource types is without relevance to the workflow engine, but may be used to e.g. display resource types differently in the user interface. Resource management services are not required to make use of this interface, they may deliver all resources as plain `WfResource` s. Application must therefore be prepared to handle resources that only implement the base interface `WfResource` .

A.2.27.1. Synopsis

```
 public interface de.danet.an.workflow.api.GroupResource extends, de.danet.an.work
}
```

See Also	Section A.2.50, “Interface UserResource” [240] , Section A.2.47, “Interface RoleResource” [236]

Inheritance Path. Section A.2.27, “Interface GroupResource” [190]

A.2.28. Exception ImportException

This exception is thrown by the `importProcessDefinitions` method of `ProcessDefinitionDirectory` and results from collected errors.

A.2.28.1. Synopsis

```
 public class de.danet.an.workflow.api.ImportException extends, java.lang.Exceptio
    implements, java.io.Serializable {
// Public Constructors

  public ImportException(String msg,
                         java.util.List prioritizedMessages);
// Public Methods

  public java.util.List messages();

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.2.28, “Exception ImportException” [190]

A.2.28.2. ImportException(String, List)

```
  public ImportException(String msg,
                         java.util.List prioritizedMessages);
```

Parameters

`msg`	the main message of the exception.
`prioritizedMessages`	the given prioritized messages.

Construct a `ImportException` from a collecting error handler. The warnings, errors, fatal error of the Exception are taken from the error handler.

A.2.28.3. messages()

```
  public java.util.List messages();
```

Parameters

return	list with error messages (may be empty).

Return all messages, i.e. warnings, errors and fatal errors. The value returned is a list of `PrioritizedMessage` .

A.2.29. Exception InvalidIdException

This exception is raised if an invalid id is passed to a lookup method.

A.2.29.1. Synopsis

```
 public class de.danet.an.workflow.api.InvalidIdException extends, java.lang
    implements, java.io.Serializable {
// Public Constructors

  public InvalidIdException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.2.29, “Exception InvalidIdException” [191]

A.2.29.2. InvalidIdException(String)

```
  public InvalidIdException(String msg);
```

Parameters

`msg`	the message

Constructs a new exception with the given message. The message should at least include the invalid id.

A.2.30. Exception InvalidKeyException

This exception is raised if an invalid key is passed to a lookup method.

A.2.30.1. Synopsis

```
 public class de.danet.an.workflow.api.InvalidKeyException extends, java.lang.Exce
    implements, java.io.Serializable {
// Public Constructors

  public InvalidKeyException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.2.30, “Exception InvalidKeyException” [192]

A.2.30.2. InvalidKeyException(String)

```
  public InvalidKeyException(String msg);
```

Parameters

`msg`	the message

Constructs a new exception with the given message. The message should at least include the invalid key.

A.2.31. Class MethodInvocationBatch

This class provides a `Batch` implementation that executes several invocations of remote objects on the server in a single transaction and returns the results. This class can be thought of as a "generic DTO".

A.2.31.1. Synopsis

```
 public class de.danet.an.workflow.api.MethodInvocationBatchimplements, de.d
// Public Constructors

  public MethodInvocationBatch();


  public MethodInvocationBatch(boolean stopOnException);

// Public Methods

  public void addInvocation(int result,
                            String method,
                            String[] argTypes,
                            Object[] args,
                            boolean discard);


  public void addInvocation(Object obj,
                            String method,
                            String[] argTypes,
                            Object[] args);


  public Object execute(Batch.Context ctx);


  public String toString();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.31, “Class MethodInvocationBatch” [193]

A.2.31.2. MethodInvocationBatch()

```
  public MethodInvocationBatch();
```

Create a new empty method invocation batch. Equivalent to `MethodInvocation-Batch(false)` .

A.2.31.3. MethodInvocationBatch(boolean)

```
  public MethodInvocationBatch(boolean stopOnException);
```

Parameters

`stopOnException`	if `true` execution is interrupted on the first encountered ex-

ception

Create a new empty method invocation batch. The flag passed as parameter controls if an exception stops the batch execution.

A.2.31.4. addInvocation(int, String, String[], Object[], boolean)

```
public void addInvocation(int result,
                          String method,
                          String[] argTypes,
                          Object[] args,
                          boolean discard);
```

Parameters

`result`	**relative** index of the result to be used for method invovation, i.e. `-1` is the previous result.
`method`	the method name.
`argTypes`	the argument types as strings suitable for `ClassLoader.loadClass` . May be `null` which is interpreted as "no parameters".
`args`	the actual arguments. May be `null` which is interpreted as "no parameters".
`discard`	if `true` the referenced result will be removed from the result list.

Adds a method invocation on a previous result to the batch.

A.2.31.5. addInvocation(Object, String, String[], Object[])

```
public void addInvocation(Object obj,
                          String method,
                          String[] argTypes,
                          Object[] args);
```

Parameters

`obj`	the objects whose method is to be invoked.
`method`	the method name.
`argTypes`	the argument types as strings suitable for `ClassLoader.loadClass` . May be `null` which is interpreted as "no parameters".
`args`	the actual arguments. May be `null` which is interpreted as "no parameters".

Adds a method invocation to the batch.

A.2.31.6. execute(Batch.Context)

```
  public Object execute(Batch.Context ctx);
```

Specified by: Method execute in interface Batch

Parameters

`ctx`	the execution context.
return	the execution result, which is of type `Result` .

Executes the registered method invocations one by one in a single transaction. Note that execution is terminated if an invoked method sets rollback only.

A.2.32. Class MethodInvocationBatch.Result

The result of an execution of this kind of batch.

A.2.32.1. Synopsis

```
 public class de.danet.an.workflow.api.MethodInvocationBatch.Resultimplement
// Public Constructors

  public MethodInvocationBatch.Result(Object[] theResults,
                                      boolean exceptionsOccured);

// Public Methods

  public Exception firstException()
    throws IllegalStateException;


  public boolean hasExceptions();


  public Object result(int i);


  public java.util.Date resultAsDate(int i);


  public int resultAsInt(int i);


  public String resultAsString(int i);


  public Object[] results();


  public String toString();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.32, “Class MethodInvocationBatch.Result” [195]

A.2.32.2. MethodInvocationBatch.Result(Object[], boolean)

```
  public MethodInvocationBatch.Result(Object[] theResults,
                                      boolean exceptionsOccured);
```

Parameters

`theResults`	the results attribute.
`exceptionsOccured`	the exceptions attribute.

Construct a new `Result` objects with the given attributes.

A.2.32.3. firstException()

```
  public Exception firstException()
    throws IllegalStateException;
```

Parameters

return	the first exception.

Exceptions

`IllegalStateException`	if the result includes no exceptions.

Return the first exception in the result list.

A.2.32.4. hasExceptions()

```
  public boolean hasExceptions();
```

Parameters

return	`true` if execeptions have occured.

Return `true` if any exceptions have occured during batch execution.

A.2.32.5. result(int)

```
  public Object result(int i);
```

Parameters

`i`	the index into the result array.
return	result.

Returns the result with the given index.

A.2.32.6. resultAsDate(int)

```
public java.util.Date resultAsDate(int i);
```

Parameters

`i`	the index into the result array.
return	result as `Date` .

Returns the result with the given index as `Date` .

A.2.32.7. resultAsInt(int)

```
public int resultAsInt(int i);
```

Parameters

`i`	the index into the result array.
return	result as `int` .

Returns the result with the given index as `int` .

A.2.32.8. resultAsString(int)

```
public String resultAsString(int i);
```

Parameters

`i`	the index into the result array.
return	result as `String` .

Returns the result with the given index as `String` .

A.2.32.9. results()

```
public Object[] results();
```

Parameters

return	the results.

The results as an array of objects. Each entry in the array corresponds to a method call and is either a result (with java primitive types wrapped in corresponding objects) or an exception thrown by the invoked method.

A.2.33. Exception NoSuchResourceException

This exception is thrown if a resource has become invalid.

A.2.33.1. Synopsis

```
 public class de.danet.an.workflow.api.NoSuchResourceException extends, java.lang.
    implements, java.io.Serializable {
// Public Constructors

  public NoSuchResourceException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.2.33, “Exception NoSuchResourceException” [198]

A.2.33.2. NoSuchResourceException(String)

```
  public NoSuchResourceException(String msg);
```

Parameters

`msg`	the detail message.

Create a new exception with the given detail message.

A.2.34. Interface Participant

This interface identifies the data type "workflow participant" in a `ProcessDataInfo` object.

A.2.34.1. Synopsis

```
 public interface de.danet.an.workflow.api.Participant {
// Public Methods

  public String getId();


  public String getName();


  public Participant.ParticipantType getParticipantType();


  public Object getResourceSelection();

}
```

Inheritance Path. Section A.2.34, “Interface Participant” [198]

A.2.34.2. getId()

```
public String getId();
```

Parameters

return	a String representing the id value

Get the id of the participant.

A.2.34.3. getName()

```
public String getName();
```

Parameters

return	a String representing the name value

Get the name of the participant.

A.2.34.4. getParticipantType()

```
public Participant.ParticipantType getParticipantType();
```

Parameters

return	a ParticipantType object representing the participant type value

Get the type of the participant.

A.2.34.5. getResourceSelection()

```
public Object getResourceSelection();
```

Parameters

return	an Object representing the resource selection value

Get the additional resource selection information passed to the constructor.

A.2.35. Class Participant.ParticipantType

This class defines the participant type for a participant.

A.2.35.1. Synopsis

```
 public static final class de.danet.an.workflow.api.Participant.ParticipantTypeimp
// Public Static Fields

  public static final Participant.ParticipantType HUMAN ;


  public static final Participant.ParticipantType ORGANIZATIONAL_UNIT ;


  public static final Participant.ParticipantType RESOURCE ;


  public static final Participant.ParticipantType RESOURCE_SET ;


  public static final Participant.ParticipantType ROLE ;


  public static final Participant.ParticipantType SYSTEM ;

// Public Static Methods

  public static Participant.ParticipantType fromString(String text)
    throws IllegalArgumentException;

// Public Methods

  public final boolean isHuman();


  public final boolean isOrganizationUnit();


  public final boolean isResource();


  public final boolean isResourceSet();


  public final boolean isRole();


  public final boolean isSystem();


  public final String toString();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.35, “Class Participant.ParticipantType” [199]

A.2.35.2. HUMAN

```
  public static final Participant.ParticipantType HUMAN ;
```

HUMAN ParticipantType.

A.2.35.3. ORGANIZATIONAL_UNIT

```
public static final Participant.ParticipantType ORGANIZATIONAL_UNIT ;
```

ORGANIZATIONAL_UNIT ParticipantType.

A.2.35.4. RESOURCE

```
public static final Participant.ParticipantType RESOURCE ;
```

RESOURCE ParticipantType.

A.2.35.5. RESOURCE_SET

```
public static final Participant.ParticipantType RESOURCE_SET ;
```

RESOURCE_SET ParticipantType.

A.2.35.6. ROLE

```
public static final Participant.ParticipantType ROLE ;
```

ROLE ParticipantType.

A.2.35.7. SYSTEM

```
public static final Participant.ParticipantType SYSTEM ;
```

SYSTEM ParticipantType.

A.2.35.8. fromString(String)

```
public static Participant.ParticipantType fromString(String text)
  throws IllegalArgumentException;
```

Parameters

`text`	participant type name to search
return	participant type object

Exceptions

`IllegalArgumentException`	if `text` is not a valid participant type name.

Get a participant type by name.

A.2.35.9. isHuman()

```
public final boolean isHuman();
```

Parameters

return	`true` if the type is "HUMAN".

Checks if the type is "HUMAN".

A.2.35.10. isOrganizationUnit()

```
public final boolean isOrganizationUnit();
```

Parameters

return	`true` if the type is "ORGANIZATIONAL_UNIT".

Checks if the type is "ORGANIZATIONAL_UNIT".

A.2.35.11. isResource()

```
public final boolean isResource();
```

Parameters

return	`true` if the type is "RESOURCE".

Checks if the type is "RESOURCE".

A.2.35.12. isResourceSet()

```
public final boolean isResourceSet();
```

Parameters

return	`true` if the type is "RESOURCE_SET".

Checks if the type is "RESOURCE_SET".

A.2.35.13. isRole()

```
public final boolean isRole();
```

Parameters

return	`true` if the type is "ROLE".

Checks if the type is "ROLE".

A.2.35.14. isSystem()

```
  public final boolean isSystem();
```

Parameters

return	`true` if the type is "SYSTEM".

Checks if the type is "SYSTEM".

A.2.35.15. toString()

```
  public final String toString();
```

Parameters

return	type as text

Returns the type as text.

A.2.36. Class PrioritizedMessage

This class presents a prioritized message that will be internationalized using the specified resource bundle and the referenced entry. For details see the description of its class contructor.

A.2.36.1. Synopsis

```
 public class de.danet.an.workflow.api.PrioritizedMessageimplements, java.io
// Public Constructors

  public PrioritizedMessage(PrioritizedMessage.Priority priority,
                            String message);


  public PrioritizedMessage(PrioritizedMessage.Priority priority,
                            String message,
                            Object[] data);

// Public Methods

  public String message();


  public String message(java.util.Locale locale);


  public PrioritizedMessage.Priority priority();


  public String toString();


  public String unmappedMessage();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.36, “Class PrioritizedMessage” [203]

A.2.36.2. PrioritizedMessage(PrioritizedMessage.Priority, String)

```
public PrioritizedMessage(PrioritizedMessage.Priority priority,
                          String message);
```

Parameters

`priority`	the priority of the given message.
`message`	a message or a resource and key reference.

See Also	`message() [204]`

Constructs a prioritized message. If the message has the format " `a.resource.bundle.base.name#key` " it is interpreted as the base name of a <code>ReosurceBundle</code> and the key of an entry in this resource bundle.

A.2.36.3. PrioritizedMessage(PrioritizedMessage.Priority, String, Object[])

```
public PrioritizedMessage(PrioritizedMessage.Priority priority,
                          String message,
                          Object[] data);
```

Parameters

`priority`	the priority of the given message.
`message`	a message or a resource and key reference.
`data`	additional data used when formatting the message.

See Also	`message() [204]`

Constructs a prioritized message. If the message has the format " `a.resource.bundle.base.name#key` " it is interpreted as the base name of a <code>ReosurceBundle</code> and the key of an entry in this resource bundle.

If the parameter `data` is not `null` , the message (or the string looked up in the resource bundle) will be fomatted using <code>MessageFormat.format</code>.

A.2.36.4. message()

```
public String message();
```

Parameters

return	the message.

See Also	`message(java.util.Locale) [205]`

Returns the message. If the message has the format " `a.resource.bundle.base.name#key` ", it will be internationalized using the specified resource bundle and the referenced entry with the default <code>Locale</code>.

A.2.36.5. message(Locale)

```
public String message(java.util.Locale locale);
```

Parameters

`locale`	the `Locale` to be used for resource bundle lookup.
return	the message.

Returns the message. If the message has the format " `a.resource.bundle.base.name#key` ", it will be internationalized using the specified resource bundle and the referenced entry using the given <code>Locale</code>.

A.2.36.6. priority()

```
public PrioritizedMessage.Priority priority();
```

Parameters

return	the priority of the message.

Returns the priority of the message.

A.2.36.7. toString()

```
public String toString();
```

Parameters

return	a string representation.

Returns a string representation of the message.

A.2.36.8. unmappedMessage()

```
public String unmappedMessage();
```

Parameters

return	the message.

Returns the message that it is not internationalized.

A.2.37. Class PrioritizedMessage.Priority

This class represents the priority of a given message. It was taken over from org.apache.log4j.Priority to avoid the dependence of the log4j library.

A.2.37.1. Synopsis

```
 public static class de.danet.an.workflow.api.PrioritizedMessage.Priorityimplement
// Public Static Fields

  public static final PrioritizedMessage.Priority DEBUG ;


  public static final PrioritizedMessage.Priority ERROR ;


  public static final PrioritizedMessage.Priority FATAL ;


  public static final PrioritizedMessage.Priority INFO ;


  public static final PrioritizedMessage.Priority WARN ;

// Protected Constructors

  protected PrioritizedMessage.Priority(int level,
                                        String levelStr);

// Public Methods

  public int compareTo(Object other);


  public String toString();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.37, “Class PrioritizedMessage.Priority” [206]

A.2.37.2. PrioritizedMessage.Priority(int, String)

```
  protected PrioritizedMessage.Priority(int level,
                                        String levelStr);
```

Parameters

`level`	the level of the priority.
`levelStr`	the string representation of the priority.

Constructor of the priority of a message.

A.2.37.3. DEBUG

```
public static final PrioritizedMessage.Priority DEBUG ;
```

The `DEBUG` priority designates fine-grained informational events that are most useful to debug an application.

A.2.37.4. ERROR

```
public static final PrioritizedMessage.Priority ERROR ;
```

The `ERROR` level designates error events that might still allow the application to continue running.

A.2.37.5. FATAL

```
public static final PrioritizedMessage.Priority FATAL ;
```

The `FATAL` level designates very severe error events that will presumably lead the application to abort.

A.2.37.6. INFO

```
public static final PrioritizedMessage.Priority INFO ;
```

The `INFO` level designates informational messages that highlight the progress of the application at coarse-grained level.

A.2.37.7. WARN

```
public static final PrioritizedMessage.Priority WARN ;
```

The `WARN` level designates potentially harmful situations.

A.2.37.8. compareTo(Object)

```
public int compareTo(Object other);
```

Specified by: Method `compareTo` in interface `Comparable`

Parameters

`other`	priority to compare with.
return	a negative integer, zero, or a positive integer as this priority is less than, equal to, or greater than the given priority.

Implements `Comparable` .

A.2.37.9. toString()

```
public String toString();
```

Parameters

return	this priority in string.

Returns the string representation of this priority.

A.2.38. Interface Process

Interface `Process` adds some functions to the `OMG process` .

A.2.38.1. Synopsis

```
 public interface de.danet.an.workflow.api.Process extends, de.danet.an.workflow.a
// Public Methods

  public Activity activityByKey(String key)
    throws RemoteException, InvalidKeyException;


  public java.util.Date createTime()
    throws RemoteException;


  public ProcessDefinition processDefinition()
    throws RemoteException;


  public void setDebugEnabled(boolean debug)
    throws RemoteException, InvalidStateException;


  public java.util.List transitions()
    throws RemoteException;

}
```

Inheritance Path. Section A.2.38, “Interface Process” [208]

A.2.38.2. activityByKey(String)

```
  public Activity activityByKey(String key)
    throws RemoteException, InvalidKeyException;
```

Parameters

`key`	the `key` of the activity
return	the activity associated with the key

Exceptions

`InvalidKeyException`	if no activity with the given key exists
`RemoteException`	if a system-level error occurs.

Returns the `Activity` with the given key. The OMG interface only defines a `method for listing all the activities` associated with the process. While, of course, one could select the activity with a certain key from that list, this would be rather insufficient.

A.2.38.3. createTime()

```
  public java.util.Date createTime()
    throws RemoteException;
```

Parameters

return	the creation time.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the creation time of the process.

A.2.38.4. processDefinition()

```
  public ProcessDefinition processDefinition()
    throws RemoteException;
```

Parameters

return	the representation of the process definition.

Exceptions

`RemoteException`	if a system-level error occurs.

Returns the process definition of this process.

A.2.38.5. setDebugEnabled(boolean)

```
  public void setDebugEnabled(boolean debug)
    throws RemoteException, InvalidStateException;
```

Parameters

`debug`	if the process is to be debugged

Exceptions

`RemoteException`	if a system-level error occurs.
`InvalidStateException`	if changing the debug mode is not allowed

Enable or disable debugging of the process. Changing the debug mode is only allowed when the process has been started.

A.2.38.6. transitions()

```
public java.util.List transitions()
  throws RemoteException;
```

Parameters

return	list of transitions for this process

Exceptions

`RemoteException`	if a system level error occurs.

Gets a list of transitions for this process.

A.2.39. Interface ProcessClosedAuditEvent

A `ProcessClosedAuditEvent` extends the `WfStateAuditEvent` . It is reported when a process changes its state to `closed` . In addition to the information provided by the base `WfStateAuditEvent` , this event provides the process' result. This makes the result available to listeners waiting for the end of a process even in cases when a process is automatically removed after completion.

A.2.39.1. Synopsis

```
 public interface de.danet.an.workflow.api.ProcessClosedAuditEvent extends, de.dan
// Public Methods

  public de.danet.an.workflow.omgcore.ProcessData result();

}
```

See Also	`result() [133]`

Inheritance Path. Section A.2.39, “Interface ProcessClosedAuditEvent” [210]

A.2.39.2. result()

```
public de.danet.an.workflow.omgcore.ProcessData result();
```

Parameters

return	Returns the result.

A.2.40. Interface ProcessDefinition

This interface defines a process definiton.

A.2.40.1. Synopsis

```
 public interface de.danet.an.workflow.api.ProcessDefinition {
// Public Static Fields

  public static final int AUDIT_SELECTION_ALL_EVENTS  = 0;


  public static final int AUDIT_SELECTION_NO_EVENTS  = 3;


  public static final int AUDIT_SELECTION_PROCESS_CLOSED_EVENTS_ONLY  = 2;


  public static final int AUDIT_SELECTION_STATE_EVENTS_ONLY  = 1;


  public static final int REMOVE_AUTOMATIC  = 1;


  public static final int REMOVE_COMPLETED  = 2;


  public static final int REMOVE_MANUAL  = 0;

// Public Methods

  public Application applicationById(String id)
    throws InvalidIdException;


  public java.util.Collection applications();


  public int auditEventSelection();


  public int cleanupMode();


  public de.danet.an.workflow.omgcore.ProcessDataInfo contextSignature();


  public FormalParameter[] formalParameters();


  public String mgrName();


  public String packageId();


  public String packageName();


  public Participant participantById(String id)
    throws InvalidIdException;


  public java.util.Collection participants();
  public ProcessDefinition.ProcessHeaderData processHeader();

  public String processId();

  public String processName();

  public boolean removeClosedProcess();

  public de.danet.an.workflow.omgcore.ProcessDataInfo resultSignature();

  public boolean storeAuditEvents();

  public org.jdom.Document toJDOM();

  public SAXEventBuffer toSAX();

  public String toXPDL();

  public String version();
}
```

Inheritance Path. Section A.2.40, “Interface ProcessDefinition” [211]

A.2.40.2. AUDIT_SELECTION_ALL_EVENTS

```
  public static final int AUDIT_SELECTION_ALL_EVENTS  = 0;
```

Select all audit events to be delivered and recorded.

A.2.40.3. AUDIT_SELECTION_NO_EVENTS

```
  public static final int AUDIT_SELECTION_NO_EVENTS  = 3;
```

Select no audit events to be delivered and recorded.

A.2.40.4. AUDIT_SELECTION_PROCESS_CLOSED_EVENTS_ONLY

```
  public static final int AUDIT_SELECTION_PROCESS_CLOSED_EVENTS_ONLY  = 2;
```

Select process closed events for delivery and recording only.

A.2.40.5. AUDIT_SELECTION_STATE_EVENTS_ONLY

```
  public static final int AUDIT_SELECTION_STATE_EVENTS_ONLY  = 1;
```

Select state change events for delivery and recording only.

A.2.40.6. REMOVE_AUTOMATIC

```
  public static final int REMOVE_AUTOMATIC  = 1;
```

Remove closed processes automatically.

A.2.40.7. REMOVE_COMPLETED

```
  public static final int REMOVE_COMPLETED  = 2;
```

Remove closed processes automatically if `closed.completed` .

A.2.40.8. REMOVE_MANUAL

```
  public static final int REMOVE_MANUAL  = 0;
```

Remove closed processes manually.

A.2.40.9. applicationById(String)

```
  public Application applicationById(String id)
    throws InvalidIdException;
```

Parameters

`id`	the application id.
return	the application.

Exceptions

`InvalidIdException`	if no application with the given id exists.

Return the application with the given id.

A.2.40.10. applications()

```
  public java.util.Collection applications();
```

Parameters

return	a collection of `Applications` .

Returns the applications defined in this process.

A.2.40.11. auditEventSelection()

```
  public int auditEventSelection();
```

Parameters

return	the filter

This method returns the selected audit events of instances of this process definition.

A.2.40.12. cleanupMode()

```
public int cleanupMode();
```

Parameters

return	the cleanup mode.

This method checks if a closed process should be removed. Parse the process definition and find out if the extendAttribute with the name of RemoveClosedProcess has the value of `MANUAL` , `AUTOMATIC` or `COMPLETED` and return the corresponding constant. Default is to remove automatically, i.e. `REMOVE_AUTOMATIC` .

A.2.40.13. contextSignature()

```
public de.danet.an.workflow.omgcore.ProcessDataInfo contextSignature();
```

Parameters

return	the process meta information.
See Also	`contextSignature() [136]`

Returns the meta information that defines how to set the context for this kind of process. Equivalent to calling `contextSignature` on a process manager for this kind of process.

A.2.40.14. formalParameters()

```
public FormalParameter[] formalParameters();
```

Parameters

return	the process meta information. The result is never `null` , rather an array with zero element is returned.

Returns the meta information that describes the formal parameters for this kind of process.

A.2.40.15. mgrName()

```
public String mgrName();
```

Parameters

return	process manager name.

The name of the associated `process manager` . While the name of a process in XPDL is just a human readable "label", the name attribute of the process manager must be unique within agiven business domain.

The standard implementation of `ProcessDefinition` derives a manager name from the package and process ids in the XPDL, separated by a slash ("/").

A.2.40.16. packageId()

```
public String packageId();
```

Parameters

return	package id.

Id of the package as specified in the XPDL description.

A.2.40.17. packageName()

```
public String packageName();
```

Parameters

return	package name.

Name of the package as specified in the XPDL description.

A.2.40.18. participantById(String)

```
public Participant participantById(String id)
  throws InvalidIdException;
```

Parameters

`id`	identity of the participant in string
return	a Participant object

Exceptions

`InvalidIdException`	if no participant with the given id exists.

Return the participant identified by the id.

A.2.40.19. participants()

```
public java.util.Collection participants();
```

Parameters

return	a collection of `Participants` for this process.

Gets the participants for this process.

A.2.40.20. processHeader()

```
public ProcessDefinition.ProcessHeaderData processHeader();
```

Parameters

return	process header data object of the process description

Returns process header data object of the process description.

A.2.40.21. processId()

```
public String processId();
```

Parameters

return	process id.

Id of the process as specified in the XPDL description.

A.2.40.22. processName()

```
public String processName();
```

Parameters

return	process name.

Name of the process as specified in the XPDL description.

A.2.40.23. removeClosedProcess()

```
public boolean removeClosedProcess();
```

Parameters

return	`true` if the closed process should be removed, otherwise

false.

Deprecated

Use `cleanupMode` . For backward compatibility, this method returns `true` if `cleanupMode` returns `REMOVE_AUTOMATIC` .

This method checks if the closed process should be removed. Parse the process definition and find out if the extendAttribute with the name of RemoveClosedProcess has the value of MANUAL, then return false; if it has the value of AUTOMATIC, then return true. Default is true.

A.2.40.24. resultSignature()

```
public de.danet.an.workflow.omgcore.ProcessDataInfo resultSignature();
```

Parameters

return	the process meta information.
See Also	`resultSignature() [139]`

Returns the meta information that describes the result for this kind of process. Equivalent to calling `resultSignature` on a process manager for this kind of process.

The implementation returns all formal IN or INOUT parameters.

A.2.40.25. storeAuditEvents()

```
public boolean storeAuditEvents();
```

Parameters

return	`true` if only state change events of the process instance are audited.

This method reports if audit events are written to the database.

A.2.40.26. toJDOM()

```
public org.jdom.Document toJDOM();
```

Parameters

return	DOM representation of the process definition

Returns the process description as XPDL JDOM tree.

A.2.40.27. toSAX()

```
  public SAXEventBuffer toSAX();
```

Parameters

return	the process definition
Since	1.2

Return the process definition as SAX event buffer.

A.2.40.28. toXPDL()

```
  public String toXPDL();
```

Parameters

return	the process definition

Returns the process description as XPDL textual description.

A.2.40.29. version()

```
  public String version();
```

Parameters

return	process defintion version.

Version of the process definition as specified in the XPDL description.

A.2.41. Interface ProcessDefinition.PackageHeaderData

An interface providing the information from the package definition header section.

A.2.41.1. Synopsis

```
 public static interface de.danet.an.workflow.api.ProcessDefinition.PackageHeaderD
// Public Methods

  public String costUnit();

  public String created();

  public String description();
  public String documentation();

  public String priorityUnit();

  public String vendor();

  public String xpdlVersion();
}
```

Inheritance Path. Section A.2.41, “Interface ProcessDefinition.PackageHeaderData” [218]

A.2.41.2. costUnit()

```
  public String costUnit();
```

Parameters

return	the cost unit.

Units used in simulation data.

A.2.41.3. created()

```
  public String created();
```

Parameters

return	the created time.

Creation date of package definition.

A.2.41.4. description()

```
  public String description();
```

Parameters

return	the description.

Short description of the package definition.

A.2.41.5. documentation()

```
  public String documentation();
```

Parameters

return	the documentation.

Operating system specific path and -filename of help file/description file.

A.2.41.6. priorityUnit()

```
public String priorityUnit();
```

Parameters

return	the priority unit.

Units used in simulation data.

A.2.41.7. vendor()

```
public String vendor();
```

Parameters

return	the vendor.

Defines the origin of this model definition and contains vendor's name, vendor's product name and product's release number.

A.2.41.8. xpdlVersion()

```
public String xpdlVersion();
```

Parameters

return	the version.

Version of the process definition specification.

A.2.42. Interface ProcessDefinition.ProcessHeaderData

An interface providing the information from the process definition header section.

A.2.42.1. Synopsis

```
 public static interface de.danet.an.workflow.api.ProcessDefinition.ProcessHeaderD
// Public Methods

  public String author();


  public String codepage();
  public String countrykey();

  public String created();

  public String description();

  public String limit();

  public ProcessDefinition.PackageHeaderData packageHeader();

  public String priority();

  public String publicationStatus();

  public java.util.List responsibles();

  public String timeEstimationDuration();

  public String timeEstimationWaiting();

  public String timeEstimationWorking();

  public String validFrom();

  public String validTo();

  public String version();
}
```

Inheritance Path. Section A.2.42, “Interface ProcessDefinition.ProcessHeaderData” [220]

A.2.42.2. author()

```
  public String author();
```

Parameters

return	the author.

Name of the author of this workflow process definition.

A.2.42.3. codepage()

```
  public String codepage();
```

Parameters

return	the codepage.

Describes the codepage used for the text parts.

A.2.42.4. countrykey()

```
public String countrykey();
```

Parameters

return	the country code.

Describes the country code based on ISO 3166. It could be either the three digits country code number, or the two alpha characters country codes.

A.2.42.5. created()

```
public String created();
```

Parameters

return	the created information.

Dreation date of the process definition.

A.2.42.6. description()

```
public String description();
```

Parameters

return	the description.

Short description of the process definition.

A.2.42.7. limit()

```
public String limit();
```

Parameters

return	the limit.

Expected duration for time management purposes in units of duration unit (duration unit does not present in this class).

A.2.42.8. packageHeader()

```
public ProcessDefinition.PackageHeaderData packageHeader();
```

Parameters

return	the package header.

The elements of package header.

A.2.42.9. priority()

```
public String priority();
```

Parameters

return	the priority.

Priority of the process type.

A.2.42.10. publicationStatus()

```
public String publicationStatus();
```

Parameters

return	the status.

Describes the status of the Workflow Process Definition.

A.2.42.11. responsibles()

```
public java.util.List responsibles();
```

Parameters

return	the list of responsible Workflow participant in String.

Describes the responsible Workflow participant(s). It is assumed that the responsible is the supervisor during the execution of the process.

A.2.42.12. timeEstimationDuration()

```
public String timeEstimationDuration();
```

Parameters

return	the estimated time.

Describes the amount of time the performer of the activity needs to perform the task.

A.2.42.13. timeEstimationWaiting()

```
public String timeEstimationWaiting();
```

Parameters

return	the estimated time.

Describes the amount of time the performer of the activity needs to perform the task.

A.2.42.14. timeEstimationWorking()

```
public String timeEstimationWorking();
```

Parameters

return	the estimated time.

Describes the amount of time the performer of the activity needs to perform the task.

A.2.42.15. validFrom()

```
public String validFrom();
```

Parameters

return	the valid from date.

Date that the workflow process definition is active from.

A.2.42.16. validTo()

```
public String validTo();
```

Parameters

return	the valid to date.

Date at witch the process definition becomes valid.

A.2.42.17. version()

```
  public String version();
```

Parameters

return	the version.

Describes the version of this workflow process definition.

A.2.43. Interface ProcessDefinitionDirectory

This interface defines a process definiton directory.

A.2.43.1. Synopsis

```
 public interface de.danet.an.workflow.api.ProcessDefinitionDirectory extend
// Public Methods

  public java.util.List importProcessDefinitions(byte[] processDefinitions)
    throws RemoteException, ImportException;


  public java.util.List importProcessDefinitions(String processDefinitions)
    throws RemoteException, ImportException;


  public boolean isEnabled(String packageId,
                           String processId)
    throws RemoteException, InvalidKeyException;


  public ProcessDefinition lookupProcessDefinition(String packageId,
                                                   String processId)
    throws InvalidKeyException, RemoteException;


  public boolean processDefinitionExists(String packageId,
                                         String processId)
    throws RemoteException;


  public java.util.Collection processDefinitions()
    throws RemoteException;


  public ProcessMgr processMgr(String packageId,
                               String processId)
    throws InvalidKeyException, RemoteException;


  public void removeProcessDefinition(String packageId,
                                      String processId)
    throws RemoteException, InvalidKeyException;


  public void setEnabled(String packageId,
                         String processId,
                         boolean enabled)
    throws RemoteException, InvalidKeyException;

}
```

Inheritance Path. Section A.2.43, “Interface ProcessDefinitionDirectory” [225]

A.2.43.2. importProcessDefinitions(byte[])

```
public java.util.List importProcessDefinitions(byte[] processDefinitions)
  throws RemoteException, ImportException;
```

Parameters

`processDefinitions`	byte array resulting from an InputStream that describes the process definitions.
return	list of prioritized message `PrioritizedMessage` . This list only includes messages of priority INFO or WARN. If any (fatal) error has occured, an `ImportException` will be thrown and the error message can be taken from there.

Exceptions

`RemoteException`	if a system-level error occurs. The import has been aborted.
`ImportException`	if the input is not a correct.

This operation method import new process definitions from an XPDL description. Note that importing an XPDL description automatically removes any existing process definitions that have the same package id as the imported package.

A.2.43.3. importProcessDefinitions(String)

```
public java.util.List importProcessDefinitions(String processDefinitions)
  throws RemoteException, ImportException;
```

Parameters

`processDefinitions`	document describing the process definitions.
return	list of prioritized message `PrioritizedMessage` . This list only includes messages of priority INFO or WARN. If any (fatal) error has occured, an `ImportException` will be thrown and the error message can be taken from there.

Exceptions

`RemoteException`	if a system-level error occurs. The import has been aborted.
`ImportException`	if the input is not a correct.

This operation method import new process definitions from an XPDL description.

Note that importing an XPDL description automatically removes any existing process definitions that have the same package id as the imported package.

A.2.43.4. isEnabled(String, String)

```
public boolean isEnabled(String packageId,
                         String processId)
  throws RemoteException, InvalidKeyException;
```

Parameters

`packageId`	Id attribute of the process package.
`processId`	Id attribute of the process.
return	if the process definition is enabled.

Exceptions

`InvalidKeyException`	if no process definition with the given ids exists.
`RemoteException`	if a system-level error occurs.

This operation method returns true if the process definition with the given ids is enabled.

A.2.43.5. lookupProcessDefinition(String, String)

```
public ProcessDefinition lookupProcessDefinition(String packageId,
                                                 String processId)
  throws InvalidKeyException, RemoteException;
```

Parameters

`packageId`	Id attribute of the process package.
`processId`	Id attribute of the process.
return	the found ProcessDefinition object.

Exceptions

`InvalidKeyException`	if no process definition with the given ids exists.
`RemoteException`	if a system-level error occurs.

This method delivers the process definition for the given ids. If no process definition with the ids exist, it throws an `IllegalArgumentException` .

A.2.43.6. processDefinitionExists(String, String)

```
public boolean processDefinitionExists(String packageId,
                                     String processId)
    throws RemoteException;
```

Parameters

`packageId`	Id attribute of the process package.
`processId`	Id attribute of the process.
return	`true` if a process definition with the given `id` exists.

Exceptions

`RemoteException`	if a system-level error occurs.

This method checks if a process definiton with the given ids exists.

A.2.43.7. processDefinitions()

```
  public java.util.Collection processDefinitions()
    throws RemoteException;
```

Parameters

return	collection

Exceptions

`RemoteException`	if a system-level error occurs.

This operation method delivers a collection of all defined process definitions. The elements of the list are of the type `ProcessDefinition`

A.2.43.8. processMgr(String, String)

```
  public ProcessMgr processMgr(String packageId,
                               String processId)
    throws InvalidKeyException, RemoteException;
```

Parameters

`packageId`	Id attribute of the process package.
`processId`	Id attribute of the process.
return	the process manager for the process type.

Exceptions

`InvalidKeyException`	if not process definition with the given ids exists.
`RemoteException`	if a system-level error occurs.

This method delivers the process manager for the process definition with the given ids.

A.2.43.9. removeProcessDefinition(String, String)

```
  public void removeProcessDefinition(String packageId,
                                      String processId)
    throws RemoteException, InvalidKeyException;
```

Parameters

`packageId`	Id attribute of the process package.
`processId`	Id attribute of the process.

Exceptions

`RemoteException`	if a system-level error occurs.
`InvalidKeyException`	if packageId or processId are (formally) invalid ids.

This operation method removes a process definition with the given ids from the database. If called for a definition that does not exist, it does nothing.

A.2.43.10. setEnabled(String, String, boolean)

```
  public void setEnabled(String packageId,
                         String processId,
                         boolean enabled)
    throws RemoteException, InvalidKeyException;
```

Parameters

`packageId`	Id attribute of the process package.
`processId`	Id attribute of the process.
`enabled`	enable the process definition or not.

Exceptions

`InvalidKeyException`	if no process definition with the given ids exists.
`RemoteException`	if a system-level error occurs.

This operation method set the process definition with the given ids as enabled or disabled.

A.2.44. Interface ProcessDirectory

Client interface for the process directory.

A.2.44.1. Synopsis

```
 public interface de.danet.an.workflow.api.ProcessDirectory extends, de.danet.an.w
// Public Methods

  public Activity lookupActivity(ActivityUniqueKey key)
    throws InvalidKeyException, RemoteException;


  public Activity.Info lookupActivityInfo(ActivityUniqueKey key)
    throws InvalidKeyException, RemoteException;


  public Process lookupProcess(String processMgrName,
                               String processKey)
    throws InvalidKeyException, RemoteException;


  public java.util.Collection processMgrNames()
    throws RemoteException;


  public java.util.Collection processNames()
    throws RemoteException;


  public java.util.Collection processes()
    throws RemoteException;


  public RangeAccess processes(query.FilterCriterion filter,
                               query.SortCriterion order)
    throws RemoteException;


  public void removeProcess(de.danet.an.workflow.omgcore.WfProcess process)
    throws RemoteException, CannotRemoveException;

}
```

Inheritance Path. Section A.2.44, “Interface ProcessDirectory” [230]

A.2.44.2. lookupActivity(ActivityUniqueKey)

```
  public Activity lookupActivity(ActivityUniqueKey key)
    throws InvalidKeyException, RemoteException;
```

Parameters

`key`	denotes the activity to be looked up.
return	the corresponding `Activity` value

Exceptions

`RemoteException`	if a system-level error occurs.
`InvalidKeyException`	if the activity specified by `key` cannot be found.

Ejb.interface-method	view-type="remote"

This method returns the activity identified be the given unique key.

A.2.44.3. lookupActivityInfo(ActivityUniqueKey)

```
public Activity.Info lookupActivityInfo(ActivityUniqueKey key)
  throws InvalidKeyException, RemoteException;
```

Parameters

`key`	denotes the activity to be looked up.
return	the corresponding `Activity.Info` value

Exceptions

`InvalidKeyException`	if the activity specified by `key` cannot be found.
`RemoteException`	if a system-level error occurs.

This method finds the activity identified be the given unique key and returns all available information about it. This is a shortcut for first looking up the activity and then retrieving the "essential" information about it.

A.2.44.4. lookupProcess(String, String)

```
public Process lookupProcess(String processMgrName,
                             String processKey)
  throws InvalidKeyException, RemoteException;
```

Parameters

`processMgrName`	type of the given process.
`processKey`	key of the process.
return	the process found.

Exceptions

`InvalidKeyException`	if no such process can be found.
`RemoteException`	if a system-level error occurs.

This method finds the process identified by the given type and key.

A.2.44.5. processes()

```
public java.util.Collection processes()
  throws RemoteException;
```

Parameters

return	collection of `Process`

Exceptions

`RemoteException`	if a system-level error occurs.

This operation method delivers a collection of all known processes. The objects of the collection are remote interface of type `Process` .

A.2.44.6. processes(FilterCriterion, SortCriterion)

```
public RangeAccess processes(query.FilterCriterion filter,
                             query.SortCriterion order)
  throws RemoteException;
```

Parameters

`filter`	a filter for the result
`order`	the sort order for the result
return	access object to `Processes`

Exceptions

`RemoteException`	if a system-level error occurs.

This method returns an access object to an ordered set of processes. The objects in the result are remote interface of type `Process` .

A.2.44.7. processMgrNames()

```
public java.util.Collection processMgrNames()
  throws RemoteException;
```

Parameters

return	collection of String

Exceptions

`RemoteException`	if a system-level error occurs.

This operation method returns a collection of the defined process types as Strings.

A.2.44.8. processNames()

```
public java.util.Collection processNames()
  throws RemoteException;
```

Parameters

return	collection of process names as String objects or an empty collection if no processes are found.

Exceptions

`RemoteException`	if a system-level error occurs.

This operation method delivers a collection of process names of created processes or an empty collection if no processes are found.

A.2.44.9. removeProcess(WfProcess)

```
public void removeProcess(de.danet.an.workflow.omgcore.WfProcess process)
  throws RemoteException, CannotRemoveException;
```

Parameters

`process`	the process to remove.

Exceptions

`RemoteException`	if a system-level error occurs.
`CannotRemoveException`	if the process cannot be removed, because it is still in progress.

Removes the given process. The process can be removed, only if it is in state "closed" or "not started".

A.2.45. Interface ProcessMgr

Interface `ProcessMgr` adds some functions to the `OMG process manager` .

The meta information returned by `contextSignature` and `resultSignature` defined in `WfProcessMgr` uses Java classes to represents primitive types, as specified for `Process-DataInfo` . In addition the type of a data item can be indicated as

`org.w3c.dom.Element.class`	denotes XML with unknown structure. Values of this type are represented as instances of `SAXEventBuffer` .
an instance of `SAXEventBuffer`	denotes XML with the known structure represented by the XML in the event buffer. Values of this type are represented as instances of `SAXEventBuffer` .
an instance of `ExternalReference`	denotes the type as specified in the process definition. If a Java type is specified as fully qualified Java class name in the localtion attribute, "java:" is prepended as protocol and values are of the specified type. Else values of this type are represented as instances of `SAXEventBuffer` .

A.2.45.1. Synopsis

```
 public interface de.danet.an.workflow.api.ProcessMgr extends, de.danet.an.workflo
// Public Methods

  public java.util.Collection findByDataItem(String itemName,
                                             String itemValue)
    throws RemoteException;


  public Process processByKey(String key)
    throws RemoteException, InvalidKeyException;

}
```

Inheritance Path. Section A.2.45, “Interface ProcessMgr” [233]

A.2.45.2. findByDataItem(String, String)

```
  public java.util.Collection findByDataItem(String itemName,
                                             String itemValue)
    throws RemoteException;
```

Parameters

`itemName`	the name of the process data item
`itemValue`	the value of the process data item
return	the collection of processes

Exceptions

`RemoteException`	if a system-level error occurs

Returns all `Process` es that have a given value in a particular process data item. Note that this method may only be used for data items of type string.

A.2.45.3. processByKey(String)

```
  public Process processByKey(String key)
    throws RemoteException, InvalidKeyException;
```

Parameters

`key`	the `key` of the process
return	the process associated with the key

Exceptions

`InvalidKeyException`	if no process with the given key exists
`RemoteException`	if a system-level error occurs

Returns the `Process` with the given key. The OMG interface only defines a `method for listing all processes` associated with a process manager. While, of course, one could select the process with a certain key from that list, this would be rather inefficient.

A.2.46. Interface RangeAccess

This interface defines paginated access to data.

A.2.46.1. Synopsis

```
 public interface de.danet.an.workflow.api.RangeAccess {
// Public Methods

  public long itemCount()
    throws RemoteException;


  public java.util.List items(long start,
                              long end)
    throws RemoteException;

}
```

Inheritance Path. Section A.2.46, “Interface RangeAccess” [235]

A.2.46.2. itemCount()

```
  public long itemCount()
    throws RemoteException;
```

Parameters

return	the number of items

Exceptions

`RemoteException`	if a system-level error occurs.

Return the number of items in the underlying list.

A.2.46.3. items(long, long)

```
  public java.util.List items(long start,
                              long end)
    throws RemoteException;
```

Parameters

`start`	the start index (zero based)
`end`	the end index
return	the selected items

Exceptions

`RemoteException`	if a system-level error occurs.

Return the subset of items from `start` to `end` .

A.2.47. Interface RoleResource

This interface extends the interface `WfResource` for resources that are roles. The distinction of resource types is without relevance to the workflow engine, but may be used to e.g. display resource types differently in the user interface. Resource management services are not required to make use of this interface, they may deliver all resources as plain `WfResource` s. Application must therefore be prepared to handle resources that only implement the base interface `WfResource` .

A.2.47.1. Synopsis

```
 public interface de.danet.an.workflow.api.RoleResource extends, de.danet.an.workf
}
```

See Also	`Section A.2.50, "Interface UserResource" [240] , Section A.2.27, "Interface GroupResource" [190]`

Inheritance Path. Section A.2.47, “Interface RoleResource” [236]

A.2.48. Interface SAXEventBuffer

This interface is implemented by classes that can provide XML content by emitting SAX events.

Applications that use XML at their interfaces can sometimes not avoid using XML like data structures for internal data representation. Holding this data as XML is not very time efficient, because it must be parsed in order to be processed (although parsing time for XML is often overestimated).

Holding the data in either W3C DOM or JDOM representations is not very memory efficient. Depending on the implementation, DOM representations use a factor of 6 to 10 compared with the file space of the same description.

A SAX event buffer is a compromise. It stores the events generated by a SAX parser using as little space as possible while providing the data very quickly to a content Handler.

A.2.48.1. Synopsis

```
 public interface de.danet.an.workflow.api.SAXEventBuffer {
// Public Methods

  public void emit(org.xml.sax.ContentHandler contentHandler)
    throws SAXException;

}
```

Inheritance Path. Section A.2.48, “Interface SAXEventBuffer” [236]

A.2.48.2. emit(ContentHandler)

```
  public void emit(org.xml.sax.ContentHandler contentHandler)
    throws SAXException;
```

Parameters

`contentHandler`	the content handler that is to receive the events.

Exceptions

`SAXException`	any SAX exception, possibly wrapping another exception.

Emits the events to the given `ContentHandler` .

A.2.49. Interface Transition

Represents a transition between two activities.

Methods of this interface do not throw `RemoteException` s as they are read-only and the data is immutable and simple.

A.2.49.1. Synopsis

```
 public interface de.danet.an.workflow.api.Transition {
// Public Static Fields

  public static final int COND_TYPE_CONDITION  = 1;


  public static final int COND_TYPE_DEFAULTEXCEPTION  = 4;


  public static final int COND_TYPE_EXCEPTION  = 3;
  public static final int COND_TYPE_OTHERWISE  = 2;
// Public Methods
  public String condition();

  public int conditionType();

  public Activity from();

  public String group();

  public String id();

  public int order();

  public Activity to();
}
```

Inheritance Path. Section A.2.49, “Interface Transition” [237]

A.2.49.2. COND_TYPE_CONDITION

```
  public static final int COND_TYPE_CONDITION  = 1;
```

Condition type "condition".

A.2.49.3. COND_TYPE_DEFAULTEXCEPTION

```
  public static final int COND_TYPE_DEFAULTEXCEPTION  = 4;
```

Condition type "default exception".

A.2.49.4. COND_TYPE_EXCEPTION

```
  public static final int COND_TYPE_EXCEPTION  = 3;
```

Condition type "exception".

A.2.49.5. COND_TYPE_OTHERWISE

```
  public static final int COND_TYPE_OTHERWISE  = 2;
```

Condition type "OTHERWISE".

A.2.49.6. condition()

```
  public String condition();
```

Parameters

return	the condition of this transition

Returns the condition associated with this transition.

A.2.49.7. conditionType()

```
public int conditionType();
```

Parameters

return	type of the condition of this transition

Returns the type of the condition associated with this transition.

A.2.49.8. from()

```
public Activity from();
```

Parameters

return	the source activity

Returns the "From" activity of this transition.

A.2.49.9. group()

```
public String group();
```

Parameters

return	the transition group

Return the identifier of the transition group this transition belongs to.

A.2.49.10. id()

```
public String id();
```

Parameters

return	the id

Returns the id of this transition.

A.2.49.11. order()

```
public int order();
```

Parameters

return	the priority

Return the priority of this transition. The priority determines the sequence in which transitions from activties with XOR split are evaluated.

A.2.49.12. to()

```
public Activity to();
```

Parameters

return	the destination activity

Returns the "To" activity of this transition.

A.2.50. Interface UserResource

This interface extends the interface `WfResource` for resources that are users. The distinction of resource types is without relevance to the workflow engine, but may be used to e.g. display resource types differently in the user interface. Resource management services are not required to make use of this interface, they may deliver all resources as plain `WfResource` s. Application must therefore be prepared to handle resources that only implement the base interface `WfResource` .

A.2.50.1. Synopsis

```
 public interface de.danet.an.workflow.api.UserResource extends, de.danet.an.workf
}
```

See Also	Section A.2.27, “Interface GroupResource” [190] , Section A.2.47, “Interface RoleResource” [236]

Inheritance Path. Section A.2.50, “Interface UserResource” [240]

A.2.51. Interface WorkflowService

This interface defines the methods provided by the workflow engine.

A.2.51.1. Synopsis

```
 public interface de.danet.an.workflow.api.WorkflowService extends, de.danet.an.wo
// Public Methods

  public de.danet.an.workflow.omgcore.WfResource asResource(java.security.Principa
    throws RemoteException, InvalidKeyException;


  public java.util.Collection authorizers(de.danet.an.workflow.omgcore.WfResource
    throws RemoteException;


  public java.security.Principal caller()
    throws RemoteException;


  public Configuration configuration()
    throws RemoteException;


  public EventSubscriber createEventSubscriber()
    throws IOException;


  public EventSubscriber createEventSubscriber(String processKey,
                                               String eventTypes)
    throws IOException;


  public void doFinish(de.danet.an.workflow.omgcore.WfActivity act,
                       de.danet.an.workflow.omgcore.ProcessData result)
    throws InvalidDataException, CannotCompleteException, RemoteException;


  public de.danet.an.workflow.omgcore.WfObject eventReceiver(de.danet.an.wor
    throws RemoteException;


  public Object executeBatch(Batch batch)
    throws RemoteException, InvocationTargetException;


  public Channel getChannel(de.danet.an.workflow.omgcore.WfProcess process,
                            String channelName)
    throws RemoteException;


  public Channel getChannel(de.danet.an.workflow.omgcore.WfProcess process,
                            String channelName,
                            boolean sendOnly)
    throws RemoteException;


  public java.util.Collection knownResources()
    throws RemoteException;


  public ProcessDefinitionDirectory processDefinitionDirectory()
    throws RemoteException;


  public ProcessDirectory processDirectory()
    throws RemoteException;


  public void registerRequester(de.danet.an.workflow.omgcore.WfRequester req
    throws RemoteException;


  public void release(de.danet.an.workflow.omgcore.WfObject obj);


  public java.util.Collection requestedBy(de.danet.an.workflow.omgcore.WfReq
    throws RemoteException;


  public de.danet.an.workflow.omgcore.WfResource resourceByKey(String key)
    throws InvalidKeyException, RemoteException;
  public java.util.Map serviceProperties()
    throws RemoteException;

}
```

Inheritance Path. Section A.2.51, “Interface WorkflowService” [240]

A.2.51.2. asResource(Principal)

```
  public de.danet.an.workflow.omgcore.WfResource asResource(java.security.Principa
    throws RemoteException, InvalidKeyException;
```

Parameters

`principal`	the principal.
return	a `WfResource` object corresponding to the given principal.

Exceptions

`InvalidKeyException`	if a resource with the given principal can't be found.
`RemoteException`	if a system-level error occurs.

Since	1.2

Given a principal, return the workflow resource associated with this principal. This method is usually used to get a `WfResource` object corresponding to the current user. The `WfResource` object can subsequently be used to e.g. determine the current user's worklist.

Calls to this method are typically delegated to `ResourceAssignmentService.asResource` . Note that since implementation of this method by the resource assignment service is optional, calling this method may result in an `UnsupportedOperationException` .

A.2.51.3. authorizers(WfResource)

```
  public java.util.Collection authorizers(de.danet.an.workflow.omgcore.WfResource
    throws RemoteException;
```

Parameters

`resource`	the resource.
return	a collection of `WfResource` objects, not including `resource`

Exceptions

`RemoteException`	if a system-level error occurs.

Since	1.2

Given a `WfResource object` , return the collection of resources this resource is authorized for.

This method usually returns all groups the resource is a member of and all roles assigned to the resource.

Calls to this method are typically delegated to `ResourceAssignmentService.authorizers` . Note that since implementation of this method by the resource assignment service is optional, calling this method may result in an `UnsupportedOperationException` .

A.2.51.4. caller()

```
public java.security.Principal caller()
  throws RemoteException;
```

Parameters

return	the caller principal.

See Also	`asResource(java.security.Principal) [242]`
Ejb.interface-method	view-type="remote"
Ejb.transaction	type="Supports"

Returns the currently (i.e. for this call) authenticated user as a `Principal` .

In an environment where authentication is performed by the container, it may not be easy to discover this information. Although most container environments provide a possibility to access the current principal, there may be a mapping between the client's representation of the currently authenticated user and the workflow engine's (i.e. server's) representation.

A.2.51.5. configuration()

```
public Configuration configuration()
  throws RemoteException;
```

Parameters

return	the configuration.

Exceptions

`RemoteException`	if a system-level error occurs.

Return the workflow engine configuration.

A.2.51.6. createEventSubscriber()

```
  public EventSubscriber createEventSubscriber()
    throws IOException;
```

Parameters

return	the subscriber

Exceptions

`IOException`	if an error occurs.

Returns an event subscriber. Event subscriber should be released using `release` when no longer needed as they may consume considerable resources.

A.2.51.7. createEventSubscriber(String, String)

```
  public EventSubscriber createEventSubscriber(String processKey,
                                               String eventTypes)
    throws IOException;
```

Parameters

`processKey`	if not `null`, receive events for the given process only
`eventTypes`	if not `null` , receive events of the given types only. Types are specified as a whitespace, comma or semicolon separated list of event names. See `WfAuditEvent` for a list of valid event names.
return	the subscriber

Exceptions

`IOException`	if an error occurs.

Returns an event subscriber that receives events as specified by the parameters. Event subscriber should be released using `release` when no longer needed as they may consume considerable resources.

A.2.51.8. doFinish(WfActivity, ProcessData)

```
  public void doFinish(de.danet.an.workflow.omgcore.WfActivity act,
                       de.danet.an.workflow.omgcore.ProcessData result)
    throws InvalidDataException, CannotCompleteException, RemoteException;
```

Parameters

`act`	the `Activity` .
`result`	the tool's result data. If `null` do not call `setResult` .

Exceptions

`InvalidDataException`	see `WfActivity.setResult(...)`
`CannotCompleteException`	see `WfActivity.complete()`
`RemoteException`	if a system-level error occurs.

Since	1.1

Set a result and complete an activity in a new transaction. This is usually required to implement tools with reasonable behaviour since a failure when calling `setResult` or `complete` on an activity may not cause the tool invocation to be repeated (as would be the case when simply calling `setResult` or `complete` due to the transaction rollback associated with an exception).

As an example consider an `InvalidDataException` . This usually occurs when the result contains an item that is not a declared process data field. Ususally, repeating the tool invocation will not solve this problem.

A.2.51.9. eventReceiver(WfAuditHandler)

```
public de.danet.an.workflow.omgcore.WfObject eventReceiver(de.danet.an.wor
  throws RemoteException;
```

Parameters

`handler`	the handler for received events.
return	the receiver.

Exceptions

`RemoteException`	if a system-level error occurs.

Deprecated

since version 1.3.2. Use `createEventSubscriber` instead and set a handler for the object thus obtained

Returns an event receiver. The events received will be handled by the given handler. Event receivers should be released using `release` when no longer needed as they may consume considerable resources.

A.2.51.10. executeBatch(Batch)

```
public Object executeBatch(Batch batch)
  throws RemoteException, InvocationTargetException;
```

Parameters

`batch`	the batch to be executed.
return	the result returned by `Batch.execute` .

Exceptions

`RemoteException`	if a system-level error occurs.
`InvocationTargetException`	if thrown by `Batch.execute`

Execute a batch in the context of the workflow service i.e. on the server.

We do not want to define a specific mechanism for implementing the remote invocation mechanism used with the workflow API. Yet it is obvious that any implementation can profit from the possibility to execute several actions as one call to the server.

A.2.51.11. getChannel(WfProcess, String)

```
public Channel getChannel(de.danet.an.workflow.omgcore.WfProcess process,
                          String channelName)
  throws RemoteException;
```

Parameters

`process`	the process to communicate with
`channelName`	the channel name
return	the channel

Exceptions

`RemoteException`	if a system-level error occurs.

Return a named communication channel to the given process. The channel may be used to send messages to the receiver tool and receive messages from the sender tool.

Channels should be `released` when no longer needed to free resources.

A.2.51.12. getChannel(WfProcess, String, boolean)

```
public Channel getChannel(de.danet.an.workflow.omgcore.WfProcess process,
                          String channelName,
                          boolean sendOnly)
  throws RemoteException;
```

Parameters

`process`	the process to communicate with
`channelName`	the channel name
`sendOnly`	if set, returns a channel that may only be used for sending messages. This may save some resources.
return	the channel

Exceptions

`RemoteException`	if a system-level error occurs.

Return a named communication channel to the given process. The channel may be used to send messages to the receiver tool and optionally receive messages from the sender tool.

Channels should be `released` when no longer needed to free resources.

A.2.51.13. knownResources()

```
public java.util.Collection knownResources()
  throws RemoteException;
```

Parameters

return	the collection of the known resources to the ras (instances of `WfResource`).

Exceptions

`RemoteException`	if a system-level error occurs.

Since	1.2
See Also	ResourceAssignmentService#knownResources

Returns at least the collection of all the workflow resources that have been assigned work items, but optionally it can return the additional workflow resources that are known to the resource assignment service. Calls to this method are typically delegated to `ResourceAssignmentService.knownResources` . Note that since implementation of this method by the resource assignment service is optional, calling this method may result in an `UnsupportedOperationException` .

A.2.51.14. processDefinitionDirectory()

```
public ProcessDefinitionDirectory processDefinitionDirectory()
  throws RemoteException;
```

Parameters

return	the process definition directory.

Exceptions

`RemoteException`	if a system-level error occurs.

Return the process definition directory of the workflow service.

A.2.51.15. processDirectory()

```
public ProcessDirectory processDirectory()
  throws RemoteException;
```

Parameters

return	the process directory.

Exceptions

`RemoteException`	if a system-level error occurs.

Return the process directory of the workflow service.

A.2.51.16. registerRequester(WfRequester)

```
public void registerRequester(de.danet.an.workflow.omgcore.WfRequester requester
  throws RemoteException;
```

Parameters

`requester`	the requester to be registered.

Exceptions

`RemoteException`	if a system-level error occurs.

Register a `WfRequester` . Registered requesters' `receiveEvent` methods will be called for their performers. Note that a requester must be registered before it is used for process creation. Else events may be lost.

A.2.51.17. release(WfObject)

```
public void release(de.danet.an.workflow.omgcore.WfObject obj);
```

Parameters

`obj`	the object which is no longer used.

Release an object obtained from the workflow service immediately instead of waiting for it to be automatically released. This may be called to optimize resource utilization.

We do not want to define a specific mechanism for implementing the remote invocation mechanism used with the workflow API. Therefore, we cannot demand that each object defines a method to release resources (like e.g. CORBA's release). This method knows about the implementation specifics and acts appropriately.

A.2.51.18. requestedBy(WfRequester)

```
public java.util.Collection requestedBy(de.danet.an.workflow.omgcore.WfReq
  throws RemoteException;
```

Parameters

`req`	the requester.
return	the processes created with the given requester.

Exceptions

`RemoteException`	if a system-level error occurs.

Return the processes requested by the given requester. This is a helper method intended to be used when implementing a `WfRequester` . Applications should use `WfRequester.performers()` instead.

A.2.51.19. resourceByKey(String)

```
public de.danet.an.workflow.omgcore.WfResource resourceByKey(String key)
  throws InvalidKeyException, RemoteException;
```

Parameters

`key`	the key.
return	a `WfResource` object corresponding to the given key.

Exceptions

`InvalidKeyException`	if the resource with the given key can't be found. As the environment is a concurrent multi user environment, `WfResource` objects (and keys obtained from `WfResource` objects) may become invalid.
`RemoteException`	if a system-level error occurs.

Since	1.2
See Also	ResourceAssignmentService#resourceByKey

Given the `key` of a `WfResource` (obtained with `resourceKey()`), return the workflow resource associated with this key.

Calls to this method are typically delegated to `ResourceAssignmentService.resourceByKey` . Note that since implementation of this method by the resource assignment service is optional, calling this method may result in an `UnsupportedOperationException` .

A.2.51.20. serviceProperties()

```
public java.util.Map serviceProperties()
  throws RemoteException;
```

Parameters

return	the service properties

Exceptions

`RemoteException`	if a system-level error occurs

Returns the properties that uniquely decribe the workflow service in the current environment.

`WorkflowService` does not implement `Serializable` because implementations of this class may have attributes that e.g. include network connections to the server and may thus not be serializable. Nevertheless it should be possible to obtain some unique reference to a workflow service and to restore this service without having to "manually" collect the (implementation dependant!) properties set for `WorkflowServiceFactory` before the call to `newInstance` .

This methods therefore returns a set of relevant properties that will restore this workflow service when set as properties of `WorkflowServiceFactory` in the same or an equivalent environment before `newInstance` is called. The properties returned by this method are, of course, based on the properties in effect when the `WorkflowService` was initially created.

Note the restriction "same or equivalent environment" in the previous paragraph. One of the explicitly mentioned properties of the `WorkflowServiceFactory` (in a J2EE based implementation) is the `InitialContext` used. If not set explicitly, the default initial context may be specified by something like " `localhost:1099` ". While the properties returned by `serviceProperties` will include this property of the connection to the JNDI provider, using the properties in a different JVM on a different machine may result in a different workflow service (or no workflow service at all) because a different JNDI server is accessed. The impossibility to transfer all relevant information between JVM's under all circumstances (`InitialContext` is not serializable as may be some other crucial information in an implementation based on some other technology than J2EE) has prevented us from demanding serializability for `WorkflowService` . The requirement to obtain the service information explicitly and to create a new instance should result in some awareness of the problems.

A.2.52. Class WorkflowServiceFactory

This class provides a factory API that enables clients to obtain a workflow service facility.

A.2.52.1. Synopsis

```
 public abstract class de.danet.an.workflow.api.WorkflowServiceFactory {
// Protected Constructors

  protected WorkflowServiceFactory();

// Public Static Methods

  public static WorkflowServiceFactory newInstance()
    throws FactoryConfigurationError;

// Public Methods

  public abstract WorkflowService newWorkflowService()
    throws FactoryConfigurationError;


  public void setProperties(java.util.Map props);


  public void setProperty(String name,
                          Object value);

// Protected Methods

  protected java.util.Map getProperties();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.2.52, “Class WorkflowServiceFactory” [250]

A.2.52.2. WorkflowServiceFactory()

```
  protected WorkflowServiceFactory();
```

Constructor. Must be overridden with a parameterless public constructor by derived class.

A.2.52.3. getProperties()

```
  protected java.util.Map getProperties();
```

Parameters

return	the defined properties

Used by derived classes to access the properties.

A.2.52.4. newInstance()

```
  public static WorkflowServiceFactory newInstance()
    throws FactoryConfigurationError;
```

Parameters

return	an instance of the `WorkflowServiceFactory` .

Exceptions

`FactoryConfigurationError`	if a factory instance can't be created.

Obtain a new instance of a `WorkflowServiceFactory` . This static method creates a new factory instance. The method uses the following ordered lookup procedure to determine the `WorkflowServiceFactory` implementation class to load:

- If an initial naming context is available, look for a a classname in `java:comp/env/de.danet.an.workflow.api.WorkflowServiceFactory` . The configuration for a class as workflow service factory thus looks like:

  ```
  <env-entry>
     <description>Configure the workflow service factory</description>
     <env-entry-name>de.danet.an.workflow.api.WorkflowServiceFactory</env-entry-\
  name>
     <env-entry-type>java.lang.String</env-entry-type>
     <env-entry-value>
                FactoryImplementationClass
  </env-entry-value>
   </env-entry>
  ```

  Note that this environment entry must be inserted in the `ejb-jar.xml` or `web.xml` for every EJB resp. servlet that calls the `newInstance` method of `WorkflowServiceFactory` .

- Use the services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file `META-INF/services/de.danet.an.workflow.api.WorkflowServiceFactory` . in jars available to the runtime.

Note that the specified workflow service factory may need additional configuration parameters.

A.2.52.5. newWorkflowService()

```
public abstract WorkflowService newWorkflowService()
  throws FactoryConfigurationError;
```

Parameters

return	the workflow service.

Exceptions

`FactoryConfigurationError`	if not all required resources can be obtained.

Creates a new instance of a `WorkflowService` .

This API does not specify how a workflow service factory or workflow service should be implemented. If, however, the implementation is J2EE/EJB based, the following additional rules apply to achieve common bahaviour for J2EE based implementations.

In the J2EE environment, clients usually obtain the connection to a server from a directory service represented by an <code>InitialContext</code> instance. This instance need not be the default initial context available to the client (think of a servlet running in a servlet container that wants to access the workflow engine running in an application server on a different machine).

In an environment that uses an <code>InitialContext</code> to obtain the connection to the server (as described above), the following ordered lookup procedure must be implemented to determine this initial context.

- If the property " `javax.naming.InitialContext` " has been set, use it as initial context. if property " `javax.naming.InitialContext.Environment` " has been set, use it to obtain the initial context (see `setProperty` .
- If a default `InitialContext` is available during the execution of `newInstance` (i.e. " `new InitialContext()` succeeds), and entries `java:comp/env/de.danet.an.workflow.api.WorkflowService.NAMING_CONTEXT_FACTORY` and `java:comp/env/de.danet.an.workflow.api.WorkflowService.NAMING_CONTEXT_URL` exist, use them to obtain the initial context.
- If defined, execute vendor specific procedures to obtain an initial context.
- If a default `InitialContext` is available during the execution of `newInstance` use it (i.e. do not try to obtain another initial context (this is the common situation where a servlet based client and the workflow engine run in one application server).

A.2.52.6. setProperties(Map)

```
public void setProperties(java.util.Map props);
```

Parameters

`props`	the properties to be set

A convenience method that sets all properties in the Map.

A.2.52.7. setProperty(String, Object)

```
public void setProperty(String name,
                        Object value);
```

Parameters

`name`	the name of the property
`value`	the value of the property

Sets a property which is passed to the `WorkflowService` produced by this factory.

Valid properties generally depend on the underlying implementation. There are, however, a few exceptions.

If the workflow service implementation is based on the J2EE environment, clients derive the connection to a server from an <code>InitialContext</code>. There are cases when the user wants or needs to override the initial context used by the workflow service implementation. It is therefore defined that setting the property " `javax.naming.InitialContext` " to a value of type <code>Context</code> overrides any default method used by the workflow service implementation to obtain the initial context.

As an alternative, the property " `javax.naming.InitialContext.Environment` " may be set to a `Hashtable` that contains the environment to be used when creating an `InitialContext` .

Subsequent versions of this interface may define additional common properties. We therefore recommended to use "fully qulified" (i.e. package style) names for properties that are specific to a workflow service implementation.

A.3. Package de.danet.an.workflow.spis.aii

This package defines the application invocation interface used by the workflow package to invoke tool agents that control applications.

Java classes that are to be used as tool agents must implement the interface `ToolAgent` . Tool agents are declared in a workflow using the `application` tag in the XPDL. This declaration is associated with the implementation class of `ToolAgent` using an extension. The extension syntax supported allows to specify the Java class to use and additional properties of the tool agent (see the *User Manual* for details).

Note that a tool agent can be implemented with full functionallity based on the tool agent interface only. The remainder of this package description explains how tool agent implementation may be simplified and how the performance of tool agent invocations may be improved. As with the `client API` , we have tried to keep this extended interface as independant of the implementation framework (J2EE) as possible. However, this attempt is limited by requirements imposed by transaction handling. While the interfaces could be kept clean of dependencies on J2EE interfaces or classes, their usage pattern is partially motivated by EJB transaction semantics.

From the workflow engine's point of view, tool agent invocation is asynchronous, i.e. it does not expect a result from the `invoke` method. The invoked application must at some point in time call `WfActivity.setResult` and `WfActivity.complete` to signal completion of its work. Although the tool agent model assumes that this is done "later" (e.g. by the application process), these calls may also be made in the implementation of the `invoke` method, thus effectively combining the tool agent with the application that it controls (making the tool agent a tool).

If you try to call `setResult` and `complete` in the `invoke` method of your tool implementation, you'll sooner or later notice that these methods (being remote calls) may throw `RemoteException` s, indicating some temporary failure. If you do not handle these exceptions, they will be handled by the workflow engine and your tool will simply be re-invoked. If, however, your invoke method is "expensive" or has side effects, you may not want it to be re-executed because of a temporary failure during `setResult` or `complete` .

Consequently, you put a while loop around these calls, repeating them until they run without a `RemoteException` . Regrettably, this is where EJB semantics come in. While the repeat pattern works in a stand-alone EJB client, it won't work here because `invoke` is called in the context of an EJB method and the `RemoteException` from `setResult` or `complete` is noticed by the EJB container, and the complete tool agent invocation transaction will be rolled back. So you have to execute the calls to `setResult` and `complete` in a new transaction. This is what `ToolAgentContext.finishActivity` has been defined for. This method calls `setResult` and `complete` in a new transaction (and repeats the calls until no `RemoteException` is thrown). If you want to use this method (or another method from the tool agent context), your tool agent must implement `ContextRequester` in order to receive the context before `invoke` is called.

Having a closer look at transactions, there is a drawback to the solution described above. If we look at open transactions, we find that there is one transaction handling the tool invocation; this has been suspended for a new transaction that executes the actual tool agent invocation (this must be done in its own transaction, else the workflow engine cannot terminate an activity if an exception is thrown by the tool agent as, again, the invoking transaction would be marked for rollback by the application server). By calling `finishActivity` during `invoke` , the transaction executing the tool invocation will also be suspended in favour of the transaction that finishes the activity. This situation may, under certain circumstances lead to deadlocks.

To avoid these "excessive" nested transaction suspends, the calls to `setResult` and `complete` should better be done after tool invocation by the the same transaction that has invoked the tool. Tools (i.e. tool agents that want to return a result and complete the activity during `invoke`) should therefore implement `ResultProvider` . This allows the tool to simply leave the result evaluated during `invoke` in an attribute from where it will be retrieved after `invoke` by the workflow engine. The workflow engine then calls `setResult` and `complete` on the activity. Besides avoiding transaction problems, usage of `ResultProvider` actually simplifies the tool implementation.

A.3.1. Additional Information

Since	V1.0

A.3.2. Exception ApplicationNotStoppedException

This exception is thrown by a `ToolAgent` if it cannot `terminate` the execution of a running application. It may not be thrown if the activity passed to `terminate` is unknown (i.e. the tool agent was never invoked for the activity or processing the activity has already completed).

A.3.2.1. Synopsis

```
 public class de.danet.an.workflow.spis.aii.ApplicationNotStoppedException e
    implements, java.io.Serializable {
// Public Constructors

  public ApplicationNotStoppedException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.3.2, “Exception ApplicationNotStoppedException” [255]

A.3.2.2. ApplicationNotStoppedException(String)

```
  public ApplicationNotStoppedException(String msg);
```

Parameters

`msg`	the detail message.

Creates a new exception with the given message.

A.3.3. Exception CannotExecuteException

This exception is thrown by a `ToolAgent` if it cannot execute a given activity.

As of version 1.3.2, this exception can be used as a wrapper for an exception that caused the tool failure. The wrapped exception can be mapped to a process level exception. See `ExceptionMappingProvider` and the user manual for details.

A.3.3.1. Synopsis

```
 public class de.danet.an.workflow.spis.aii.CannotExecuteException extends, java.l
    implements, java.io.Serializable {
// Public Constructors

  public CannotExecuteException(String msg);


  public CannotExecuteException(String msg,
                                Throwable cause);

// Public Methods

  public String toString();

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.3.3, “Exception CannotExecuteException” [256]

A.3.3.2. CannotExecuteException(String)

```
  public CannotExecuteException(String msg);
```

Parameters

`msg`	the detail message.

Creates a new exception with the given message.

A.3.3.3. CannotExecuteException(String, Throwable)

```
  public CannotExecuteException(String msg,
                                Throwable cause);
```

Parameters

`msg`	the detail message
`cause`	the cause

Creates a new exception with the given message and cause.

A.3.4. Interface ContextRequester

This interface marks a `ToolAgent` as a requester of a `ToolAgentContext` . If a tool agent implements this interface, the `setContext` method will be called before the tool is invoked (i.e. before method `invoke` is called).

A.3.4.1. Synopsis

```
 public interface de.danet.an.workflow.spis.aii.ContextRequester {
// Public Methods

  public void setToolAgentContext(ToolAgentContext context);

}
```

Since	1.2

Inheritance Path. Section A.3.4, “Interface ContextRequester” [257]

A.3.4.2. setToolAgentContext(ToolAgentContext)

```
  public void setToolAgentContext(ToolAgentContext context);
```

Parameters

`context`	the engine context

Makes an engine context available to the tool agent.

A.3.5. Interface ExceptionMappingProvider

This interface must be implemented by `ToolAgent` s that want the workflow engine to map the causes of `CannotExecuteExceptions` to process exceptions that cause the current activity to be abandoned. This state change will then be handled by special transitions in the process description.

The mappings provided by a tool can always be extended or restricted by the application definition in the process description. See the user manual for details.

A.3.5.1. Synopsis

```
 public interface de.danet.an.workflow.spis.aii.ExceptionMappingProvider {
// Public Methods

  public java.util.Collection exceptionMappings();

}
```

See Also	`abandon(java.lang.String) [146]`

Inheritance Path. Section A.3.5, “Interface ExceptionMappingProvider” [257]

A.3.5.2. exceptionMappings()

```
public java.util.Collection exceptionMappings();
```

Parameters

return	the mappings

Return the collection of mappings predefined by this tool.

A.3.6. Class ExceptionMappingPro-vider.ExceptionMapping

Define a single exception mapping.

A.3.6.1. Synopsis

```
 public static class de.danet.an.workflow.spis.aii.ExceptionMappingProvider.Except
// Public Constructors

  public ExceptionMappingProvider.ExceptionMapping(Class javaException);


  public ExceptionMappingProvider.ExceptionMapping(Class javaException,
                                                   String processException);


  public ExceptionMappingProvider.ExceptionMapping(Class javaException,
                                                   String processException,
                                                   boolean suspend);

// Public Methods

  public Class getJavaException();


  public String getProcessException();


  public boolean getSuspendActivity();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.3.6, “Class ExceptionMappingPro-vider.ExceptionMapping” [258]

A.3.6.2. ExceptionMappingProvider.ExceptionMapping(Class)

```
public ExceptionMappingProvider.ExceptionMapping(Class javaException);
```

Parameters

`javaException`	the Java exception
`processException`	the process exception

Create a new mapping. If the tool throws a `CannotExecuteException` with a Java exception of the given class (or derived from the given class) as cause, then the invoking activity is suspended.

A.3.6.3. ExceptionMappingProvider.ExceptionMapping(Class, String)

```
public ExceptionMappingProvider.ExceptionMapping(Class javaException,
                                                 String processException);
```

Parameters

`javaException`	the Java exception
`processException`	the process exception

Create a new mapping. If the tool throws a `CannotExecuteException` with a Java exception of the given class (or derived from the given class) as cause, then this Java exception is mapped to the given process exception.

A.3.6.4. ExceptionMappingProvider.ExceptionMapping(Class, String, boolean)

```
public ExceptionMappingProvider.ExceptionMapping(Class javaException,
                                                 String processException,
                                                 boolean suspend);
```

Parameters

`javaException`	the Java exception
`processException`	the process exception
`suspend`	if `true` the invoking activity is suspended

Create a new mapping. If the tool throws a `CannotExecuteException` with a Java exception of the given class (or derived from the given class) as cause, then the invoking activity is suspended.

A.3.6.5. getJavaException()

```
public Class getJavaException();
```

Parameters

return	value of javaException.

Get the value of javaException.

A.3.6.6. getProcessException()

```
public String getProcessException();
```

Parameters

return	value of processException.

Get the value of processException.

A.3.6.7. getSuspendActivity()

```
public boolean getSuspendActivity();
```

Parameters

return	Returns `true` if the invoking activity is to be suspended in response to the exception.

A.3.7. Interface ExecutionModeProvider

This interface can be implemented by `ToolAgents` that provide information about their preferred execution mode.

Note that if a tool agent implements this interface, the mode returned by `executionMode` overrides any settings in XPDL.

A.3.7.1. Synopsis

```
 public interface de.danet.an.workflow.spis.aii.ExecutionModeProvider {
// Public Static Fields

  public static final int ASYNCHR  = 1;

  public static final int SYNCHR  = 2;

// Public Methods

  public int executionMode();

}
```

Deprecated

As of version 1.2, there are no different execution modes any more.

Inheritance Path. Section A.3.7, "Interface ExecutionModeProvider" [260]

A.3.7.2. ASYNCHR

```
public static final int ASYNCHR  = 1;
```

Indicates that the tool is to be executed asynchronously with respect to state evaluation (see user manual for details).

A.3.7.3. SYNCHR

```
  public static final int SYNCHR  = 2;
```

Indicates that the tool is to be executed synchronously with respect to state evaluation (see user manual for details).

A.3.7.4. executionMode()

```
  public int executionMode();
```

Parameters

return	one of `ASYNCHR` or `SYNCHR`

Returns the preferred execution mode of the tools.

A.3.8. Interface ResultProvider

This interface marks a `ToolAgent` as provider of a result during `invoke` .

A.3.8.1. Synopsis

```
 public interface de.danet.an.workflow.spis.aii.ResultProvider {
// Public Methods

  public Object result();

}
```

Since	1.1
See Also	`de.danet.an.workflow.spis.aii [254]`

Inheritance Path. Section A.3.8, “Interface ResultProvider” [261]

A.3.8.2. result()

```
  public Object result();
```

Parameters

return	the result data as a <code>Map</code> of formal parameter names and values or `null` if the invocation does not return any data. As special case, an instance of `ExceptionRes-ult` may be returned if the activity is to be abandoned.

Return the result evaluated during `invoke` . The method will only be called once after each invoke, i.e. the attribute holding the result may be cleared in this method to allow early garbage collection.

Note that since a tool agent implementation must be thread safe, the result evaluated during `invoke` must be kept in a `ThreadLocal` attribute.

A.3.9. Class ResultProvider.ExceptionResult

A special kind of result that may be returned by `result` . It causes the activity to be abandoned and the exception name used in the constructor to be signaled to the workflow engine.

A.3.9.1. Synopsis

```
 public static class de.danet.an.workflow.spis.aii.ResultProvider.ExceptionResulti
// Public Constructors

  public ResultProvider.ExceptionResult(String exceptionName);


  public ResultProvider.ExceptionResult(String exceptionName,
                                        boolean suspendActivity);

// Public Methods

  public String exceptionName();


  public boolean suspendActivity();


  public String toString();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

See Also	`abandon(java.lang.String) [146]`

Inheritance Path. java.lang.Object-> Section A.3.9, “Class ResultProvider.ExceptionResult” [262]

A.3.9.2. ResultProvider.ExceptionResult(String)

```
  public ResultProvider.ExceptionResult(String exceptionName);
```

Parameters

`exceptionName`	the name of the exception to be signaled

May be used by a tool agent implementation to create a special result that causes the activity to be abandoned, and the exception with the given name to be signaled to the workflow engine for transition evaluation.

A.3.9.3. ResultProvider.ExceptionResult(String, boolean)

```
  public ResultProvider.ExceptionResult(String exceptionName,
                                        boolean suspendActivity);
```

Parameters

`exceptionName`	the name of the exception to be signaled
`suspendActivity`	if `true` suspend the invoking activity

May be used by a tool agent implementation to create a special result that causes the activity to be abandoned, and the exception with the given name to be signaled to the workflow engine for transition evaluation. In addition, the invoking activity may be suspended.

A.3.9.4. exceptionName()

```
public String exceptionName();
```

Parameters

return	the exception name

Return the exception name passed to the constructor.

A.3.9.5. suspendActivity()

```
public boolean suspendActivity();
```

Parameters

return	the suspend activity flag

Return the suspend activity flag passed to the constructor.

A.3.10. Interface ToolAgent

This interface is used to control applications that execute work items. An application may implement this interface directly (it is "workflow aware"). Usually, however, this interface will be implemented by some adapter class that controls the application, and thus acts as an agent for the application.

An application is declared to participate in a workflow by the `application` tag in the XPDL. This declaration is associated with the implementation class of `ToolAgent` using an extension. The extension syntax supported allows to specify properties of the tool agent (see the *User Manual* for details).

An implementation of `ToolAgent` must be aware that several application invocations may be performed using the same instance of the class implementing `ToolAgent` . This implies that implementations must be thread-safe.

By default, the `invoke` method has full access to the activity as specified by the `activity interface` . Implementations that require long term storage of the activity must be aware that the activity is passed as an interface to a remote object and therefore unsuitable for long term persistence. Tool agents should persist the activity's unique key instead of the activity. The activity may later be retrieved from the key using `ProcessDirectory.lookupActivity` (obtaining the process directory from the `WorkflowService` .

A.3.10.1. Synopsis

```
 public interface de.danet.an.workflow.spis.aii.ToolAgent {
// Public Methods

  public void invoke(de.danet.an.workflow.api.Activity activity,
                     de.danet.an.workflow.api.FormalParameter[] formalParameters,
                     java.util.Map actualParameters)
    throws RemoteException, CannotExecuteException;


  public void terminate(de.danet.an.workflow.api.Activity activity)
    throws ApplicationNotStoppedException, RemoteException;

}
```

See Also	`de.danet.an.workflow.spis.aii [254]`

Inheritance Path. Section A.3.10, "Interface ToolAgent" [263]

A.3.10.2. invoke(Activity, FormalParameter[], Map)

```
  public void invoke(de.danet.an.workflow.api.Activity activity,
                     de.danet.an.workflow.api.FormalParameter[] formalParameters,
                     java.util.Map actualParameters)
    throws RemoteException, CannotExecuteException;
```

Parameters

`activity`	the activity to be executed. The supplied object must be serializable in order to support applications running as servers
`formalParameters`	the formal parameter definition as specified in the process definition. May be used e.g. by generic applications to adapt to specific formal parameter lists
`actualParameters`	the actual parameters of the application invocation. There is an entry in the map for every formal parameter using the id as key in the map. `OUT` parameters are initialized to zero. The map and the objects contained in the map must be serializable in order to support applications running as servers.

Exceptions

`RemoteException`	if a temporary problem occurs and the workflow engine should retry the tool invocation (usually thrown when a deadlock situation occurs while accessing the activity).
`CannotExecuteException`	if thrown, causes the activity to be terminated unless a mapping to a process exception is defined for the cause of the `CannotExecuteException` . If such a mapping is defined, the activity is abandoned. See the user manual for details.

Invoke an application on the given activity.

A.3.10.3. terminate(Activity)

```
public void terminate(de.danet.an.workflow.api.Activity activity)
  throws ApplicationNotStoppedException, RemoteException;
```

Parameters

`activity`	the activity to be canceled. The supplied object must be serializable in order to support applications running as servers.

Exceptions

`ApplicationNotStoppedException`	if execution cannot be terminated (see `ApplicationNotStoppedException`).
`RemoteException`	if a temporary problem occurs and the workflow engine should retry the tool termination

Terminates execution of the given activity. If the activity has been terminated already, the method should do nothing (i.e. under certain conditions this method may be called more than once for a given activity).

Only the activity's methods `key` and `uniqueKey` should be used in an implementation of this method. Else, depending on the application server and database configuration, deadlocks may occur.

Up to version 1.3 this method has been called by the workflow engine when processing an invoked tool's call of `WfActivity.complete()` . The reasoning has been that the engine cannot be sure that `complete()` is called by the tool that has previously been invoked. (To ensure this, the engine would have to pass some token to the tool during invocation and the tool would have to pass this back to the engine when calling `complete()` . Such a procedure is, however, not defined by the OMG API.) By invoking `terminate()` on the currently running tool when processing `complete()` , the engine tried to make sure that at least the invoked tool is terminated if some imposter calls `complete()` instead of the invoked tool.

The overlapping of the tool processing the completion and the invocation of `terminate()` on the tool has, however, turned out to cause various problems with respect to transactions if the tool uses a database. Therefore, this procedure has been abandoned. Note that this change does not formally change the API as there has never been a contract about `terminate()` being called as part of processing `complete()` . The subject has been explained extensively only to provide an explanation if someone observes this change of behaviour.

A.3.11. Interface ToolAgentContext

This interface defines methods of the workflow engine that are available to a tool agent. An agent implementation that wants to use these methods must implement the `ContextRequester` interface.

A.3.11.1. Synopsis

```
 public interface de.danet.an.workflow.spis.aii.ToolAgentContext extends, ja
// Public Methods

  public void abandonActivity(ResultProvider.ExceptionResult result)
    throws TransitionNotAllowedException;
  public void abandonActivity(String exception)
    throws TransitionNotAllowedException;


  public String applicationId();


  public void finishActivity(java.util.Map result)
    throws InvalidDataException, CannotCompleteException;


  public de.danet.an.workflow.api.Activity lookupActivity(de.danet.an.workflow.api
    throws InvalidKeyException, RemoteException;

}
```

Since	1.2

Inheritance Path. Section A.3.11, “Interface ToolAgentContext” [265]

A.3.11.2. abandonActivity(ResultProvider.ExceptionResult)

```
  public void abandonActivity(ResultProvider.ExceptionResult result)
    throws TransitionNotAllowedException;
```

Parameters

`result`	the exception information

Exceptions

`TransitionNotAllowedException`	see `Activity.abandon()`

Abandon the invoking activity in a new transaction and maybe suspend it.

A.3.11.3. abandonActivity(String)

```
  public void abandonActivity(String exception)
    throws TransitionNotAllowedException;
```

Parameters

`exception`	the exception to signal

Exceptions

`TransitionNotAllowedException`	see `Activity.abandon()`

Abandon the invoking activity in a new transaction.

A.3.11.4. applicationId()

```
public String applicationId();
```

Parameters

return	the id

Return the id given to the application in the process definition.

A.3.11.5. finishActivity(Map)

```
public void finishActivity(java.util.Map result)
  throws InvalidDataException, CannotCompleteException;
```

Parameters

`result`	the tool's result data. If `null` do not call `setResult` .

Exceptions

`InvalidDataException`	see `WfActivity.setResult(...)`
`CannotCompleteException`	see `WfActivity.complete()`

Set a result and complete the invoking activity in a new transaction.

A.3.11.6. lookupActivity(ActivityUniqueKey)

```
public de.danet.an.workflow.api.Activity lookupActivity(de.danet.an.workfl
  throws InvalidKeyException, RemoteException;
```

Parameters

`auk`	the activity's unique key
return	the activity

Exceptions

`InvalidKeyException`	if the activity specified by `auk` cannot be found.
`RemoteException`	if a system-level error occurs.

Lookup the given ativity specified by its unique key.

A.3.12. Interface XMLArgumentTypeProvider

This interface can be implemented by `ToolAgents` that provide information about the accepted coding of arguments that describe XML data.

Note that if a tool agent implements this interface, the type returned by `requestedXMLArgumentType` overrides any settings in XPDL.

A.3.12.1. Synopsis

```
 public interface de.danet.an.workflow.spis.aii.XMLArgumentTypeProvider {
// Public Static Fields

  public static final int XML_AS_JDOM  = 2;


  public static final int XML_AS_SAX  = 3;


  public static final int XML_AS_W3C_DOM  = 1;

// Public Methods

  public int requestedXMLArgumentType();

}
```

Inheritance Path. Section A.3.12, “Interface XMLArgumentTypeProvider” [268]

A.3.12.2. XML_AS_JDOM

```
  public static final int XML_AS_JDOM  = 2;
```

Pass parameters as JDOM when tool is invoked. The parameter is passed in as `java.util.List` , containing one or more elements, to support single-rooted XML structures as well as non-single rooted structures.

A.3.12.3. XML_AS_SAX

```
  public static final int XML_AS_SAX  = 3;
```

Pass parameters as `SAXEventBuffer` when tool is invoked.

A.3.12.4. XML_AS_W3C_DOM

```
  public static final int XML_AS_W3C_DOM  = 1;
```

Pass parameters as W3C DOM when tool is invoked. The parameter is passed in as `org.w3c.dom.DocumentFragement` , containing one or more elements, to support single-rooted XML structures as well as non-single rooted structures.

A.3.12.5. requestedXMLArgumentType()

```
  public int requestedXMLArgumentType();
```

Parameters

return	one of `XML_AS_W3C_DOM` , `XML_AS_JDOM` or `XML_AS_SAX`

Return the requested type for XML arguments.

A.4. Package de.danet.an.workflow.spis.ras

This package defines the Resource Assignment Service used in the `de.danet.an.workflow` package. This service is both used (by the core workflow component) and provided (as a sample implementation by `de.danet.an.workflow.assignment`).

The service interface follows the standard conventions for a service interface, i.e. it defines an abstract factory class and a service interface.

A.4.1. Additional Information

Since	1.0

A.4.2. Interface ActivityFinder

This interface defines facilities that map activity ids to `WfActivity` objects. `ActivityFinders` are used (in combination with an identifier) to identify activities in calls to a `ResourceAssignmentService` . They are guaranteed to be serializable objects with properly implemented <code>equals</code> and <code>hashCode</code> methods. It is expected that resource assignment facilities keep a peristent, indexed record of `ActivityFinder` s and store references to activities as tuples of a finder index and an activity identifier.

A.4.2.1. Synopsis

```
 public interface de.danet.an.workflow.spis.ras.ActivityFinder {
// Public Methods

  public de.danet.an.workflow.omgcore.WfActivity find(String actId)
    throws NoSuchActivityException, RemoteException;

}
```

Inheritance Path. Section A.4.2, “Interface ActivityFinder” [269]

A.4.2.2. find(String)

```
  public de.danet.an.workflow.omgcore.WfActivity find(String actId)
    throws NoSuchActivityException, RemoteException;
```

Parameters

`actId`	the activity id (unique in the scope of this `ActivityFinder` .
return	the `WfActivity` found.

Exceptions

`NoSuchActivityException`	if no activity with the given `actId` can be found.
`RemoteException`	if a system-level error occurs.

Return the activity that is associated with the given id.

A.4.3. Error FactoryConfigurationError

This exception is thrown by the `newInstance` method of `ResourceAssignmentService-Factory`.

A.4.3.1. Synopsis

```
 public class de.danet.an.workflow.spis.ras.FactoryConfigurationError extends, jav
// Public Constructors

  public FactoryConfigurationError();


  public FactoryConfigurationError(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Error-> Section A.4.3, “Error FactoryConfigurationError” [270]

A.4.3.2. FactoryConfigurationError()

```
  public FactoryConfigurationError();
```

Creates a new exception.

A.4.3.3. FactoryConfigurationError(String)

```
  public FactoryConfigurationError(String msg);
```

Parameters

`msg`	the detail message.

Creates a new exception with the given message.

A.4.4. Exception NoSuchActivityException

This excpetion is thrown by the `find` method of `ActivityFinder` if no activity can be found for a given key.

A.4.4.1. Synopsis

```
 public class de.danet.an.workflow.spis.ras.NoSuchActivityException extends,
    implements, java.io.Serializable {
// Public Constructors

  public NoSuchActivityException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.4.4, “Exception NoSuchActivityException” [270]

A.4.4.2. NoSuchActivityException(String)

```
  public NoSuchActivityException(String msg);
```

Parameters

`msg`	the detail message.

Create a new exception with the given detail message.

A.4.5. Interface ResourceAssignmentService

This interface defines the resource assignment facility used by the workflow component. A central design issue for this interface is the identification of activites.

The `"key" method` of `WfActivity` is by definition only unique within the scope of the containing process and can thus not easily be used to identify a single activity in a workflow engine. Even worse, a resource assignment service might be used by more than one workflow engine.

At this interface, an activity is therefore identified using an `ActivityFinder` and an identifier that is unique with respect to the `ActivityFinder` . The `ActivityFinder` provides both a namespace to allow different consumers to request resources and a means for the assignment facility to map the identifier back to an actual `WfActivity` object. See the description of `ActivityFinder` for more details.

From the workflow engine's point of view, the resource assignment service is the only source of objects of type `WfResource` and `WfAssignment` . Implementations of resource assignment services may be (but need not be) based on a resource management service as defined in package `de.danet.an.workflow.spis.rms` .

A.4.5.1. Synopsis

```
 public interface de.danet.an.workflow.spis.ras.ResourceAssignmentService {
// Public Methods

  public de.danet.an.workflow.omgcore.WfResource asResource(java.security.Pr
    throws RemoteException, InvalidKeyException;
  public java.util.Collection assignments(ActivityFinder finder,
                                          String actId,
                                          de.danet.an.workflow.omgcore.WfActivity a
    throws RemoteException;


  public java.util.Collection authorizers(de.danet.an.workflow.omgcore.WfResource
    throws RemoteException;


  public java.util.Collection autoAssignResources(ActivityFinder finder,
                                                  String actId,
                                                  de.danet.an.workflow.omgcore.WfAc
                                                  java.security.Principal principa
                                                  de.danet.an.workflow.api.Partici
    throws RemoteException;


  public void changeAssignment(ActivityFinder finder,
                               String actId,
                               de.danet.an.workflow.omgcore.WfActivity activity,
                               de.danet.an.workflow.omgcore.WfResource oldResource
                               de.danet.an.workflow.omgcore.WfResource newResource
    throws RemoteException, InvalidResourceException, AlreadyAssignedException, No


  public de.danet.an.workflow.omgcore.WfResource getResource(de.danet.an.workflow.o
    throws RemoteException;


  public boolean isMemberOfWorkItems(de.danet.an.workflow.omgcore.WfResource resou
                                     de.danet.an.workflow.omgcore.WfAssignment ass
    throws RemoteException, NoSuchResourceException;


  public java.util.Collection knownResources()
    throws RemoteException;


  public void removeAssignment(ActivityFinder finder,
                               String actId,
                               de.danet.an.workflow.omgcore.WfActivity activity,
                               de.danet.an.workflow.omgcore.WfResource resource)
    throws RemoteException, InvalidResourceException, NotAssignedException;


  public de.danet.an.workflow.omgcore.WfResource resourceByKey(String key)
    throws InvalidKeyException, RemoteException;


  public java.util.Collection workItems(de.danet.an.workflow.omgcore.WfResource re
    throws RemoteException, NoSuchResourceException;

}
```

Inheritance Path. Section A.4.5, “Interface ResourceAssignmentService” [271]

A.4.5.2. asResource(Principal)

```
  public de.danet.an.workflow.omgcore.WfResource asResource(java.security.Principa
    throws RemoteException, InvalidKeyException;
```

Parameters

`principal`	the principal.
return	a `WfResource` object corresponding to the given principal.

Exceptions

`InvalidKeyException`	if the resource with the given key can't be found.
`RemoteException`	if a system-level error occurs.

Since	1.2

Given a principal, return the workflow resource associated with this principal.

As the workflow core does not have a defined access to a resource management facility, this mapping functionality must be brought to the workflow core by the resource assignment service. If the resource assignment service is based on a resource management service as defined in package `de.danet.an.workflow.spis.rms` , it can simply delegate this call to `ResourceManagementService.asResource` . The workflow engine does not need this method for its operation; however, it provides a method for accessing this information as part of the client interface to ease the implementation of clients that e.g. want to generate a list of assignments for the current user. The implementation of this method by a resource assignment service is therefore optional. If not implemented, a call to this method must result in a <code>UnsupportedOperationException</code>.

A.4.5.3. assignments(ActivityFinder, String, WfActivity)

```
public java.util.Collection assignments(ActivityFinder finder,
                                        String actId,
                                        de.danet.an.workflow.omgcore.WfAct
  throws RemoteException;
```

Parameters

`actId`	a unique (with respect to an `ActivityFinder`) identifier for the Activity. The length of `actId` is guaranteed not to exceed 64.
`finder`	the finder used to lookup activities by their `finderId` s.
`activity`	the activity.
return	the collection of assignments (instances of `WfAssignment`).

Exceptions

`RemoteException`	if a system-level error occurs.

Return the assignments to an activity.

A.4.5.4. authorizers(WfResource)

```
public java.util.Collection authorizers(de.danet.an.workflow.omgcore.WfResource
  throws RemoteException;
```

Parameters

`resource`	the resource.
return	a collection of `WfResource` objects, not including `resource`

Exceptions

`RemoteException`	if a system-level error occurs.

Since	1.2

Given a `WfResource object` , return the collection of resources this resource is authorized for.

The resource assignment service usually uses its underlying resource management facility to implement this method, returning all groups the resource is a member of and all roles assigned to the resource. Resource assigments facilities may, however, modify this information e.g. according to configured delegation rules.

If the resource assignment service is based on a resource management service as defined in package `de.danet.an.workflow.spis.rms` , it can simply delegate this call to `ResourceManagementService.authorizers` . The workflow engine does not need this method for its operation; however, it provides a method for accessing this information as part of the client interface to ease the implementation of clients. The implementation of this method by a resource assignment service is therefore optional. If not implemented, a call to this method must result in a <code>UnsupportedOperationException</code>.

A.4.5.5. autoAssignResources(ActivityFinder, String, WfActivity, Principal, Participant)

```
public java.util.Collection autoAssignResources(ActivityFinder finder,
                                                String actId,
                                                de.danet.an.workflow.omgcore.WfA
                                                java.security.Principal principa
                                                de.danet.an.workflow.api.Partici
  throws RemoteException;
```

Parameters

`actId`	a unique (with respect to an `ActivityFinder`) identifier for the Activity. The length of `actId` is guaranteed not to exceed 64.
`finder`	the finder used to lookup activities by their `finderId` s.
`activity`	the activity that is about to become ready.

`principal`	the creator of the process, may be `null` .
`participant`	the `Participant` that describes resource selection criteria. The paramter may be `null`
return	the assigned resources (instances of `WfResource`).

Exceptions

`RemoteException`	if a system-level error occurs.

See Also	`Section A.4.2, “Interface ActivityFinder” [269]`

Triggers the automatic assignment of resources to an activity that is about to become ready.

Usually, criteria for the resource selection must be determined within the resource assignment, e.g. based on the name of the activity, the process it belongs to etc. In some cases, however, the worflow component may have some resource selection information available. The workflow component may have obtained such information e.g. as part of the process description. If such information is available, it may optionally be passed to the automatic assignment. The type and valid values of such information depends totally on the resource assignment service used and remains undefined in the scope of this interface.

A.4.5.6. changeAssignment(ActivityFinder, String, WfActivity, WfResource, WfResource)

```
  public void changeAssignment(ActivityFinder finder,
                               String actId,
                               de.danet.an.workflow.omgcore.WfActivity activ
                               de.danet.an.workflow.omgcore.WfResource oldRe
                               de.danet.an.workflow.omgcore.WfResource newRe
    throws RemoteException, InvalidResourceException, AlreadyAssignedExcepti
```

Parameters

`finder`	the finder used to lookup activities by their `finderId` s
`actId`	a unique (with respect to an `ActivityFinder`) identifier for the Activity. The length of `actId` is guaranteed not to exceed 64
`activity`	the activity being enacted
`oldResource`	the resource that has its assignment removed
`newResource`	the resource to be assigned

Exceptions

`RemoteException`	if a system-level error occurs

`InvalidResourceException`	if the resource is invalid. As the environment is a concurrent multi user environment, `WfResource` objects may become invalid
`AlreadyAssignedException`	if the assignment already exists
`NotAssignedException`	if there is no assignment to the old resource

See Also	`Section A.4.2, “Interface ActivityFinder” [269]`

Change an assignment for enacting an activity. This method is called by the workflow engine in `Activity.changeAssignment` which should be used by resource assignment services to implement `WfAssignment.setAssignee`.

A.4.5.7. getResource(WfAssignment)

```
public de.danet.an.workflow.omgcore.WfResource getResource(de.danet.an.workflow.
  throws RemoteException;
```

Parameters

`asnmnt`	the assignment
return	the resource

Exceptions

`RemoteException`	if a system-level error occurs.

Since	1.3.4

Get the resource associated with an Assignment.

A.4.5.8. isMemberOfWorkItems(WfResource, WfAssignment)

```
public boolean isMemberOfWorkItems(de.danet.an.workflow.omgcore.WfResource resou
                                   de.danet.an.workflow.omgcore.WfAssignment ass
  throws RemoteException, NoSuchResourceException;
```

Parameters

`resource`	the resource.
`assignment`	the assignment in question.
return	`true` if the `assignment` belongs to the work items of the `resource`.

Exceptions

`RemoteException`	if a system-level error occurs.
`NoSuchResourceException`	if the resource is invalid. As the environment is a concurrent multi user environment, `WfResource` objects may become invalid.

Find out if a given assignment belongs to the work items assigned to a particular resource.

A.4.5.9. knownResources()

```
public java.util.Collection knownResources()
  throws RemoteException;
```

Parameters

return	the collection of resources known to the resource assignment service (instances of `WfResource`).

Exceptions

`RemoteException`	if a system-level error occurs.

Returns at least the collection of all the workflow resources being assigned to activities, but should also return the additional workflow resources that are known to the resource assignment service.

If the resource assignment service is based on a resource management service as defined in package `de.danet.an.workflow.spis.rms` , it can simply delegate this call to `ResourceMan-agementService.listResources` . The workflow engine does not need this method for its operation; however, it provides a method for accessing this information as part of the client interface to ease the implementation of clients. The implementation of this method by a resource assignment service is therefore optional. If not implemented, a call to this method must result in a <code>UnsupportedOperationException</code>.

A.4.5.10. removeAssignment(ActivityFinder, String, WfActivity, WfResource)

```
public void removeAssignment(ActivityFinder finder,
                             String actId,
                             de.danet.an.workflow.omgcore.WfActivity activ
                             de.danet.an.workflow.omgcore.WfResource resou
  throws RemoteException, InvalidResourceException, NotAssignedException;
```

Parameters

`finder`	the finder used to lookup activities by their `finderId` s
`actId`	a unique (with respect to an `ActivityFinder`) identifier for the Activity. The length of `actId` is guaranteed not to

	exceed 64.
`activity`	the activity that is about to become ready
`resource`	the resource to be assigned

Exceptions

`RemoteException`	if a system-level error occurs
`InvalidResourceException`	if the resource is invalid. As the environment is a concurrent multi user environment, `WfResource` objects may become invalid.
`NotAssignedException`	if the resource is not assigned to the given activity

See Also	`Section A.4.2, “Interface ActivityFinder” [269]`

Remove the assignment of a resource to an activity. This method is called by the workflow engine in `Activity.removeAssignment` which, in turn, should be used by resource management services to implement `WfResource.release` .

A.4.5.11. resourceByKey(String)

```
public de.danet.an.workflow.omgcore.WfResource resourceByKey(String key)
  throws InvalidKeyException, RemoteException;
```

Parameters

`key`	the key.
return	a `WfResource` object corresponding to the given key.

Exceptions

`InvalidKeyException`	if the resource with the given key can't be found. As the environment is a concurrent multi user environment, `WfResource` objects (and keys obtained from `WfResource` objects) may become invalid.
`RemoteException`	if a system-level error occurs.

Since	1.2

Given the `key` of a `WfResource` (obtained with `resourceKey()`), return the workflow resource associated with this key.

For the workflow core, the resource assignment interface is the only source of `WfResource` objects. While `resourceKey()` provides an easy mapping of those objects to unique keys, the reverse mapping can only be provided by the resource management facility that has created the `WfResource` objects.

As the workflow core does not have a defined access to a resource management facility, this reverse mapping functionality must be brought to the workflow core by the resource assignment service. (Which is quite reasonable, as it has delivered the `WfResource` objects in the first place.) If the resource assignment service is based on a resource management service as defined in package `de.danet.an.workflow.spis.rms` , it can simply delegate this call to `ResourceManagementService.resourceByKey` . The workflow engine does not need this method for its operation; however, it provides a method for accessing this information as part of the client interface to ease the implementation of clients. The implementation of this method by a resource assignment service is therefore optional. If not implemented, a call to this method must result in a <code>UnsupportedOperationException</code>.

A.4.5.12. workItems(WfResource)

```
  public java.util.Collection workItems(de.danet.an.workflow.omgcore.WfResou
    throws RemoteException, NoSuchResourceException;
```

Parameters

`resource`	the resource.
return	the collection of assigned work items (instances of `WfAssignment`).

Exceptions

`RemoteException`	if a system-level error occurs.
`NoSuchResourceException`	if the resource is invalid. As the environment is a concurrent multi user environment, `WfResource` objects may become invalid.

Return the assignments of a given resource.

A.4.6. Class ResourceAssignmentServiceFactory

Defines a factory API that enables the workflow component to obtain a resource assignment facility.

A.4.6.1. Synopsis

```
 public abstract class de.danet.an.workflow.spis.ras.ResourceAssignmentServi
// Protected Constructors

  protected ResourceAssignmentServiceFactory();

// Public Static Methods

  public static ResourceAssignmentServiceFactory newInstance()
    throws FactoryConfigurationError;

// Public Methods

  public abstract ResourceAssignmentService newResourceAssignmentService()
    throws FactoryConfigurationError;
```

```
}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.4.6, “Class ResourceAssignmentServiceFactory” [279]

A.4.6.2. ResourceAssignmentServiceFactory()

```
protected ResourceAssignmentServiceFactory();
```

Constructor. Must be overridden with a parameterless public constructor by derived class.

A.4.6.3. newInstance()

```
public static ResourceAssignmentServiceFactory newInstance()
  throws FactoryConfigurationError;
```

Parameters

return	an instance of the `ResourceAssignmentService-Factory` .

Exceptions

`FactoryConfigurationErr-or`	if a factory instance can't be created.

Obtain a new instance of a `ResourceAssignmentServiceFactory` . This static method creates a new factory instance. The method uses the following ordered lookup procedure to determine the `ResourceAssignmentServiceFactory` implementation class to load:

- If an initial naming context is available, look for a a classname in `java:comp/env/de.danet.an.workflow.spis.ras.ResourceAssignmentServiceFactory` . The configuration for a class as resource assignment service thus looks like:

  ```
  <env-entry>
     <description>Configure the resource assignment factory</description>
     <env-entry-name>de.danet.an.workflow.spis.ras.ResourceAssignmentServiceFac\
  tory</env-entry-name>
     <env-entry-type>java.lang.String</env-entry-type>
     <env-entry-value>
              FactoryImplementationClass
  </env-entry-value>
   </env-entry>
  ```

  Note that this environment entry must be inserted in the `ejb-jar.xml` or `web.xml` for every EJB resp. servlet that calls the `newInstance` method of `ResourceAssignmentServiceFactory` .

- Use the services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file `META-INF/services/`

`de.danet.an.workflow.spis.ras.ResourceAssignmentServiceFactory` . in jars available to the runtime.

A.4.6.4. newResourceAssignmentService()

```
public abstract ResourceAssignmentService newResourceAssignmentService()
  throws FactoryConfigurationError;
```

Parameters

return	the resource assignment service.

Exceptions

`FactoryConfigurationError`	if not all required resources can be obtained.

Creates a new instance of a `ResourceAssignmentService` .

A.5. Package de.danet.an.workflow.spis.rms

This package defines the interface to a resource management service (RMS) as used in the `de.danet.an.workflow.spis.ras` sample implementation. Note that the implementation of the workflow core does not require a resource management service. However, the sample resource assignment service (RAS) needs a resource management service. We have based the sample RAS on this interface to allow a deployer to use the sample RAS with any resource management service by simply writing an adapter.

The service interface follows the standard conventions for a service interface, i.e. it defines an abstract factory class and a service interface.

Implementing this interface implies a particular problem. The RMS must provide objects that implement `WfResource` . This interface has, as specified by OMG, methods `workItems` `isMemberOfWorkItems` and `release` . While it is obvious (from an OO point of view) that those methods are defined as methods of `WfResource` the RMS cannot really implement them. They can only be implemented by an RAS, as it is its task to track assignments. Thus the objects defined and returned by an RMS can only implement the methods by delegating to the RAS.

There may, however, be several instances of RASs in an environment that request resources from the RMS. How can the RMS know, where to delegate to? One way to solve this problem would be to have every RAS "register" itself with the RMS if it assigns a resource from that RMS. This approach has a major drawback. In order to be able to recover from an application restart, the RMS would have to implement such a registry for RASs in persistent store. Depending on the implementation of an RAS, this may be from difficult to impossible.

A much more simple solution is to have a central runtime registry of all available RASs and use the RASs registered there to implement the above mentioned `WfResource` methods (1) . This approach is even more charming as something very close to such a registry exists anyway. While not designed to fulfill this function in the first place, the `ResourceAssignmentServiceFactory` actually implements a kind of registry, as it must know about all RASs in the system (else there is no way that the workflow core uses it for resource assignment anyway).

The `ResourceAssignmentServiceFactory` therefore provides the methods needed to implement `workItems` `isMemberOfWorkItems` and `release` . As a convenience this package provides a class <code>ResourceSupport</code> that uses the methods provided by the `Resour-`

`ceAssignmentServiceFactory` and can be directly used by an RMS as base class for providing its implementation of `WfResource` .

(1)Few things in life are free. There is a small performance penalty to pay for this approach. As the RMS doesn't keep track of the RASs that have requested a particular resource, the central registry has to query all known RASs. However, tracking RAS information (e.g. with identifiers usable as hints for the central registry) in the RMS would impose an overhead as well.

A.5.1. Additional Information

Since	1.0

A.5.2. Class DefaultGroupResource

This class provides a `GroupResource` implementation based on the `BasicResource` . The class ensures that the resource key can be distinguished as a group resource's key.

A.5.2.1. Synopsis

```
 public class de.danet.an.workflow.spis.rms.DefaultGroupResource extends, de.danet
    implements, de.danet.an.workflow.api.GroupResource, java.io.Serializable {
// Public Constructors

  public DefaultGroupResource(ResourceAssignmentContext cbh,
                              String id,
                              String name);

// Public Static Methods

  public static String getId(String key);


  public static boolean isValidKey(String key);

// Public Methods

  public String getId();

}
```

Methods inherited from de.danet.an.workflow.spis.rms.DefaultResource : `equals` , `hashCode` , `isMemberOfWorkItems` , `release` , `resourceKey` , `resourceName` , `workItems`

Methods inherited from java.lang.Object : `clone` , `finalize` , `getClass` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.5.3, “Class DefaultResource” [283] -> Section A.5.2, “Class DefaultGroupResource” [282]

A.5.2.2. DefaultGroupResource(ResourceAssignmentContext, String, String)

```
  public DefaultGroupResource(ResourceAssignmentContext cbh,
                              String id,
                              String name);
```

Parameters

```
cbh
id
name
```

Create a new instance with all attributes initialized to defaults or the given values.

A.5.2.3. getId()

```
  public String getId();
```

Return the id passed to the constructor.

A.5.2.4. getId(String)

```
  public static String getId(String key);
```

Return the id part of a key.

A.5.2.5. isValidKey(String)

```
  public static boolean isValidKey(String key);
```

Check if the given key is a group resource's key.

A.5.3. Class DefaultResource

This class provides a default implementation of the `WfResource` 's methods `workItems` , `isMemberOfWorkItems` and `release` . The implementation is based on the methods of a `ResourceAssignmentContext` passed to the constructor.

A.5.3.1. Synopsis

```
 public class de.danet.an.workflow.spis.rms.DefaultResourceimplements, de.da
// Public Constructors

  public DefaultResource(ResourceAssignmentContext cbh,
                         String key,
                         String name);

// Public Methods

  public boolean equals(Object obj);


  public int hashCode();


  public boolean isMemberOfWorkItems(de.danet.an.workflow.omgcore.WfAssignme
    throws RemoteException, IllegalStateException;


  public void release(de.danet.an.workflow.omgcore.WfAssignment fromAssignme
                      String releaseInfo)
    throws RemoteException, NotAssignedException;


  public String resourceKey()
    throws RemoteException;


  public String resourceName()
```

```
    throws RemoteException;


  public java.util.Collection workItems()
    throws RemoteException, IllegalStateException;

}
```

Direct known subclasses : `de.danet.an.workflow.spis.rms.DefaultGroupResource` , `de.danet.an.workflow.spis.rms.DefaultRoleResource` , `de.danet.an.workflow.spis.rms.DefaultUserResource`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.5.3, “Class DefaultResource” [283]

A.5.3.2. DefaultResource(ResourceAssignmentContext, String, String)

```
  public DefaultResource(ResourceAssignmentContext cbh,
                         String key,
                         String name);
```

Parameters

`assignSvc`	the callback handler
`key`	the resource's key
`name`	the resource's name

The constructor. It ensures that a valid factory exists.

A.5.3.3. isMemberOfWorkItems(WfAssignment)

```
  public boolean isMemberOfWorkItems(de.danet.an.workflow.omgcore.WfAssignment ass
    throws RemoteException, IllegalStateException;
```

Specified by: Method isMemberOfWorkItems in interface WfResource

Parameters

`assignment`	the assignment in question.
return	`true` if the association exists.

Exceptions

`RemoteException`	if a system-level error occurs. This is actually a remapping of the `NoSuchResourceException` thrown by `Resour-ceAssignmentSer-`

	`vice.isMemberOfWorkItems(...)` . It must be remapped because this method's signature is specified by `WfResource.isMemberOfWorkItems(...)` .
`IllegalStateException`	if the resource has become invalid.

Checks if a given `WfAssignment` is associated with this resource.

A.5.3.4. release(WfAssignment, String)

```
  public void release(de.danet.an.workflow.omgcore.WfAssignment fromAssignme
                      String releaseInfo)
    throws RemoteException, NotAssignedException;
```

Specified by: Method release in interface WfResource

Parameters

`fromAssignment`	the specific assignment.
`releaseInfo`	specifies additional information on the reason for realizing the resource as input.

Exceptions

`NotAssignedException`	if the resource is not associated with the given assignment.
`RemoteException`	if a system-level error occurs.

Signals to the resource that it is no longer needed for a specific assignment. The default implementation calls `removeAssignment` on the activity.

A.5.3.5. resourceKey()

```
  public String resourceKey()
    throws RemoteException;
```

Specified by: Method resourceKey in interface WfResource

Parameters

return	key of resource

Exceptions

`RemoteException`	problems accessing resource

Retrieve the key of a resource.

A.5.3.6. resourceName()

```
public String resourceName()
  throws RemoteException;
```

Specified by: Method resourceName in interface WfResource

Parameters

return	name of resource

Exceptions

`RemoteException`	problems accessing resource

Retrieve the name of a resource.

A.5.3.7. workItems()

```
public java.util.Collection workItems()
  throws RemoteException, IllegalStateException;
```

Specified by: Method workItems in interface WfResource

Parameters

return	the associated `WfAssignments` s.

Exceptions

`RemoteException`	if a system-level error occurs.
`IllegalStateException`	if the resource has become invalid. This is actually a remapping of the `NoSuchResourceException` thrown by `ResourceAssignmentService.workItems()` . It must be remapped because this method's signature is specified by `WfResource.workItems()` .

This method returns the `WfAssignments` s associated with a resource.

A.5.4. Class DefaultRoleResource

This class provides a `RoleResource` implementation based on the `BasicResource` . The class ensures that the resource key can be distinguished as a role resource's key.

A.5.4.1. Synopsis

```
public class de.danet.an.workflow.spis.rms.DefaultRoleResource extends, de.danet.
   implements, de.danet.an.workflow.api.RoleResource, java.io.Serializable {
```

```
// Public Constructors

  public DefaultRoleResource(ResourceAssignmentContext cbh,
                             String id,
                             String name);

// Public Static Methods

  public static String getId(String key);


  public static boolean isValidKey(String key);

// Public Methods

  public String getId();

}
```

Methods inherited from de.danet.an.workflow.spis.rms.DefaultResource : `equals` , `hashCode` , `isMemberOfWorkItems` , `release` , `resourceKey` , `resourceName` , `workItems`

Methods inherited from java.lang.Object : `clone` , `finalize` , `getClass` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.5.3, “Class DefaultResource” [283] -> Section A.5.4, “Class DefaultRoleResource” [286]

A.5.4.2. DefaultRoleResource(ResourceAssignmentContext, String, String)

```
  public DefaultRoleResource(ResourceAssignmentContext cbh,
                             String id,
                             String name);
```

Parameters

`cbh`
`id`
`name`

Create a new instance with all attributes initialized to defaults or the given values.

A.5.4.3. getId()

```
  public String getId();
```

Return the id passed to the constructor.

A.5.4.4. getId(String)

```
  public static String getId(String key);
```

Return the id part of a key.

A.5.4.5. isValidKey(String)

```
  public static boolean isValidKey(String key);
```

Check if the given key is a role resource's key.

A.5.5. Class DefaultUserResource

This class provides a `UserResource` implementation based on the `BasicResource` . The class ensures that the resource key can be distinguished as a user resource's key.

A.5.5.1. Synopsis

```
 public class de.danet.an.workflow.spis.rms.DefaultUserResource extends, de.danet.a
    implements, de.danet.an.workflow.api.UserResource, java.io.Serializable {
// Public Constructors

  public DefaultUserResource(ResourceAssignmentContext cbh,
                             String id,
                             String name);

// Public Static Methods

  public static String getId(String key);


  public static boolean isValidKey(String key);

// Public Methods

  public String getId();

}
```

Methods inherited from de.danet.an.workflow.spis.rms.DefaultResource : `equals` , `hashCode` , `isMemberOfWorkItems` , `release` , `resourceKey` , `resourceName` , `workItems`

Methods inherited from java.lang.Object : `clone` , `finalize` , `getClass` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.5.3, “Class DefaultResource” [283] -> Section A.5.5, “Class DefaultUserResource” [288]

A.5.5.2. DefaultUserResource(ResourceAssignmentContext, String, String)

```
  public DefaultUserResource(ResourceAssignmentContext cbh,
                             String id,
                             String name);
```

Parameters

`cbh`
`id`
`name`

Create a new instance with all attributes initialized to defaults or the given values.

A.5.5.3. getId()

```
  public String getId();
```

Return the id passed to the constructor.

A.5.5.4. getId(String)

```
  public static String getId(String key);
```

Return the id part of a key.

A.5.5.5. isValidKey(String)

```
  public static boolean isValidKey(String key);
```

Check if the given key is a user resource's key.

A.5.6. Error FactoryConfigurationError

This exception is thrown by the `newInstance` method of `ResourceManagementService-Factory`.

A.5.6.1. Synopsis

```
 public class de.danet.an.workflow.spis.rms.FactoryConfigurationError extend
// Public Constructors

  public FactoryConfigurationError(Exception cause);


  public FactoryConfigurationError(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Error-> Section A.5.6, “Error FactoryConfigurationError” [289]

A.5.6.2. FactoryConfigurationError(Exception)

```
  public FactoryConfigurationError(Exception cause);
```

Creates a new exception.

A.5.6.3. FactoryConfigurationError(String)

```
  public FactoryConfigurationError(String msg);
```

Parameters

`msg`	the detail message.

Creates a new exception with the given message.

A.5.7. Interface ResourceAssignmentContext

This interface defines the callbacks that are needed by `DefaultResource` to obtain information from the server.

A.5.7.1. Synopsis

```
 public interface de.danet.an.workflow.spis.rms.ResourceAssignmentContext extends,
// Public Methods

  public boolean isMemberOfWorkItems(de.danet.an.workflow.omgcore.WfResource resou
                                     de.danet.an.workflow.omgcore.WfAssignment ass
    throws RemoteException, NoSuchResourceException;


  public java.util.Collection workItems(de.danet.an.workflow.omgcore.WfResource re
    throws RemoteException, NoSuchResourceException;

}
```

Inheritance Path. Section A.5.7, “Interface ResourceAssignmentContext” [290]

A.5.7.2. isMemberOfWorkItems(WfResource, WfAssignment)

```
  public boolean isMemberOfWorkItems(de.danet.an.workflow.omgcore.WfResource resou
                                     de.danet.an.workflow.omgcore.WfAssignment ass
    throws RemoteException, NoSuchResourceException;
```

Parameters

`resource`	the resource.
`assignment`	the assignment in question.
return	`true` if the `assignment` belongs to the work items of the `resource`.

Exceptions

`RemoteException`	if a system-level error occurs.
`NoSuchResourceException`	if the resource is invalid. As the environment is a concurrent multi user environment, `WfResource` objects may become invalid.

Find out if a given assignment belongs to the work items assigned to a particular resource.

A.5.7.3. workItems(WfResource)

```
  public java.util.Collection workItems(de.danet.an.workflow.omgcore.WfResource re
    throws RemoteException, NoSuchResourceException;
```

Parameters

`resource`	the resource.
return	the collection of assigned work items (instances of `WfAssignment`).

Exceptions

`RemoteException`	if a system-level error occurs.
`NoSuchResourceException`	if the resource is invalid. As the environment is a concurrent multi user environment, `WfResource` objects may become invalid.

Return the assignments of a given resource.

A.5.8. Interface ResourceManagementService

This interface defines the workflow resource management service used in the workflow package.

A.5.8.1. Synopsis

```
 public interface de.danet.an.workflow.spis.rms.ResourceManagementService {
// Public Methods

  public de.danet.an.workflow.omgcore.WfResource asResource(java.security.Pr
    throws ResourceNotFoundException, RemoteException;


  public java.util.Collection authorizers(de.danet.an.workflow.omgcore.WfRes
    throws RemoteException;


  public java.util.Collection listResources()
    throws RemoteException;


  public de.danet.an.workflow.omgcore.WfResource resourceByKey(String key)
    throws ResourceNotFoundException, RemoteException;


  public java.util.Collection selectResources(Object resSel)
    throws RemoteException, UnsupportedOperationException;

}
```

Inheritance Path. Section A.5.8, “Interface ResourceManagementService” [291]

A.5.8.2. asResource(Principal)

```
  public de.danet.an.workflow.omgcore.WfResource asResource(java.security.Pr
    throws ResourceNotFoundException, RemoteException;
```

Parameters

`principal`	the principal.
return	a `WfResource` object corresponding to the given principal.

Exceptions

`ResourceNotFoundExcep-tion`	if the StaffMember with the given key can't be found or the key is not associate with an StaffMember object.
`RemoteException`	if a system-level error occurs.

Given a principal, return the workflow resource associated with this principal by the resource management facility.

This method is usually used to get a `WfResource` object corresponding to the current user. The `WfResource` object can subsequently be used to e.g. determine the current user's worklist.

A.5.8.3. authorizers(WfResource)

```
public java.util.Collection authorizers(de.danet.an.workflow.omgcore.WfResource
  throws RemoteException;
```

Parameters

`resource`	the resource.
return	a collection of `WfResource` objects, not including `re-source`

Exceptions

`RemoteException`	if a system-level error occurs.

Given a `WfResource object` , return the collection of resources this resource is authorized for.

This method usually returns all groups the resource is a member of and all roles assigned to the resource.

A.5.8.4. listResources()

```
public java.util.Collection listResources()
  throws RemoteException;
```

Parameters

return	collection of `WfResource` objects.

Exceptions

`RemoteException`	if a system-level error occurs.

List all available resources.

A.5.8.5. resourceByKey(String)

```
public de.danet.an.workflow.omgcore.WfResource resourceByKey(String key)
  throws ResourceNotFoundException, RemoteException;
```

Parameters

`key`	the key.
return	a `WfResource` object corresponding to the given key.

Exceptions

`ResourceNotFoundException`	if the StaffMember or an StaffGroup with the given key can't be found.
`RemoteException`	if a system-level error occurs.

Given a `key` , return the workflow resource associated with this key.

This method is usually used to get a `WfResource` object corresponding to the given key.

A.5.8.6. selectResources(Object)

```
public java.util.Collection selectResources(Object resSel)
  throws RemoteException, UnsupportedOperationException;
```

Parameters

`resSel`	an object that describes resource selection criteria.
return	collection of `WfResource` objects.

Exceptions

`RemoteException`	if a system-level error occurs.
`UnsupportedOperationException`	if the resource management service does not support this feature.

This optional method selects resources based on the resource selection criteria passed as parameter.

Usually, criteria for the resource selection must be determined within the resource assignment, based on the list of resources obtained with `listResources` . Implementations of resource management facilities may, however, support some query functionality that eases this task for the resource assignment service. The resource assignment service may have received such resource selection information from the workflow engine via `autoAssignResources` (the workflow component has obtained the information probably as part of the process description and passed it through transparently).

A.5.9. Class ResourceManagementServiceFactory

Defines a factory API that enables a workflow component to obtain a workflow resource management service.

A.5.9.1. Synopsis

```
 public abstract class de.danet.an.workflow.spis.rms.ResourceManagementServiceFact
// Protected Constructors

  protected ResourceManagementServiceFactory();

// Public Static Methods

  public static ResourceManagementServiceFactory newInstance()
    throws FactoryConfigurationError;

// Public Methods

  public ResourceAssignmentContext getResourceAssignmentContext();


  public de.danet.an.workflow.spis.ras.ResourceAssignmentService getResourceAssign


  public abstract ResourceManagementService newResourceManagementService()
    throws FactoryConfigurationError;


  public void setResourceAssignmentContext(ResourceAssignmentContext resourceAssig


  public void setResourceAssignmentService(de.danet.an.workflow.spis.ras.ResourceA

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.5.9, “Class ResourceManagementServiceFactory” [294]

A.5.9.2. ResourceManagementServiceFactory()

```
  protected ResourceManagementServiceFactory();
```

Constructor. Must be overridden with a parameterless public constructor by derived class.

A.5.9.3. getResourceAssignmentContext()

```
  public ResourceAssignmentContext getResourceAssignmentContext();
```

Parameters

return	Returns the configured resource assignment context.

See Also	`setResourceAssignmentContext(de.danet.an.workflow.spis.rms.ResourceAssignmentContext) [296]`

A.5.9.4. getResourceAssignmentService()

```
public de.danet.an.workflow.spis.ras.ResourceAssignmentService getResource
```

Parameters

return	the resource assignment service.

See Also	`setResourceAssignmentService(de.danet.an.workflow.spis.ras.ResourceAssignmentService) [297]`

Deprecated

see `setResourceAssignmentService`

Return the resource assignment service set with `setResourceAssignmentService` .

A.5.9.5. newInstance()

```
public static ResourceManagementServiceFactory newInstance()
  throws FactoryConfigurationError;
```

Parameters

return	an instance of the `ResourceManagementServiceFactory` .

Exceptions

`FactoryConfigurationError`	if a factory instance can't be created.

Obtain a new instance of a `ResourceManagementServiceFactory` . This static method creates a new factory instance . The method uses the following ordered lookup procedure to determine the `ResourceManagementServiceFactory` implementation class to load:

- If an initial naming context is available, look for a a classname in `java:comp/env/de.danet.an.workflow.spis.rms.ResourceManagementServiceFactory` . The configuration for a class as resource management service thus looks like:

```
<env-entry> <description>Configure the resource management
factory</description>
<env-entry-name>de.danet.an.workflow.spis.ras.ResourceManagementServiceFac
```

```
ry</env-entry-name>
 <env-entry-type>java.lang.String</env-entry-type>
 <env-entry-value>
              FactoryImplementationClass
</env-entry-value>
 </env-entry> <env-entry>
```

Note that this environment entry must be inserted in the `ejb-jar.xml` or `web.xml` for every EJB resp. servlet that calls the `newInstance` method of `ResourceManagementServiceFactory`.

- Use the services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file `META-INF/services/de.danet.an.workflow.spis.rms.ResourceManagementServiceFactory`. in jars available to the runtime.

A.5.9.6. newResourceManagementService()

```
public abstract ResourceManagementService newResourceManagementService()
  throws FactoryConfigurationError;
```

Parameters

return	the resource management service.

Exceptions

`FactoryConfigurationError`	if a service instance can't be created.

Creates a new instance of a `ResourceManagementService`.

A.5.9.7. setResourceAssignmentContext(ResourceAssignmentContext)

```
public void setResourceAssignmentContext(ResourceAssignmentContext resourceAssig
```

Parameters

`resourceAssignmentContext`	The resourceAssignmentContext to set.

Specifies the resource assignment service to be used by the instances of `ResourceManagementService` subsequently created (by calling `newResourceManagementService()`).

A resource management service needs a reference to a resource assignment context to be able to implement `WfResource` objects. E.g. the method `workItems` requires that a `WfAssignment` be returned, something that can only be done in cooperation with the resource assignment service.

A.5.9.8. setResourceAssignmentService(ResourceAssignmentService)

```
  public void setResourceAssignmentService(de.danet.an.workflow.spis.ras.Res
```

Parameters

`service`	the resource management service.
See Also	`getResourceAssignmentService() [295]`

Deprecated

the resource management service needs only a small subset of the methods provided by a `ResourceAssignmentService` . To allow greater flexibility when implementing a resource management service, the requirement for a complete resource assignment service has been replaced with the requirement for a `ResourceAssignmentContext` .

Specifies the resource assignment service to be used by the instances of `ResourceManagementService` subsequently created (by calling `newResourceManagementService()`).

A resource management service needs a reference to a resource assignment service to be able to implement `WfResource` objects. The method `workItems` requires that a `WfAssignment` be returned, something that can only be done in cooperation with the resource assignment service.

An implementation problem may arise from the requirement to set a `ResourceAssignmentService` as implementations of `ResourceAssignmentService` that in turn rely on a `ResourceManagementService` may need a `ResourceManagementService` in order to be instantiated. Such implementations of `ResourceAssignmentService` have therefore to delay the actual request for a `ResourceManagementService` instance until they have made themselves known to the `ResourceManagementServiceFactory` .

A.5.10. Exception ResourceNotFoundException

This exception is thrown by the resource service implementation if no resource for a given key can be found.

A.5.10.1. Synopsis

```
 public class de.danet.an.workflow.spis.rms.ResourceNotFoundException extend
    implements, java.io.Serializable {
// Public Constructors

  public ResourceNotFoundException();


  public ResourceNotFoundException(String msg);

}
```

Methods inherited from java.lang.Throwable : `fillInStackTrace` , `getCause` , `getLocalizedMessage` , `getMessage` , `getStackTrace` , `initCause` , `printStackTrace` , `setStackTrace` , `toString`

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `wait`

Inheritance Path. java.lang.Object-> java.lang.Throwable-> java.lang.Exception-> Section A.5.10, “Exception ResourceNotFoundException” [297]

A.5.10.2. ResourceNotFoundException()

```
public ResourceNotFoundException();
```

Creates a new exception.

A.5.10.3. ResourceNotFoundException(String)

```
public ResourceNotFoundException(String msg);
```

Parameters

`msg`	the detail message.

Creates a new exception with the given message.

A.6. Package de.danet.an.workflow.tools.util

This package provides support classes for the implementation of tool agents and applications. Unlike the `core API` and the `extended API` (including the `application invocation interface`) this package is *not* intended to be part of a general Java workflow API. Rather, it provides some WfMOpen specific utilities that ease the implementation of tool agents and applications to be used with the WfMOpen workflow engine.

Using the asynchronous tool invocation API typically requires that the activity and the data passed to the tool agent as parameters are saved in persistent store. The tool (application) that is to actually work with the information typically runs completely independent of the workflow engine in another operating system process or at least in another thread. It accesses the persisted information and e.g. presents the tasks that require human interaction in a GUI. When the interaction has completed, the application retrieves the activity and calls " `setResult()` " and " `complete()` " on it.

To support the implementation of the function described above, this package provides a <code>SimpleApplicationDirectory</code>. This class implements a general purpose persistent store for the data passed to the tool agent (i.e. the activity and the actual parameters) and additional state information. Tool agents create ("register") instances in this directory. Each instance represents a tool invocation (task) currently running in an external application. Creating an entry results in a unique id that may subsequently be used to retrieve, update or remove the instance. Instances may also be retrieved using an arbitrary key associated with the instance by the registering tool agent, or by activity or by searching instances assigned to a particular `WfResource` .

Tool agents using the directory should be derived from `SimpleApplicationAgent` . They can access the directory by simply invoking the method `applicationDirectory()` . External application should use the workflow service and `SimpleApplicationDirectoryLookup` to access the directory. Applications that run within the application server (or components running within the applictaion server that act on behalf of the external application) may obtain the application directory's local interface by looking it up in JNDI. The name to use for the lookup depends on your application configuration, see section "Application and client assembly", subsection "Workflow module" in the user manual.

Note that the directory may also be used to persist application state across several tool agent invocations by `associating` an existing instance with a new activity. An example of this requirement is the timer tool. One tool agent creates a timer in a conceptually independently running application and gets an id for it. Another tool agent may then cancel this time or wait for it to expire. I.e. the task initiated by the first tool agent in the application runs spans several tool agent invocations.

Entries in the directory are automatically cleaned when the process they relate to (i.e. the activity's container) is removed.

A.6.1. Additional Information

Since	V1.2

A.6.2. Interface DirectInvocable

This interface marks a tool agent as requiring no or only read-only access to the activity passed to `invoke` .

The issue arises from the necessity to invoke the tool agent in its own transaction. In order to provide the tool agent with full access to the activity, the activity must not be involved in another transaction. However, as tool agent invocation obviously updates the state of the activity, the activity (having its state updated) is involved in a transaction when the tool agent is invoked. To resolve this, WfMOpen does not invoke the tool agent directly. Rather, its sends a message to a tool agent invocation queue as part of the state update transaction and completes the transaction. The message is then retrieved from the queue and the tool agent is invoked without involving the activity in the message handling transaction.

Of course, this induces a considerable overhead which can be avoided if the tool agent accesses the activity not at all or uses only the methods `key` , `activityUniqueKey` or `container` . Experience shows that this is true for a lot of tool agents; either because they let an application running in another thread or process do the work, or because they use the `ResultProvider` interface and do not require access to the activity. If a tool agent satisfies these criteria, it may make this known to the workflow engine by implementing this marker interface. If a tool agent implements this interface, the engine does not put a message on the tool invocation queue but rather invokes the tool agent directly during state update.

Note that since the activity is involved in the state update transaction, the engine cannot call `setResult` and `complete` in a new transaction after tool agent invocation. Rather, these methods will be called in the same transaction as the tool agent invocation. Thus if `RemoteException` s occur when these methods are called by the workflow engine, the complete transaction, including the tool invocation, will be repeated. Tools that are "expensive" to execute or have side effects should not implement `DirectInvocable` .

A.6.2.1. Synopsis

```
 public interface de.danet.an.workflow.tools.util.DirectInvocable {
}
```

Inheritance Path. Section A.6.2, “Interface DirectInvocable” [299]

A.6.3. Class SimpleApplicationAgent

This is a base class for writing agents for applications that have a life cycle exceeding a single tool agent invocation, and have only state information that can efficiently be persisted as a binary large object in an RDBMs.

An example for such an application is the wait tool (see *User Manual*). A wait tool (or "wait application") is created by one tool agent invocation. Then another tool agent invocation waits for the completion of the wait, while yet another tool agent invocation may cause the wait tool to complete prematurely. The wait tool has state that is controlled by the tool agents and a timer provided by the enviroment. Its state is therefore completely serializable (no open sockets, tool controlled threads etc.).

This base class mainly provides access to the `application directory` that does all the important work.

A.6.3.1. Synopsis

```
 public abstract class de.danet.an.workflow.tools.util.SimpleApplicationAgentimple
// Public Constructors

  public SimpleApplicationAgent();

// Public Methods

  public void terminate(de.danet.an.workflow.api.Activity activity)
    throws ApplicationNotStoppedException, RemoteException;

// Protected Methods

  protected SimpleApplicationDirectoryLocal applicationDirectory()
    throws ResourceNotAvailableException;

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.6.3, “Class SimpleApplicationAgent” [299]

A.6.3.2. SimpleApplicationAgent()

```
  public SimpleApplicationAgent();
```

Creates an instance of `SimpleApplicationAgent` with all attributes initialized to default values.

A.6.3.3. applicationDirectory()

```
  protected SimpleApplicationDirectoryLocal applicationDirectory()
    throws ResourceNotAvailableException;
```

Parameters

return	the application directory

Exceptions

`ResourceNotAvailableException`	if an application directory EJB cannot be instantiated

Return the application directory.

A.6.3.4. terminate(Activity)

```
  public void terminate(de.danet.an.workflow.api.Activity activity)
    throws ApplicationNotStoppedException, RemoteException;
```

Specified by: Method terminate in interface ToolAgent

Parameters

`activity`	the activity to be canceled

Exceptions

`ApplicationNotStoppedException`	if execution cannot be terminated (see `ApplicationNotStoppedException`). workflow engine should retry the tool invocation
`RemoteException`	if a temporary problem occurs and the workflow engine should retry the tool invocation

Terminates execution of the given activity. This base implementation simply removes the application instance fom the application directory.

A.6.4. Interface SimpleApplicationDirectory

Remote interface for SimpleApplicationDirectory.

A.6.4.1. Synopsis

```
 public interface de.danet.an.workflow.tools.util.SimpleApplicationDirectory
// Public Methods

  public SimpleApplicationInfo infoByActivity(de.danet.an.workflow.api.Activ
    throws InvalidKeyException, RemoteException;


  public java.util.Collection infosByApplication(String applName)
    throws RemoteException;


  public java.util.Collection infosByKey(String applName,
                                         String applInstKey)
    throws InvalidKeyException, RemoteException;


  public java.util.Collection infosByResource(String applName,
                                              String resourceKey)
    throws RemoteException;


  public SimpleApplicationInfo instanceInfo(long instId)
    throws InvalidKeyException, RemoteException;


  public long registerInstance(String applName,
                               de.danet.an.workflow.api.Activity activity,
                               Object state,
                               boolean saveAssignment)
    throws RemoteException;


  public long registerInstance(String applName,
                               String applInstKey,
                               de.danet.an.workflow.api.Activity activity,
                                Object state,
                                boolean saveAssignment)
    throws RemoteException;


  public void removeInstance(long instId)
    throws RemoteException;


  public void updateInvokingActivity(long instId,
                                      de.danet.an.workflow.api.ActivityUniqueKey au
    throws InvalidKeyException, RemoteException;


  public void updateResourceKey(long instId,
                                 String resourceKey)
    throws InvalidKeyException, RemoteException;


  public void updateState(long instId,
                           Object state)
    throws InvalidKeyException, RemoteException;

}
```

See Also	de.danet.an.workflow.util
Xdoclet-generated	at April 14 2009

Inheritance Path. Section A.6.4, “Interface SimpleApplicationDirectory” [301]

A.6.4.2. infoByActivity(ActivityUniqueKey)

```
  public SimpleApplicationInfo infoByActivity(de.danet.an.workflow.api.ActivityUni
    throws InvalidKeyException, RemoteException;
```

Parameters

`auk`	the unique key of the activity an application instance is expected to be registered for.
return	the info

Exceptions

`InvalidKeyException`	if there is no data available for the given activity

Return the information associated with the activity.

A.6.4.3. infosByApplication(String)

```
  public java.util.Collection infosByApplication(String applName)
    throws RemoteException;
```

Parameters

`applName`	the application name
return	the infos as collection

Return infos associated with a given application.

A.6.4.4. infosByKey(String, String)

```
public java.util.Collection infosByKey(String applName,
                                       String applInstKey)
  throws InvalidKeyException, RemoteException;
```

Parameters

`applName`	the application name
`applInstKey`	the key associated with the instance
return	the infos or an empty collection if no infos with the given application name and key exist

Return the infos associated with the given application name and key.

A.6.4.5. infosByResource(String, String)

```
public java.util.Collection infosByResource(String applName,
                                            String resourceKey)
  throws RemoteException;
```

Parameters

`applName`	the application name
`resourceKey`	the resource's key
return	the infos as collection

Return infos associated with a given application and resource.

A.6.4.6. instanceInfo(long)

```
public SimpleApplicationInfo instanceInfo(long instId)
  throws InvalidKeyException, RemoteException;
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`

return	the info

Exceptions

`InvalidKeyException`	if there is no data available for the given id

Return the information associated with the application instance.

A.6.4.7. registerInstance(String, Activity, Object, boolean)

```
public long registerInstance(String applName,
                             de.danet.an.workflow.api.Activity activity,
                             Object state,
                             boolean saveAssignment)
  throws RemoteException;
```

Parameters

`applName`	the application name
`activity`	the invoking activity
`state`	the application state
`saveAssignment`	if `true` the assigned resource will be saved to allow searching application instances with a particular assignee
return	the instance id

Register a new application instance. This method returns a unique key that can be used to `retrieve` , `update` and `removeInstance(long) [305]` remove the instance information.

A.6.4.8. registerInstance(String, String, Activity, Object, boolean)

```
public long registerInstance(String applName,
                             String applInstKey,
                             de.danet.an.workflow.api.Activity activity,
                             Object state,
                             boolean saveAssignment)
  throws RemoteException;
```

Parameters

`applName`	the application name
`applInstKey`	an arbitrary key for this instance, up to 1000 characters long
`activity`	the invoking activity
`state`	the application state
`saveAssignment`	if `true` the assigned resource will be saved to allow searching application instances with a particular assignee

return	the instance id

Register a new application instance. This method returns a unique key that may be used to

A.6.4.9. removeInstance(long)

```
public void removeInstance(long instId)
  throws RemoteException;
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`

Remove an application instance.

A.6.4.10. updateInvokingActivity(long, ActivityUniqueKey)

```
public void updateInvokingActivity(long instId,
                                   de.danet.an.workflow.api.ActivityUnique
  throws InvalidKeyException, RemoteException;
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`
`auk`	the new activity's unique key. May be `null` if the application instance is temporarily not associated with an activity.

Exceptions

`InvalidKeyException`	if there is no application instance with the given id

Update the activity associated with the given application instance. This is useful if an application instance is started by one tool (agent) invocation and stopped by another.

Be careful to ensure the eventual termination of the application. If the creating activity has completed, the terminate method of the tool agent that started the application will not be called on abnormal process completion. So, if a process is terminated abnormally and the starting activity is closed and the stopping activity has not yet been started (and associated with the application) the application will not be stopped. This should normally not be a problem for simple applications.

As a convenience, any application information that is still registered after a process completion will automatically be deleted.

The new activity must belong to the same process as the activity that initially created the application instance.

A.6.4.11. updateResourceKey(long, String)

```
public void updateResourceKey(long instId,
                                String resourceKey)
    throws InvalidKeyException, RemoteException;
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`
`resourceKey`	the associated resource

Exceptions

`InvalidKeyException`	if there is no application instance with the given id

Update the resource associated with the given application instance id.

A.6.4.12. updateState(long, Object)

```
  public void updateState(long instId,
                          Object state)
    throws InvalidKeyException, RemoteException;
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`
`state`	the new state

Exceptions

`InvalidKeyException`	if there is no application instance with the given id

Update the state information associated with the given application instance id.

A.6.5. Class SimpleApplicationDirectoryEJB

This EJB provides a directory for simple applications. Applications are considered simple in this context if their state can efficiently be represented (and stored) using a single serializable object.

This directory maps a unique id to the activity unique key information of the executing activity and an associated state (and vice versa). Optionally, the assignment time and an assigned resource may be saved in this directory as well.

A.6.5.1. Synopsis

```
 public class de.danet.an.workflow.tools.util.SimpleApplicationDirectoryEJBimpleme
// Public Constructors

  public SimpleApplicationDirectoryEJB();
```

```
// Public Methods

  public void ejbActivate()
    throws EJBException;


  public void ejbCreate()
    throws CreateException;


  public void ejbPassivate()
    throws EJBException;


  public void ejbRemove();


  public SimpleApplicationInfo infoByActivity(de.danet.an.workflow.api.Activ
    throws InvalidKeyException;


  public java.util.Collection infosByApplication(String applName);


  public java.util.Collection infosByKey(String applName,
                                         String applInstKey)
    throws InvalidKeyException;


  public java.util.Collection infosByResource(String applName,
                                              String resourceKey);


  public SimpleApplicationInfo instanceInfo(long instId)
    throws InvalidKeyException;


  public long registerInstance(String applName,
                               de.danet.an.workflow.api.Activity activity,
                               Object state,
                               boolean saveAssignment);


  public long registerInstance(String applName,
                               String applInstKey,
                               de.danet.an.workflow.api.Activity activity,
                               Object state,
                               boolean saveAssignment);


  public void removeInstance(long instId);


  public void setSessionContext(javax.ejb.SessionContext context);


  public void updateInvokingActivity(long instId,
                                     de.danet.an.workflow.api.ActivityUnique
    throws InvalidKeyException;


  public void updateResourceKey(long instId,
                                String resourceKey)
    throws InvalidKeyException;


  public void updateState(long instId,
                          Object state)
    throws InvalidKeyException;
}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

See Also	de.danet.an.workflow.util
Ejb.bean	name="SimpleApplicationDirectory" jndi-name="ejb/@@@_JNDI_Name_Prefix_@@@SimpleApplicationDirectory" local-jndi-name="ejb/@@@_JNDI_Name_Prefix_@@@SimpleApplicationDirectoryLocal" display-name="Simple Application Directory EJB" type="Stateless" transaction-type="Container" view-type="both"
Jonas.bean	ejb-name="SimpleApplicationDirectory"
Ejb.transaction	type="Required"
Ejb.permission	role-name="WfMOpenAdmin"
Ejb.ejb-external-ref	ref-name="ejb/JdbcKeyGenLocal" link="KeyGen" type="Session" view-type="local" home="de.danet.an.util.KeyGenLocalHome" business="de.danet.an.util.KeyGenLocal"
Ejb.resource-ref	res-ref-name="jdbc/WfEngine" res-type="javax.sql.DataSource" res-auth="Container"
Jonas.resource	res-ref-name="jdbc/WfEngine" jndi-name="jdbc_1"
Weblogic.enable-call-by-reference	True
Weblogic.resource-description	res-ref-name="jdbc/WfEngine" jndi-name="DefaultDS"
Weblogic.transaction-isolation	TRANSACTION_READ_COMMITTED

Inheritance Path. java.lang.Object-> Section A.6.5, “Class SimpleApplicationDirectoryEJB” [306]

A.6.5.2. SimpleApplicationDirectoryEJB()

```
public SimpleApplicationDirectoryEJB();
```

Creates an instance of `SimpleApplicationDirectoryEJB` with all attributes initialized to default values.

A.6.5.3. ejbActivate()

```
public void ejbActivate()
  throws EJBException;
```

Specified by: Method `ejbActivate` in interface `SessionBean`

See Also	javax.ejb.SessionBean

Not called for stateless session beans.

A.6.5.4. ejbCreate()

```
public void ejbCreate()
  throws CreateException;
```

Exceptions

`CreateException`	if creation fails

Create a new EJB.

A.6.5.5. ejbPassivate()

```
public void ejbPassivate()
  throws EJBException;
```

Specified by: Method `ejbPassivate` in interface `SessionBean`

See Also	javax.ejb.SessionBean

Not called for stateless session beans.

A.6.5.6. ejbRemove()

```
public void ejbRemove();
```

Specified by: Method `ejbRemove` in interface `SessionBean`

Remove this EJB.

A.6.5.7. infoByActivity(ActivityUniqueKey)

```
public SimpleApplicationInfo infoByActivity(de.danet.an.workflow.api.Activ
  throws InvalidKeyException;
```

Parameters

`auk`	the unique key of the activity an application instance is expected to be registered for.
return	the info

Exceptions

`InvalidKeyException`	if there is no data available for the given activity

Ejb.interface-method	view-type="both"

Return the information associated with the activity.

A.6.5.8. infosByApplication(String)

```
public java.util.Collection infosByApplication(String applName);
```

Parameters

`applName`	the application name
return	the infos as collection

Ejb.interface-method	view-type="both"

Return infos associated with a given application.

A.6.5.9. infosByKey(String, String)

```
public java.util.Collection infosByKey(String applName,
                                       String applInstKey)
  throws InvalidKeyException;
```

Parameters

`applName`	the application name
`applInstKey`	the key associated with the instance
return	the infos or an empty collection if no infos with the given application name and key exist

Ejb.interface-method	view-type="both"

Return the infos associated with the given application name and key.

A.6.5.10. infosByResource(String, String)

```
public java.util.Collection infosByResource(String applName,
                                            String resourceKey);
```

Parameters

`applName`	the application name
`resourceKey`	the resource's key
return	the infos as collection

Ejb.interface-method	view-type="both"

Return infos associated with a given application and resource.

A.6.5.11. instanceInfo(long)

```
public SimpleApplicationInfo instanceInfo(long instId)
  throws InvalidKeyException;
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`
return	the info

Exceptions

`InvalidKeyException`	if there is no data available for the given id

Ejb.interface-method	view-type="both"

Return the information associated with the application instance.

A.6.5.12. registerInstance(String, Activity, Object, boolean)

```
public long registerInstance(String applName,
                             de.danet.an.workflow.api.Activity activity,
                             Object state,
                             boolean saveAssignment);
```

Parameters

`applName`	the application name
`activity`	the invoking activity
`state`	the application state
`saveAssignment`	if `true` the assigned resource will be saved to allow searching application instances with a particular assignee
return	the instance id

Ejb.interface-method	view-type="both"

Register a new application instance. This method returns a unique key that can be used to `retrieve` , `update` and `removeInstance(long) [312]` remove the instance information.

A.6.5.13. registerInstance(String, String, Activity, Object, boolean)

```
public long registerInstance(String applName,
                             String applInstKey,
                             de.danet.an.workflow.api.Activity activity,
                             Object state,
                             boolean saveAssignment);
```

Parameters

`applName`	the application name
`applInstKey`	an arbitrary key for this instance, up to 1000 characters long
`activity`	the invoking activity
`state`	the application state
`saveAssignment`	if `true` the assigned resource will be saved to allow searching application instances with a particular assignee
return	the instance id

Ejb.interface-method	view-type="both"

Register a new application instance. This method returns a unique key that may be used to

A.6.5.14. removeInstance(long)

```
public void removeInstance(long instId);
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`

Ejb.interface-method	view-type="both"

Remove an application instance.

A.6.5.15. setSessionContext(SessionContext)

```
public void setSessionContext(javax.ejb.SessionContext context);
```

Specified by: Method `setSessionContext` in interface `SessionBean`

Parameters

`context`	the context

Save the session context asigned by the container.

A.6.5.16. updateInvokingActivity(long, ActivityUniqueKey)

```
  public void updateInvokingActivity(long instId,
                                    de.danet.an.workflow.api.ActivityUniqueK
    throws InvalidKeyException;
```

Parameters

`instId`	the application instance id previously assigned by `regis-terInstance`
`auk`	the new activity's unique key. May be `null` if the application instance is temporarily not associated with an activity.

Exceptions

`InvalidKeyException`	if there is no application instance with the given id

Ejb.interface-method	view-type="both"

Update the activity associated with the given application instance. This is useful if an application instance is started by one tool (agent) invocation and stopped by another.

Be careful to ensure the eventual termination of the application. If the creating activity has completed, the terminate method of the tool agent that started the application will not be called on abnormal process completion. So, if a process is terminated abnormally and the starting activity is closed and the stopping activity has not yet been started (and associated with the application) the application will not be stopped. This should normally not be a problem for simple applications.

As a convenience, any application information that is still registered after a process completion will automatically be deleted.

The new activity must belong to the same process as the activity that initially created the application instance.

A.6.5.17. updateResourceKey(long, String)

```
  public void updateResourceKey(long instId,
                               String resourceKey)
    throws InvalidKeyException;
```

Parameters

`instId`	the application instance id previously assigned by `regis-terInstance`
`resourceKey`	the associated resource

Exceptions

`InvalidKeyException`	if there is no application instance with the given id

Ejb.interface-method	view-type="both"

Update the resource associated with the given application instance id.

A.6.5.18. updateState(long, Object)

```
  public void updateState(long instId,
                          Object state)
    throws InvalidKeyException;
```

Parameters

`instId`	the application instance id previously assigned by `regis-terInstance`
`state`	the new state

Exceptions

`InvalidKeyException`	if there is no application instance with the given id

Ejb.interface-method	view-type="both"

Update the state information associated with the given application instance id.

A.6.6. Interface SimpleApplicationDirectoryHome

Home interface for SimpleApplicationDirectory.

A.6.6.1. Synopsis

```
 public interface de.danet.an.workflow.tools.util.SimpleApplicationDirectoryHome e
// Public Static Fields

  public static final String COMP_NAME  = java:comp/env/ejb/SimpleApplicationDirec


  public static final String JNDI_NAME  = ejb/@@@_JNDI_Name_Prefix_@@@SimpleApplic

// Public Methods

  public SimpleApplicationDirectory create()
    throws CreateException, RemoteException;

}
```

See Also	de.danet.an.workflow.util
Xdoclet-generated	at April 14 2009

Inheritance Path. Section A.6.6, “Interface SimpleApplicationDirectoryHome” [314]

A.6.7. Interface SimpleApplicationDirectoryLocal

Local interface for SimpleApplicationDirectory.

A.6.7.1. Synopsis

```
 public interface de.danet.an.workflow.tools.util.SimpleApplicationDirectory
// Public Methods

  public SimpleApplicationInfo infoByActivity(de.danet.an.workflow.api.Activ
    throws InvalidKeyException;


  public java.util.Collection infosByApplication(String applName);


  public java.util.Collection infosByKey(String applName,
                                         String applInstKey)
    throws InvalidKeyException;


  public java.util.Collection infosByResource(String applName,
                                              String resourceKey);


  public SimpleApplicationInfo instanceInfo(long instId)
    throws InvalidKeyException;


  public long registerInstance(String applName,
                               de.danet.an.workflow.api.Activity activity,
                               Object state,
                               boolean saveAssignment);


  public long registerInstance(String applName,
                               String applInstKey,
                               de.danet.an.workflow.api.Activity activity,
                               Object state,
                               boolean saveAssignment);


  public void removeInstance(long instId);


  public void updateInvokingActivity(long instId,
                                     de.danet.an.workflow.api.ActivityUnique
    throws InvalidKeyException;


  public void updateResourceKey(long instId,
                                String resourceKey)
    throws InvalidKeyException;


  public void updateState(long instId,
                          Object state)
    throws InvalidKeyException;

}
```

See Also	de.danet.an.workflow.util

Xdoclet-generated	at April 14 2009

Inheritance Path. Section A.6.7, “Interface SimpleApplicationDirectoryLocal” [315]

A.6.7.2. infoByActivity(ActivityUniqueKey)

```
  public SimpleApplicationInfo infoByActivity(de.danet.an.workflow.api.ActivityUni
    throws InvalidKeyException;
```

Parameters

`auk`	the unique key of the activity an application instance is expected to be registered for.
return	the info

Exceptions

`InvalidKeyException`	if there is no data available for the given activity

Return the information associated with the activity.

A.6.7.3. infosByApplication(String)

```
  public java.util.Collection infosByApplication(String applName);
```

Parameters

`applName`	the application name
return	the infos as collection

Return infos associated with a given application.

A.6.7.4. infosByKey(String, String)

```
  public java.util.Collection infosByKey(String applName,
                                         String applInstKey)
    throws InvalidKeyException;
```

Parameters

`applName`	the application name
`applInstKey`	the key associated with the instance
return	the infos or an empty collection if no infos with the given application name and key exist

Return the infos associated with the given application name and key.

A.6.7.5. infosByResource(String, String)

```
public java.util.Collection infosByResource(String applName,
                                            String resourceKey);
```

Parameters

`applName`	the application name
`resourceKey`	the resource's key
return	the infos as collection

Return infos associated with a given application and resource.

A.6.7.6. instanceInfo(long)

```
public SimpleApplicationInfo instanceInfo(long instId)
  throws InvalidKeyException;
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`
return	the info

Exceptions

`InvalidKeyException`	if there is no data available for the given id

Return the information associated with the application instance.

A.6.7.7. registerInstance(String, Activity, Object, boolean)

```
public long registerInstance(String applName,
                             de.danet.an.workflow.api.Activity activity,
                             Object state,
                             boolean saveAssignment);
```

Parameters

`applName`	the application name
`activity`	the invoking activity
`state`	the application state

`saveAssignment`	if `true` the assigned resource will be saved to allow searching application instances with a particular assignee
return	the instance id

Register a new application instance. This method returns a unique key that can be used to `retrieve` , `update` and `removeInstance(long) [318]` remove the instance information.

A.6.7.8. registerInstance(String, String, Activity, Object, boolean)

```
public long registerInstance(String applName,
                             String applInstKey,
                             de.danet.an.workflow.api.Activity activity,
                             Object state,
                             boolean saveAssignment);
```

Parameters

`applName`	the application name
`applInstKey`	an arbitrary key for this instance, up to 1000 characters long
`activity`	the invoking activity
`state`	the application state
`saveAssignment`	if `true` the assigned resource will be saved to allow searching application instances with a particular assignee
return	the instance id

Register a new application instance. This method returns a unique key that may be used to

A.6.7.9. removeInstance(long)

```
public void removeInstance(long instId);
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`

Remove an application instance.

A.6.7.10. updateInvokingActivity(long, ActivityUniqueKey)

```
public void updateInvokingActivity(long instId,
                                   de.danet.an.workflow.api.ActivityUniqueKey au
  throws InvalidKeyException;
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`
`auk`	the new activity's unique key. May be `null` if the application instance is temporarily not associated with an activity.

Exceptions

`InvalidKeyException`	if there is no application instance with the given id

Update the activity associated with the given application instance. This is useful if an application instance is started by one tool (agent) invocation and stopped by another.

Be careful to ensure the eventual termination of the application. If the creating activity has completed, the terminate method of the tool agent that started the application will not be called on abnormal process completion. So, if a process is terminated abnormally and the starting activity is closed and the stopping activity has not yet been started (and associated with the application) the application will not be stopped. This should normally not be a problem for simple applications.

As a convenience, any application information that is still registered after a process completion will automatically be deleted.

The new activity must belong to the same process as the activity that initially created the application instance.

A.6.7.11. updateResourceKey(long, String)

```
public void updateResourceKey(long instId,
                              String resourceKey)
  throws InvalidKeyException;
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`
`resourceKey`	the associated resource

Exceptions

`InvalidKeyException`	if there is no application instance with the given id

Update the resource associated with the given application instance id.

A.6.7.12. updateState(long, Object)

```
public void updateState(long instId,
                        Object state)
  throws InvalidKeyException;
```

Parameters

`instId`	the application instance id previously assigned by `registerInstance`
`state`	the new state

Exceptions

`InvalidKeyException`	if there is no application instance with the given id

Update the state information associated with the given application instance id.

A.6.8. Interface SimpleApplicationDirectoryLocalHome

Local home interface for SimpleApplicationDirectory.

A.6.8.1. Synopsis

```
 public interface de.danet.an.workflow.tools.util.SimpleApplicationDirectoryLocalH
// Public Static Fields

  public static final String COMP_NAME  = java:comp/env/ejb/SimpleApplicationDirec


  public static final String JNDI_NAME  = ejb/@@@_JNDI_Name_Prefix_@@@SimpleApplic

// Public Methods

  public SimpleApplicationDirectoryLocal create()
    throws CreateException;

}
```

See Also	de.danet.an.workflow.util
Xdoclet-generated	at April 14 2009

Inheritance Path. Section A.6.8, “Interface SimpleApplicationDirectoryLocalHome” [320]

A.6.9. Class SimpleApplicationDirectoryLookup

This class provides a `Batch` implementation that looks up and returns the remote interface of the `SimpleApplicationDirectory` .

A.6.9.1. Synopsis

```
 public class de.danet.an.workflow.tools.util.SimpleApplicationDirectoryLookupimpl
// Public Constructors

  public SimpleApplicationDirectoryLookup();

// Public Methods

  public Object execute(de.danet.an.workflow.api.Batch.Context ctx)
    throws InvocationTargetException;

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.6.9, “Class SimpleApplicationDirectoryLookup” [320]

A.6.9.2. execute(Batch.Context)

```
  public Object execute(de.danet.an.workflow.api.Batch.Context ctx)
    throws InvocationTargetException;
```

Specified by: Method execute in interface Batch

Parameters

`ctx`	the execution context
return	the result as defined by the implementing class

Exceptions

`InvocationTargetException`	wraps exceptions as defined by the implementing class

Description copied from interface: <link linkend="METHOD-DE.DANET.AN.WORKFLOW.API.BATCH.EXECUTE-DE.DANET.AN.WORKFLOW.API.BATCH.CONTEXT-">execute</link>

Execute the batch.

A.6.10. Class SimpleApplicationInfo

This class provides a container for the application instance information managed by the `SimpleApplicationDirectory` .

A.6.10.1. Synopsis

```
 public class de.danet.an.workflow.tools.util.SimpleApplicationInfoimplement
// Public Methods

  public de.danet.an.workflow.api.ActivityUniqueKey activityUniqueKey();


  public java.util.Date assignedAt();


  public long id();


  public String resourceKey();


  public Object state();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> Section A.6.10, “Class SimpleApplicationInfo” [321]

A.6.10.2. activityUniqueKey()

```
public de.danet.an.workflow.api.ActivityUniqueKey activityUniqueKey();
```

Parameters

return	activity's unique key

Return the activity unique key.

A.6.10.3. assignedAt()

```
public java.util.Date assignedAt();
```

Parameters

return	assignment timestamp

Return the assignment timestamp.

A.6.10.4. id()

```
public long id();
```

Parameters

return	instance id

Return the application instance id.

A.6.10.5. resourceKey()

```
public String resourceKey();
```

Parameters

return	assigned resource key

Return the assigned resource.

A.6.10.6. state()

```
public Object state();
```

Parameters

return	application state

Return the application state.

Appendix B. The service provider classes

B.1. Package de.danet.an.workflow.ejbs.client

This package contains client side helper classes used to access the EJBs.

B.1.1. Additional Information

Since	V1.0

B.1.2. Class StandardWorkflowServiceFactory

Implements the workflow service factory.

Usage of this class as service factory requires an additional configuration parameter. The service factory implementation needs to connect to the workflow engine EJB. In order to do so, it needs a JNDI name to look up the engine's home interface. As JNDI names must be changeable by the application deployer, the name can't be hard coded.

This factory therefore uses the following ordered lookup procedure to determine the JNDI name of the workflow engine EJB home interface:

- Look for a property " `de.danet.an.workflow.engine` " among the properties set for this factory.
- Look for a name in `java:comp/env/de.danet.an.workflow.engine` . The configuration for the `StandardWorkflowServiceFactory` using this mechanism thus looks like:

  ```
  <env-entry>
     <description>Configure the chosen factory</description>
     <env-entry-name>de.danet.an.workflow.engine</env-entry-name>
     <env-entry-type>java.lang.String</env-entry-type>
     <env-entry-value>
            JNDI name of workflow engine EJB home
  </env-entry-value>
   </env-entry>
  ```

  Note that this environment entry must be inserted in the `ejb-jar.xml` or `web.xml` for every EJB resp. servlet that calls the <code>newInstance</code> method of `WorkflowServiceFactory` .
- Find the application resource file `de.danet.an.workflow-wfs.properties` and look for an entry " `engine = JNDI name of workflow engine EJB home` ".

B.1.2.1. Synopsis

```
 public class de.danet.an.workflow.ejbs.client.StandardWorkflowServiceFactory
// Public Constructors

  public StandardWorkflowServiceFactory();
```

```
// Public Methods

  public WorkflowService newWorkflowService()
    throws FactoryConfigurationError;

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> WorkflowServiceFactory-> de.danet.an.workflow.ejbs.client.StandardWorkflowServiceFactory [325]

B.1.2.2. StandardWorkflowServiceFactory()

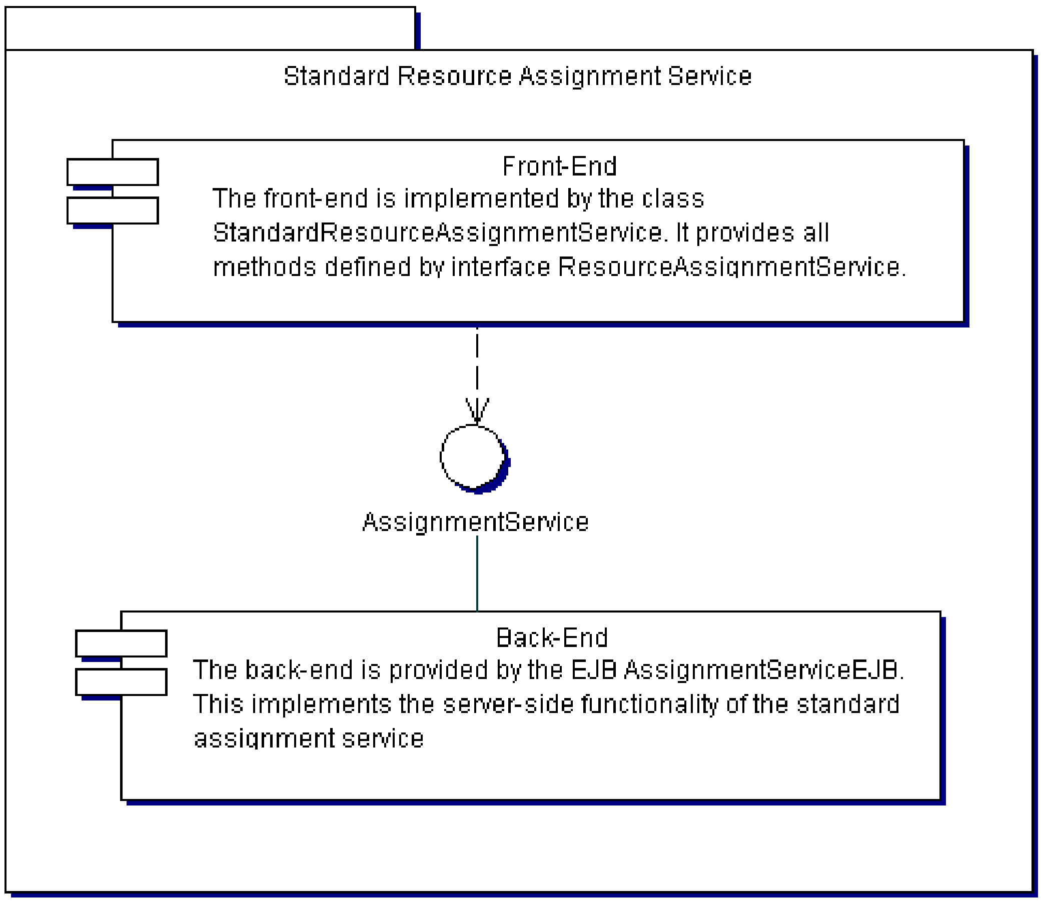
```
  public StandardWorkflowServiceFactory();
```

Creates an instance of `StandardWorkflowServiceFactory` and looks up the associated EJB.

B.2. Package de.danet.an.workflow.assignment

This package provides a simple resource assignment service as specified by `de.danet.an.workflow.spis.ras` . It is independant of a specific resource management service, as it uses only the service API defined by `de.danet.an.workflow.rms` to access a resource management system.

The service is implemented by the class `StandardResourceAssignmentService` . It acts as a front-end (client) for the server-side component (back-end). The server-side component is provided by the `AssignmentServiceEJB` .


Standard Resource Assignment Service
Front-End
The front-end is implemented by the class StandardResourceAssignmentService. It provides all methods defined by interface ResourceAssignmentService.
AssignmentService
Back-End
The back-end is provided by the EJB AssignmentServiceEJB. This implements the server-side functionality of the standard assignment service

B.2.1. Additional Information

Since	1.0

B.2.2. Class StandardResourceAssignmentService-Factory

Implements a simple resource assignment service factory.

This implementation uses an instance of <code>ResourceManagementService</code> to access a resource management facility. To obtain the service it calls the <code>newInstance</code> method of <code>ResourceManagementServiceFactory</code>. Thus if this factory (`StandardResourceAssignmentServiceFactory`) is configured as resource assignment service factory, all configuration information required by `ResourceManagementServiceFactory.newInstance` (and by the actually configured resource management service factory implementation) must be available when the <code>newInstance</code> method of `ResourceAssignmentServiceFactory` is called.

B.2.2.1. Synopsis

```
 public class de.danet.an.workflow.assignment.StandardResourceAssignmentServ
// Public Constructors

  public StandardResourceAssignmentServiceFactory()
```

```
    throws FactoryConfigurationError;

// Public Methods

  public boolean equals(Object obj);

  public int hashCode();

  public ResourceAssignmentService newResourceAssignmentService()
    throws FactoryConfigurationError;

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> ResourceAssignmentServiceFactory-> de.danet.an.workflow.assignment.StandardResourceAssignmentServiceFactory [327]

B.2.2.2. StandardResourceAssignmentServiceFactory()

```
  public StandardResourceAssignmentServiceFactory()
    throws FactoryConfigurationError;
```

Exceptions

`FactoryConfigurationError`	if the required resources cannot be obtained.

Constructor.

B.2.2.3. equals(Object)

```
  public boolean equals(Object obj);
```

Parameters

`obj`	the factory to compare with.
return	`true` if the objects are equal.

Two resource assignment service factories are equal if they are identically configured.

B.2.2.4. hashCode()

```
  public int hashCode();
```

Parameters

return	the hash code.

Generate a hash code.

B.3. Package de.danet.an.workflow.rmsimpls.dbrms

B.3.1. Class DatabaseRmsFactory

This class provides an implementaion of the factory API based on information obtained from a database.

Usage of this class as service factory requires several configuration parameters for accessing the database. The factory uses the following ordered lookup procedure to determine these parameters:

- Look for parameters in JNDI. Lookup parameters in `java:comp/env/de.danet.an.workflow.rmsimpls.dbrms.parameter` . The configuration for a parameter using this mechanism thus looks like:

  ```
   <env-entry>
     <description>Configure database parameter
              parameter
  </description>
     <env-entry-name>de.danet.an.workflow.rmsimpls.dbrms.
              parameter
  </env-entry-name>
     <env-entry-type>java.lang.String</env-entry-type>
     <env-entry-value>
              Value of parameter
  </env-entry-value>
   </env-entry>
  ```

  Note that the environment entries must be inserted in the `ejb-jar.xml` or `web.xml` for every EJB resp. servlet that calls the <code>newInstance</code> method of `ResourceManagementServiceFactory` .

- Find the application resource file `de.danet.an.workflow.rmsimpls.dbrms-factory.properties` and look for entries " `parameter = Value of parameter` ".

The following parameters are used:

`dataSource`	The lookup name of the data source in JNDI (e.g. `java:/DefaultDS`).
`allUsersQuery`	The query that returns the known users as a two column result with the user's entry's primary key in the first column and the user's display name in the second.
`allRolesQuery`	The query that returns the known roles as a two column result with the role's entry's primary key in the first column and the role's display name in the second.
`allGroupsQuery`	The query that returns the known groups as a two column result with the group's entry's primary key in the first column and group's display name in the second.
`userNameQuery`	The query that returns user data as a single column result with the user's display name in the only column. The user's entry's

	primary key is set as query parameter.
`roleNameQuery`	The query that returns role data as a single column result with the role's display name in the only column. The role's entry's primary key is set as query parameter.
`groupNameQuery`	The query that returns group data as a single column result with the group's display name in the only column. The group's entry's primary key is set as query parameter.
`userLookupQuery`	The query that returns a user's data as a two column result with the user's entry's primary key in the first column and the user's display name in the second. The user's account name (usually the login name) is set as query parameter.
`roleLookupQuery`	The query that returns a role's data as a two column result with the role's entry's primary key in the first column and the role's display name in the second. The role's "account" name (clear text name) is set as query parameter.
`groupLookupQuery`	The query that returns a group's data as a two column result with the group's entry's primary key in the first column and the group's name display in the second. The groups's "account" name (clear text name) is set as query parameter.
`rolesQuery`	The query that returns a user's roles as a two column result with the role's entry's primary key in the first column and the role's display name in the second. The user's entry's primary key is set as query parameter.
`groupsQuery`	The query that returns a user's groups as a two column result with the group's entry's primary key in the first column and the group's display name in the second. The user's entry's primary key is set as query parameter.

B.3.1.1. Synopsis

```
 public class de.danet.an.workflow.rmsimpls.dbrms.DatabaseRmsFactory extends, Reso
// Public Constructors

  public DatabaseRmsFactory()
    throws FactoryConfigurationError;

// Public Methods

  public ResourceManagementService newResourceManagementService()
    throws FactoryConfigurationError;

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> ResourceManagementServiceFactory-> de.danet.an.workflow.rmsimpls.dbrms.DatabaseRmsFactory [329]

B.3.1.2. DatabaseRmsFactory()

```
  public DatabaseRmsFactory()
    throws FactoryConfigurationError;
```

Exceptions

`FactoryConfigurationError`	if configuration information is missing or invalid.

Create a new instance of this factory.

B.4. Package de.danet.an.workflow.rmsimpls.eisrms

B.4.1. Class EisRmsFactory

This class provides an implementaion of the factory API based on information obtained from an RMS resource adapter.

The resource adapter must provide the `RmsConnectionFactory` . The factory is obtained from JNDI using the logical name `java:comp/env/ra/RMS` . This is the only configuration option of this resource management service factory, as anything else is configured using properties of the resource adapter.

B.4.1.1. Synopsis

```
 public class de.danet.an.workflow.rmsimpls.eisrms.EisRmsFactory extends, Re
// Public Constructors

  public EisRmsFactory()
    throws FactoryConfigurationError;

// Public Methods

  public ResourceManagementService newResourceManagementService()
    throws FactoryConfigurationError;

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> ResourceManagementServiceFactory-> de.danet.an.workflow.rmsimpls.eisrms.EisRmsFactory [331]

B.4.1.2. EisRmsFactory()

```
  public EisRmsFactory()
    throws FactoryConfigurationError;
```

Exceptions

`FactoryConfigurationError`	if configuration information is missing or invalid.

Create a new instance of this factory.

B.5. Package

de.danet.an.workflow.rmsimpls.eisrms.aci

This package defines the client API of the resource adapter modules for the EIS RMS.

B.5.1. Additional Information

Since	V1.4

B.5.2. Interface RmsConnection

This interface defines a connection to a resource management system accessed via JCA.

B.5.2.1. Synopsis

```
 public interface de.danet.an.workflow.rmsimpls.eisrms.aci.RmsConnection extends,
// Public Methods

  public java.util.Collection authorizers(String key)
    throws ResourceException;


  public void close()
    throws ResourceException;


  public java.util.Collection listResources()
    throws ResourceException;


  public RmsEntry lookupResource(String key)
    throws ResourceException, NameNotFoundException;


  public RmsEntry lookupUserByAccountName(String name)
    throws ResourceException, NameNotFoundException;


  public java.util.Collection selectResources(Object resSel)
    throws ResourceException;

}
```

Inheritance Path. de.danet.an.workflow.rmsimpls.eisrms.aci.RmsConnection [332]

B.5.2.2. authorizers(String)

```
  public java.util.Collection authorizers(String key)
    throws ResourceException;
```

Parameters

`key`	the resource's key
return	a collection of `RmsEntry` objects, not including the resource identified by the key

Exceptions

`ResourceException`	if an error occurs

Given a resource's key, return the collection of resource entries this resource is authorized for.

This method usually returns all groups a user is a member of and all roles assigned to a user.

B.5.2.3. close()

```
public void close()
  throws ResourceException;
```

Close the connection.

B.5.2.4. listResources()

```
public java.util.Collection listResources()
  throws ResourceException;
```

Exceptions

`ResourceException`	if an error occurs

List all available resources.

B.5.2.5. lookupResource(String)

```
public RmsEntry lookupResource(String key)
  throws ResourceException, NameNotFoundException;
```

Parameters

`type`	the key
`name`	the name

Find a resource given its key.

B.5.2.6. lookupUserByAccountName(String)

```
public RmsEntry lookupUserByAccountName(String name)
  throws ResourceException, NameNotFoundException;
```

Parameters

`name`	the name

Find a user given its account (login) name.

B.5.2.7. selectResources(Object)

```
  public java.util.Collection selectResources(Object resSel)
    throws ResourceException;
```

Parameters

`resSel`	an object that describes resource selection criteria
return	collection of `RmsEntry` objects

Exceptions

`UnsupportedOperationException`	if the resource management service does not support this feature.
`ResourceException`	if an error occurs

This optional method selects resources based on the resource selection criteria passed as parameter.

Usually, criteria for the resource selection must be determined within the resource assignment, based on the list of resources obtained with `listResources` . Implementations of resource management facilities may, however, support some query functionality that eases this task for the resource assignment service. The resource assignment service may have received such resource selection information from the workflow engine via <code>autoAssignResources</code> (the workflow component has obtained the information propably as part of the process description and passed it through transparently).

B.5.3. Interface RmsConnectionFactory

This interface defines a factory for connections to a resource management system accessed via JCA.

B.5.3.1. Synopsis

```
 public interface de.danet.an.workflow.rmsimpls.eisrms.aci.RmsConnectionFactory ex
// Public Methods

  public RmsConnection getConnection()
    throws ResourceException;

}
```

Inheritance Path. de.danet.an.workflow.rmsimpls.eisrms.aci.RmsConnectionFactory [334]

B.5.4. Class RmsEntry

This class describes a resource.

B.5.4.1. Synopsis

```
 public final class de.danet.an.workflow.rmsimpls.eisrms.aci.RmsEntryimplements, j
// Public Static Fields

  public static final int RESOURCE_TYPE_GROUP  = 2;


  public static final int RESOURCE_TYPE_ROLE  = 3;
  public static final int RESOURCE_TYPE_USER  = 1;

// Public Constructors

  public RmsEntry(int type,
                  String key,
                  String displayName);

// Public Methods

  public String getDisplayName();


  public String getKey();


  public int getType();

}
```

Methods inherited from java.lang.Object : `clone` , `equals` , `finalize` , `getClass` , `hashCode` , `notify` , `notifyAll` , `toString` , `wait`

Inheritance Path. java.lang.Object-> de.danet.an.workflow.rmsimpls.eisrms.aci.RmsEntry [334]

B.5.4.2. RmsEntry(int, String, String)

```
  public RmsEntry(int type,
                  String key,
                  String displayName);
```

Parameters

```
type
key
name
```

B.5.4.3. getDisplayName()

```
  public String getDisplayName();
```

Parameters

return	Returns the name.

B.5.4.4. getKey()

```
  public String getKey();
```

Parameters

return	Returns the key.

B.5.4.5. getType()

```
public int getType();
```

Parameters

return	Returns the type.

Appendix C. The demo applications

To support new users in getting acquainted with Danet's workflow component, we provide applications that bundle the workflow component with a sample portal. The portal includes the management portlets and the XForms tool on pre-configured pages.

Obviously, there are various configuration options for such a sample application, beginning with the choice of the portal. These options would soon lead to an unmanageable number of EAR files in the distribution. To avoid this, we have provided an installer that configures and deployes a demo application of your choice. It does this using ant build files that may serve as a good starting point for your own application. If you're interested, have a look at the sources (or unpack the installer, and look at the `demos` subfolder).

C.1. Installing a demo application

The installation consists of the following steps:

- Download and unpack JBoss 4.0.4.GA or later.
- Create a JBoss server configuration for the demo.
- Create the datasource for the demo, initialize the database and create and deploy the demo.

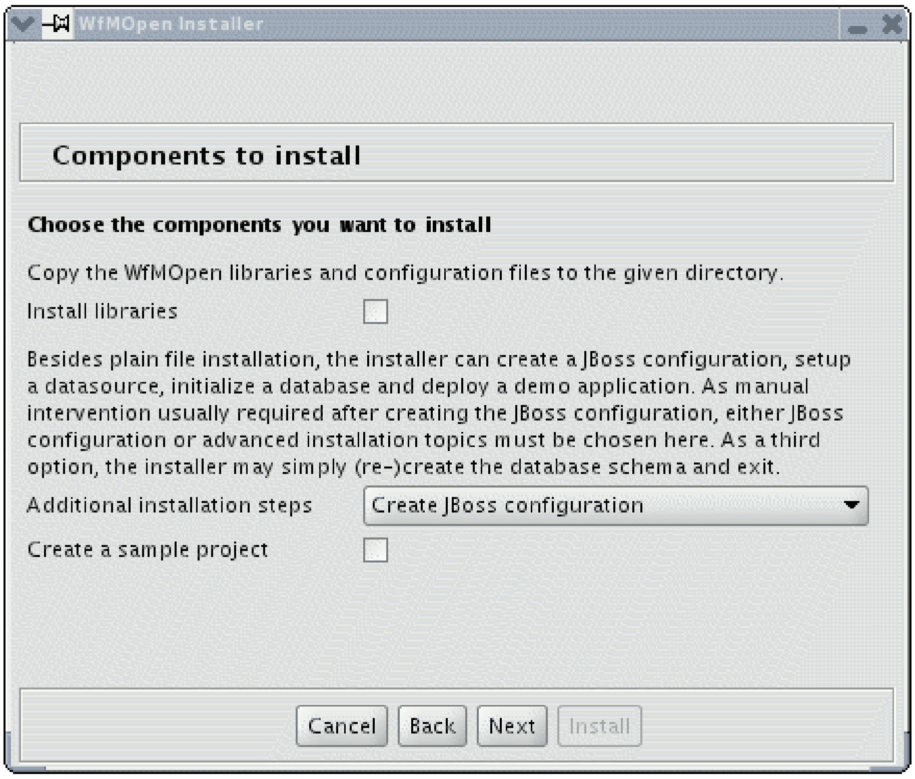
C.1.1. Installing JBoss

There isn't much to add to the title. Download the binaries and unpack them. The root directory of the unpacked binaries (usually something like `jboss-`*`n.m.o`*) will be referred to as `$JBOSS_HOME` below.

C.1.2. Create JBoss server configuration

A JBoss server can have several configurations. As distributed, it has the predefined configuration "default", "minimal" and "all". We follow the good practice of creating our own configuration. This leaves the JBoss distribution intact and you don't have to unpack a complete JBoss for every application environment that you want to experiment with.

The server configuration can be created by the installer. The installer is distributed as an executable jar. It is invoked as "java -jar `wfmopen-`*`n.m`*`-installer.jar`". To create the server configuration, select "Create JBoss configuration" and proceed.



Components to install

Choose the components you want to install

Copy the WfMOpen libraries and configuration files to the given directory.

Install libraries ☐

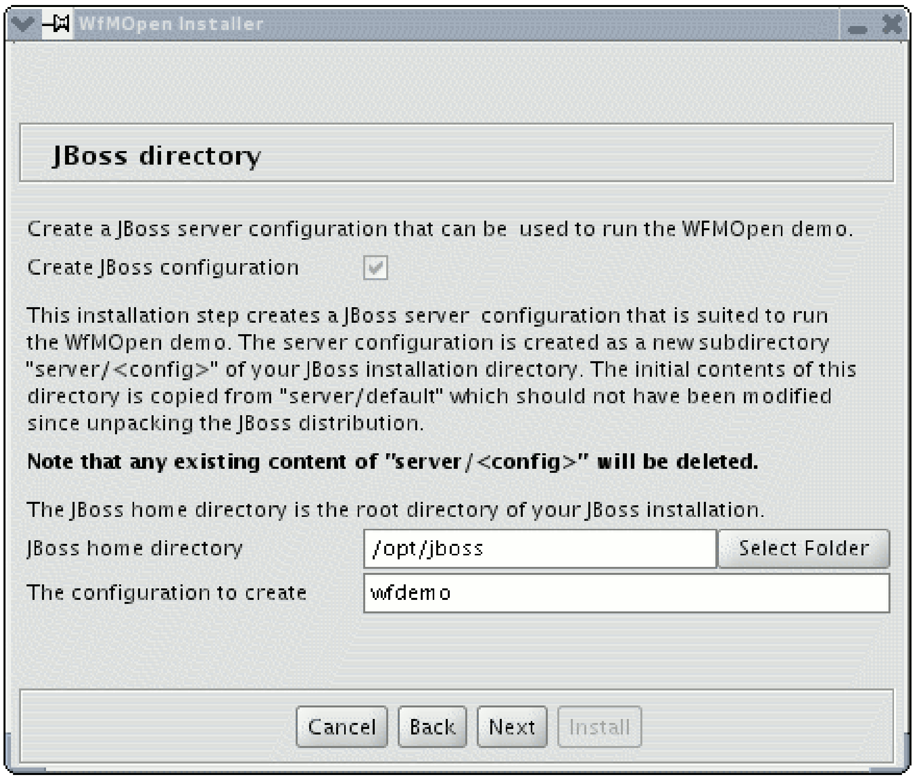
Besides plain file installation, the installer can create a JBoss configuration, setup a datasource, initialize a database and deploy a demo application. As manual intervention usually required after creating the JBoss configuration, either JBoss configuration or advanced installation topics must be chosen here. As a third option, the installer may simply (re-)create the database schema and exit.

Additional installation steps: Create JBoss configuration

Create a sample project ☐

Figure C.1. Create JBoss configuration

The next screen prompts you for your JBoss installation directory, i.e. `$JBOSS_HOME`, and the configuration you want to create. The new configuration is created in `$JBOSS_HOME/server/`*`configuration`* as a copy of `$JBOSS_HOME/server/default`. The default configuration should therefore not have been modified since unpacking JBoss.



JBoss directory

Create a JBoss server configuration that can be used to run the WFMOpen demo.

Create JBoss configuration ✓

This installation step creates a JBoss server configuration that is suited to run the WfMOpen demo. The server configuration is created as a new subdirectory "server/<config>" of your JBoss installation directory. The initial contents of this directory is copied from "server/default" which should not have been modified since unpacking the JBoss distribution.

Note that any existing content of "server/<config>" will be deleted.

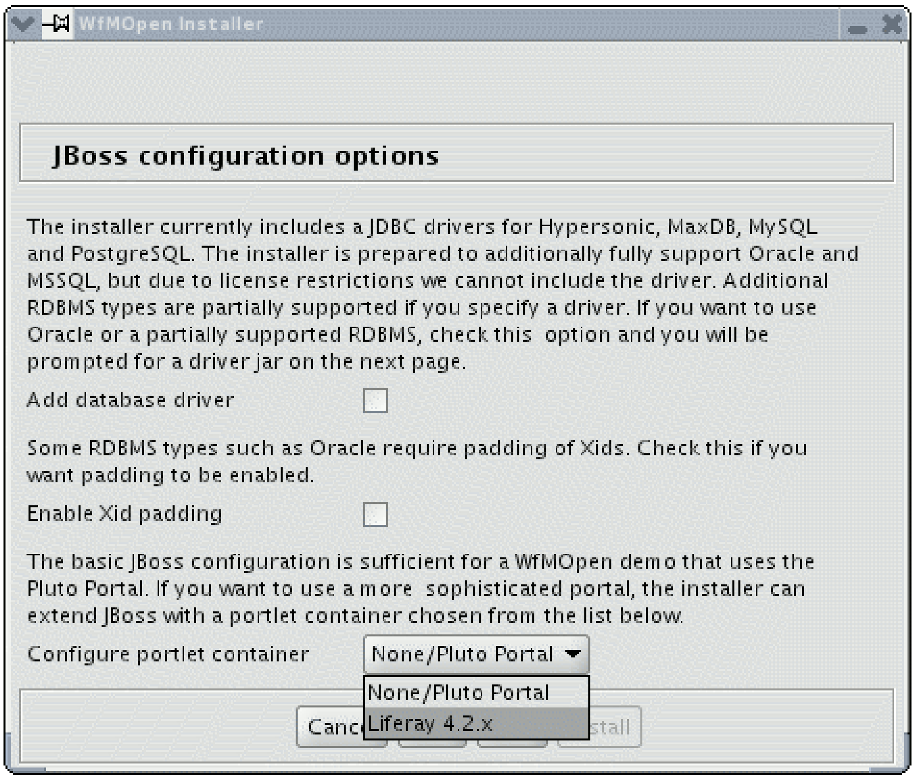
The JBoss home directory is the root directory of your JBoss installation.

JBoss home directory: /opt/jboss

The configuration to create: wfdemo

Figure C.2. JBoss installation directory

In addition to copying the default configuration, the installer can add database drivers, enable Xid padding (required for e.g. Oracle) and add a sample Liferay portal to the new JBoss configuration.



JBoss configuration options

The installer currently includes a JDBC drivers for Hypersonic, MaxDB, MySQL and PostgreSQL. The installer is prepared to additionally fully support Oracle and MSSQL, but due to license restrictions we cannot include the driver. Additional RDBMS types are partially supported if you specify a driver. If you want to use Oracle or a partially supported RDBMS, check this option and you will be prompted for a driver jar on the next page.

Add database driver ☐

Some RDBMS types such as Oracle require padding of Xids. Check this if you want padding to be enabled.

Enable Xid padding ☐

The basic JBoss configuration is sufficient for a WfMOpen demo that uses the Pluto Portal. If you want to use a more sophisticated portal, the installer can extend JBoss with a portlet container chosen from the list below.

Configure portlet container: None/Pluto Portal

Figure C.3. Liferay configuration options

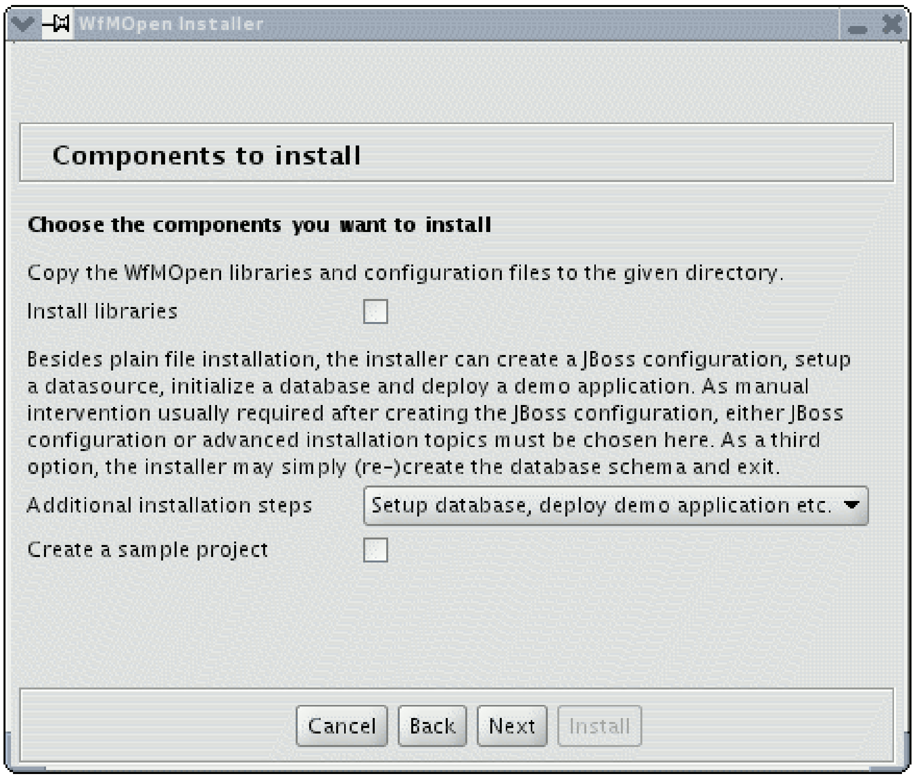
If you choose to add Liferay to the new JBoss configuration you will be prompted for a directory with some files from the Liferay distribution. You have to download these files yourself from the Liferay distribution site http://sourceforge.net/project/showfiles.php?group_id=49260&package_id=42607 as they are not bundled with the installer.

Eventually, the installer provides somes hints about how to start the new JBoss configuration, creates the configuration and exists. You must now start JBoss with the newly created configuration and, if the output indicates no errors, restart the installer.

The reason why the installation is done in two steps is that the second step needs a running JBoss server.

C.1.3. Creating datasources etc.

Restart the installer and proceed with "Setup database ...".


WfMOpen Installer

Components to install

Choose the components you want to install

Copy the WfMOpen libraries and configuration files to the given directory.

Install libraries ☐

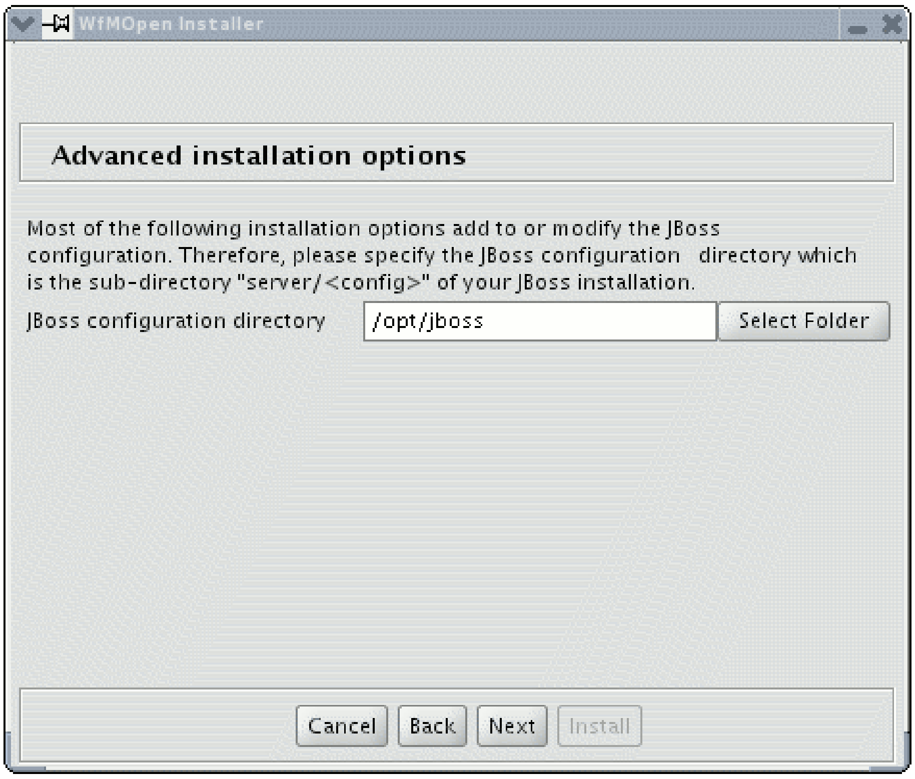
Besides plain file installation, the installer can create a JBoss configuration, setup a datasource, initialize a database and deploy a demo application. As manual intervention usually required after creating the JBoss configuration, either JBoss configuration or advanced installation topics must be chosen here. As a third option, the installer may simply (re-)create the database schema and exit.

Additional installation steps: Setup database, deploy demo application etc.

Create a sample project ☐

Figure C.4. Setup wfdemo configuration

You'll have to enter your `$JBOSS_HOME/server/`*`configuration`* directory once more on the next screen, as the installer cannot currently remember this setting from the first step.


WfMOpen Installer

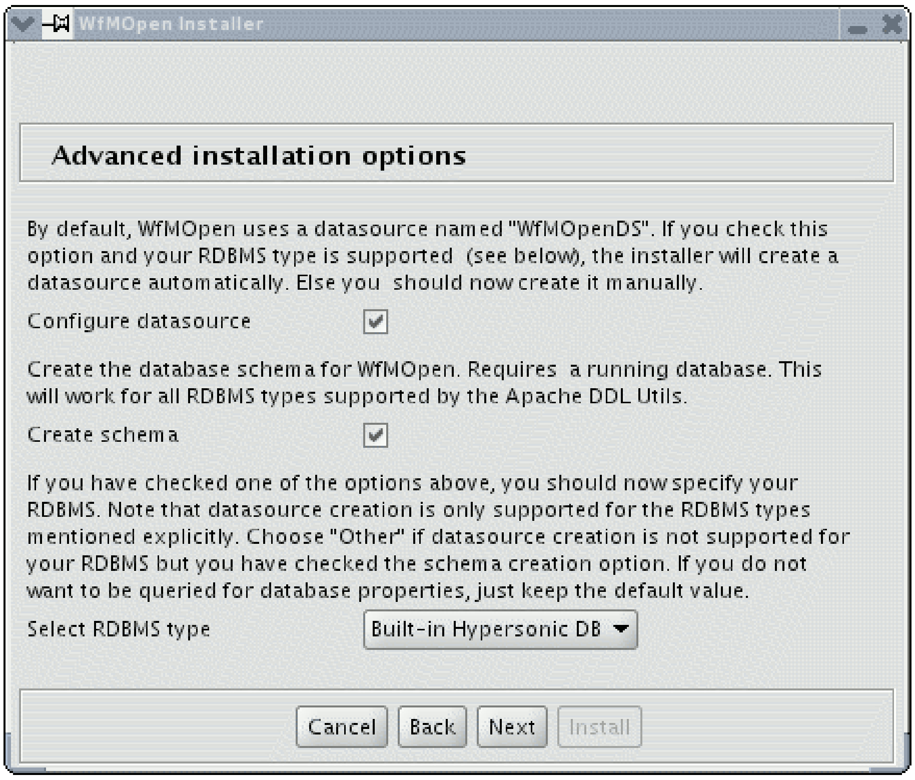
Advanced installation options

Most of the following installation options add to or modify the JBoss configuration. Therefore, please specify the JBoss configuration directory which is the sub-directory "server/<config>" of your JBoss installation.

JBoss configuration directory: /opt/jboss

Figure C.5. JBoss configuration directory

The WfMOpen components require a datasource. The name of the datasource is set in the application server specific configuration files of the EJBs and may, of course, be changed there. It defaults to "WfMOpenDS" and the installer can create a matching datasource configuration in JBoss for several RDBMS types. Unless your database is not supported, or you want to do it by hand for some other reason, you should check this option and select your RDBMS type as described below.



Advanced installation options

By default, WfMOpen uses a datasource named "WfMOpenDS". If you check this option and your RDBMS type is supported (see below), the installer will create a datasource automatically. Else you should now create it manually.

Configure datasource ✓

Create the database schema for WfMOpen. Requires a running database. This will work for all RDBMS types supported by the Apache DDL Utils.

Create schema ✓

If you have checked one of the options above, you should now specify your RDBMS. Note that datasource creation is only supported for the RDBMS types mentioned explicitly. Choose "Other" if datasource creation is not supported for your RDBMS but you have checked the schema creation option. If you do not want to be queried for database properties, just keep the default value.

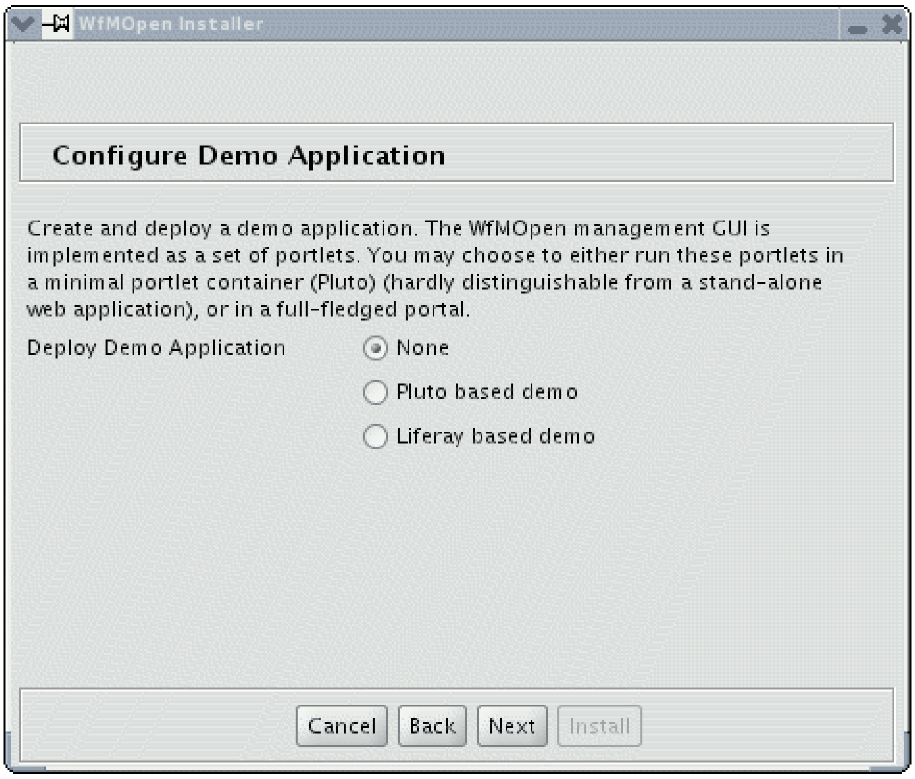
Select RDBMS type: Built-in Hypersonic DB

Figure C.6. Advanced installation options

WfMOpen requires the existance of several tables (a.k.a schema) in the database. The installer can create and initialize these tables for all databases supported by the Apache DDLUtils. Normally, you should leave this option checked.

Finally you have to select your RDBMS type. If your RDBMS is listed, the installer supports datasource creation and it will continue with simplified screens for the specification of the database instance. If you select "Other", it continues with generic screens that allow you to specifiy the parameters for JDBC setup and schema initialization.

After prompting for the information for datasource creation and schema initialization, you may now choose the demo application to deploy.


WfMOpen Installer

Configure Demo Application

Create and deploy a demo application. The WfMOpen management GUI is implemented as a set of portlets. You may choose to either run these portlets in a minimal portlet container (Pluto) (hardly distinguishable from a stand-alone web application), or in a full-fledged portal.

Deploy Demo Application [✓ None] Pluto based demo Liferay based demo

Figure C.7. Choose demo application

If you select "Pluto based demo", no further information is required. The Pluto based demo provides a minimal portlet container that allows a set of portlets to be used almost as a stand-alone application. The next screen will simply provide some information where to point your browser at after the installation and you can proceed to the execution of the installation.

If you choose "Liferay based demo", the portlets will be configured for a Liferay portal. The Liferay portal must previously have been configured e.g. using the installer option described above.

C.1.4. Working with the Pluto based demo

You can find more information about the Pluto potal driver on its home page (see Apache Pluto site [http://portals.apache.org/pluto/v101/userguide/portal.html]). Basically, the demo uses the portal driver as described. The main difference is that we have added a login page to establich a user's identity.

As the Pluto portal driver (being a minimal implementation of a portal) does not include user management, we have resorted to simple file based management for this demo. You find the configured users and groups in `$JBOSS_HOME/server/wfdemo/config/wfdemopluto-*.properties`. The files `wfdemopluto-users.properties` and `wfdemopluto-roles.properties` use the format described for the JBoss file based authentication module (see org.jboss.security.auth.spi.UsersRolesLoginModule [http://docs.jboss.com/jbossas/admindevel326/html/ch8.chapter.html#d0e16599]) and are used by JBoss to verify credentials and assign roles (see application policy "wfdemopluto" in `$JBOSS_HOME/server/`*`configuration`*`/config/login-config.properties`)

The mentioned files and the additional file `wfdemopluto-groups.properties` are also used by the resource management system configured for the demo (see Section 6.3, “The underlying resource management service” [66], Section 6.4.2, “EIS based RMSes” [67] and Section 6.4.2.2.1, “Properties based resource adapter” [67]). So all users, groups and roles configured in these files are available for activity assignments in the workflow.

C.1.5. Working with the Liferay based demo

Liferay is a full fledged portal that comes with its own user management facility.

Liferay does not maintain different roles on the J2EE level. I.e. neither Liferay groups nor Liferay roles are mapped to roles in the J2EE environment. Instead Liferay makes all users member of a single role "user". The configuration created by the installer therefore reduces the security contraints on the workflow engine EJBs to mamabership in role "user", i.e. all Liferay portal users are by default allowed to access the EJBs.

Appendix D. Installing Liferay

The Liferay Portal [http://www.liferay.com] comes with installation instructions for JBoss. We think that these instructions do not lead to the best result. They imply changes to the base JBoss configuration that are confusing for the casual JBoss user and that are not really necessary. We therefore provide in this chapter modified installation instructions that we use to setup a Liferay Portal.

D.1. Creating a configuration

A JBoss server can have several configurations. As distributed, it has the predefined configuration "default", "minimal" and "all". We follow the good practice of creating our own configuration. This leaves the JBoss distribution intact and you don't have to unpack a complete JBoss for every application environment that you want to experiment with.

As a first step, we create a configuration for the Liferay Portal. Create a new directory `$JBOSS_HOME/server/`*`configuration`* using e.g. "`liferay`" for *`configuration`*. Copy the complete content of your start configuration (typically `$JBOSS_HOME/server/default/`) to this newly created directory.

Remove file `$JBOSS_HOME/server/`*`configuration`*`/lib/hibernate3.jar`[1].

D.2. Deploying Liferay

D.2.1. Disabling the default root context application

Rename *`$JBOSS_HOME`*`/server/`*`configuration`*`/deploy/jbossweb-tomcat55.sar/ROOT.war` to `ROOT.war.sav` (or delete it).

D.2.2. Unpacking the JBoss extension libraries

Go to directory *`$JBOSS_HOME`*`/server/`*`configuration`*`/lib/` and unpack the file `liferay-portal-dependencies-X.Y.Z.zip` from the Liferay distribution.

D.2.3. Unpacking the EAR

Now go to directory *`$JBOSS_HOME`*`/server/`*`configuration`*`/deploy/` and create a new directory "`liferay-portal.ear`". Unpack the content of the distribution file `liferay-portal-X.Y.Z.ear` into this directory[2][3].

D.2.4. Fixing the transaction manager configuration

Rename the just unpacked file `counter-ejb.jar` to `tmp.jar`. Create a new directory `counter-ejb.jar` as subdirectory of *`$JBOSS_HOME`*`/server/`*`configuration`*`/deploy/liferay-portal.ear/` and unpack the content of `tmp.jar` into it. Remove `tmp.jar`.

[1]JBoss' approach to provide Hibernate functionality as (non-standard) part of the container may be convenient under certain circumstances, but it may also lead to conflicts with Hibernate libraries provided by deployments (e.g. Liferay comes with a specific Hibernate version).
[2]We unpack the EAR (and embedded components, see below) partially because this allows easier access to some configuration files that we have to adapt.
[3]If you use Windows, you may experience problems with filenames being too long. This happens if the length of the path to your JBoss installation exceeds a certain limit. As a workaround, create the directory `liferay-portal.ear/` in the top-level directory of your drive, unpack the EAR into it, and then move it to *`$JBOSS_HOME`*`/server/`*`configuration`*`/deploy/`.

Edit *$JBOSS_HOME*/server/*configuration*/deploy/liferay-portal.ear/counter-ejb.jar/META-INF/counter-spring-enterprise.xml. Replace the bean definition

```
<bean id="liferayTransactionManager"
  class="org.springframework.orm.hibernate3.HibernateTransactionManager"
  lazy-init="true">
  <property name="dataSource">
    <ref bean="liferayDataSource" />
  </property>
  <property name="sessionFactory">
    <ref bean="liferaySessionFactory" />
  </property>
</bean>
```

with

```
<bean id="liferayTransactionManager"
  class="org.springframework.transaction.jta.JtaTransactionManager"
  lazy-init="true"/>
```

D.2.5. Adapting the portal configuration

Rename the file portal-ejb.jar previously unpacked into *$JBOSS_HOME*/server/*configuration*/deploy/liferay-portal.ear/ to tmp.jar. Create a new directory portal-ejb.jar as subdirectory of *$JBOSS_HOME*/server/*configuration*/deploy/liferay-portal.ear/ and unpack the content of tmp.jar into it. Remove tmp.jar.

Create a new file *$JBOSS_HOME*/server/*configuration*/deploy/liferay-portal.ear/portal-ejb.jar/portal-ext.properties with the following content:

```
portal.release=enterprise
portal.ctx=/
auto.deploy.dest.dir=../server/configuration/deploy
auto.deploy.deploy.dir=../server/configuration/liferay-deploy
lucene.dir=../server/configuration/data/lucene
jcr.jackrabbit.repository.root=../server/configuration/data/jackrabbit

portal.configuration=false
portal.jaas.enable=true
```

Note that the relative paths used assume that JBoss will be started with $JBOSS_HOME/bin as working directory.

D.2.6. Configure security

With the configuration above, Liferay uses the standard JBoss mechanism for configuring security (instead of doing it programatically). Open *$JBOSS_HOME*/server/*configuration*/conf/login-config.xml in an editor and at its end, but before the final </policy> insert

```
<!-- Security configuration for Liferay -->
<application-policy name="PortalRealm">
  <authentication>
    <login-module code="com.liferay.portal.kernel.security.jaas.PortalLoginModule"
  </authentication>
</application-policy>
```

D.3. Providing a database

Liferay can be used with many databases. This section covers only using Liferay with embedded Hypersonic SQL. This is the easiest route because you do not have to install an RDBMS.

First create a datasource configuration in `$JBOSS_HOME/server/`*`configuration`*`/deploy`. We suggest to create a deployment descriptor `liferay-ds.xml` with the content:

```
<?xml version="1.0"?>

<datasources>
  <local-tx-datasource>
    <jndi-name>jdbc/LiferayPool</jndi-name>
    <connection-url>jdbc:hsqldb:${jboss.server.data.dir}${/}hypersonic${/}li
    <driver-class>org.hsqldb.jdbcDriver</driver-class>
    <user-name>sa</user-name>
    <password></password>
    <min-pool-size>5</min-pool-size>
    </local-tx-datasource>
</datasources>
```

Create a directory "`sql/`" and unpack `liferay-portal-sql-X.Y.Z.zip` from the liferay distribution into it. Change your working directory to `sql/`.

Now create a new Hypersonic DB with this command:

```
java -cp "$JBOSS_HOME/server/configuration/deploy/liferay-portal.ear/portal-
$JBOSS_HOME/server/configuration/deploy/liferay-portal.ear/lib/util-java.jar
$JBOSS_HOME/server/configuration/lib/hsqldb.jar" \
com.liferay.portal.tools.DBLoader \
hypersonic $JBOSS_HOME/server/configuration/data/hypersonic/liferayDB
```

Note that on Windows, you'll have to replace the column (":") used as path separator above with the semicolon (";").

D.4. Starting JBoss

Change the working directory to `$JBOSS_HOME/bin/`. Run `run.bat` with options "`-c liferay`".

Appendix E. Notes

- This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

Appendix F. GNU General Public License

Version 2, June 1991
Copyright © 2000 Free Software Foundation, Inc.

Free Software Foundation, Inc.
59 Temple Place, Suite 330,
Boston,
MA 02111-1307
USA

.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

F.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

F.2. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

F.2.1. Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a “work based on the Program” means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term “modification”.) Each licensee is addressed as “you”.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

F.2.2. Section 1

You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

F.2.3. Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License.

   Exception:

   If the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

F.2.4. Section 3

You may copy and distribute the Program (or a work based on it, under Section 2 in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

F.2.5. Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

F.2.6. Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

F.2.7. Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

F.2.8. Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

F.2.9. Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

F.2.10. Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

F.2.11. Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

F.2.12. NO WARRANTY

Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

F.2.13. Section 12

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Index