XML Application Processing Engine

Developer's Guide

Product Version 2.1.56

Version : 1.28
Date : 2006 March 25
Author : Virtual Weaver Interactive

The purpose of this document is to describe how to use XAPE to create Java applications described in XML.
XAPE is a Java API built with Xotics Core API. It uses many features of this base API.
XAPE is to be considered as an application framework, which can be integrated in any Java program, such as stand-alone application or J2EE servlet. With XAPE, an application is defined by a set of XML documents representing processing and data.

Architecture Overview

XAP Engine is designed to execute applications. A XAPE application is defined by an XML document containing processing instructions, written in XAPE-APP dialect. To be executable, each application must be registered (we say 'loaded') by XAPE, to detail how to execute it.

An application currently executed is called 'Application Instance', which can be considered as a process. An Application Instance is composed with a set of XML documents (and XML only) :

- the main application document, written in XAPE-APP, which contains processing instructions ;
- the Private Context, a private document written in XAPE-Context, containing current request and a space to store any data ;
- any additional documents, needed for Instance execution, including other XAPE-APP documents.

The main application document and the Context can be accessed by current Instance only, whereas additional documents can be common to, shared by several Instances, including from different applications.
Every document accessible to an Instance belongs to a group, called 'Space'. The Space containing mandatory documents (main app document and Context) of an Instance is called the 'Private Space'. It's the default group for Instance documents and is reserved to the Instance, whereas other Spaces are called 'Share Spaces', they are used to make accessible particular documents to several Instances.

The only way to communicate with a XAPE Instance is to send message (also called 'request'). A message has a target application and possibly Instance. It can require a response or not. If target is not found, Instance is created (if allowed) before message delivery. This is the way to create new Instance.

XAP Engine can execute any Application Instance in dedicated thread from Engine's configurable pool of threads. The Engine can also let user's thread control Instance execution. For instance, Xotics Editor, which is powered by XAPE, is processed by its own dedicated thread, whereas Servlet version of the Engine (XapServlet) is executed by threads of Servlet Container.

API contents

The main package of XAPE API is com.virtualweaver.xotics.dialect.xape.engine. It contains the main class of this API, XapEngine. This class can configure, start and stop XAP Engine, deploy Applications and send messages to launch and communicate with Application Instances.
XapEngine is embedded in an executable version, XapPlayer (in package ...xape.player), which adds also an XML dialect for configuration and deployment of XAP Engine. XapEngine is integrated too in a J2EE Servlet called XapServlet (in pkg ...xape.servlet).
XAPE provides also two other essential dialects : XAPE-APP for creating processing documents, and XAPE-Context for representing Private Context documents.

Note : XAPE API needs a JAR file from Sun J2EE 1.3.1 JDK, j2ee.jar. This file must be present in the same directory as XAPE JAR file.

Processing dialect (XAPE-APP)

This is the application processing dialect. It can be considered as a programming language, described in XML and XPath, easily extensible. Each element of this dialect represents a basic instruction of this language. Its implementation is made with Xotics Core API so, with XPath requesting support, this dialect is particularly efficient in XML document handling.
Thus, XAPE-APP elements are all implemented as XO objects (and derived classes), and implement a special Java interface making them able to perform custom processing. This processing is helped by XAPE runtime environment, an interface to the Engine.

Overview

In a XAPE-APP document, processing instructions are structured in states, represented by <state> element. A <state> is a processing container representing a particular state of the Instance. An APP document can contain any count of <state> and jump from one to other like a finite-state automaton. There is two special and mandatory states, <state-start> and <state-end>. Application Instance starts the execution of its main APP document by processing the content of <start-state>, and finishes by processing content of <end-state>.
A <state> processing unit contains mostly a list of <request-handler> elements, each containing a condition evaluated as an XPath expression. The first <request-handler> having its condition evaluated to 'true' is processed. By default, after selected <request-handler> has been processed, Instance stays in current <state> and evaluate its <request-handler> again. If an instruction <next> has been encountered, Instance jumps to the state specified by @state and continues processing. It is not required for a <state> to have <request-handler> ; two other processing elements can be children of <state>, <preprocess> which is executed before <request-handler> evaluations, and <postprocess> processed at the end of state cycle.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<application id="MainApp xmlns=".../application200.dtd" >
  <state-start waitForRequest="false" >
    <preprocess >
      <bind-document docName="Functions" location="Functions.xml" />
      <call-function select="doc(Functions)fn:id('startFunc')" />
      <next state="MainApp.NormalState" />
    </preprocess>
  </state-start>
  <state waitForRequest="true" id="MainApp.NormalState" >
    <request-handler id="request1"
         test="doc(privateXapContext)//action-event/@command = 'Select'" >
      <call-function select="doc(Functions)fn:id('Func2')" />
      <next-end />
    </request-handler>
  </state>
  <state-end/>
</application>
```

Sample of XAPE-APP document

An Instance can wait for a request message and perform a processing in response to this request. This is specified in <state> element by @waitForRequest.
So, a state processing cycle is the sequence of :

- waiting for a request, if @waitForRequest is true;
- processing <preprocess> element, if present ;
- evaluating each <request-handler>, if any, and processing the one returning 'true' (if any) ;
- processing <postprocess> element, if present ;

The Instance ends after processing <state-end>, which can be processed just once. So, requesting the end of an Instance consists in jumping to <state-end> by a <next> element.

XPath requesting

XPath requesting is a fundamental feature for XAPE-APP dialect. It is based on Xotics Core API XPath requesting support, enhanced by XAPE API.

XPath format

A request can appear anywhere in a XAPE-APP document, in attribute or PCDATA values. Like with Xotics Core API, XPath requests are performed on a single target document (which is a restriction of XPath specs), an XPath request has the format :

```
doc(<sharespace>#<docname>)<xpath>
```

where :

- <sharespace># is the optional name of a Share Space containing target document ; if document is bound to Application Instance, that is, belongs to Instance Private Space, this field and the separator '#' must not appear ;
- <docname> is the name of an XML target document, accessible to current Application Instance,
- <xpath> is the XPath request text.

Notice that doc() is optional : if not mentionned, target document is current main Instance processing document. There can be only one occurence of doc() in a request and must be at start of the request.

Textual value resolution

All XAPE-APP attribute and PCDATA values can have parts of text whose value is the result of XPath request. Such part has the following format :

```
input text : “some parts of text result from {/root/element/@name} requests”

XML : <element name=”xpath”>

resolved text : “some parts of text result from xpath requests”
```

Everything between enclosing { } is evaluated as an XPath request and result is converted into string (if necessary) by xs:string() XSD type. The string result then replaces request and brackets.

Requests can be nested, like this :

```
input text : “this text {/root/container/{/root/element/@name}} requests”

XML : <element name=”text()”>
      <container>result from xpath</container>

resolution process :
first step :    “this text {/root/container/text()} requests”
resolved text : “this text result from xpath requests”
```

This operation is automatically performed by XAPE runtime environment method resolveValue(), and in xpath requesting methods, resolution is done also on XPath string before evaluation.
Every XAPE-APP element calls this method on its attributes and PCDATA content when possible (almost everywhere) when starting its process() method execution. When implementing a XAPE task, the developer should do the same when retrieving argument values.

Runtime environment

Every XAPE task or processable element from XAPE-APP or any other extending dialect needs a runtime environment to perform its processing. Each Application Instance has its own runtime environment. This runtime environment is both an interface to interact with the Engine and a store for some Instance data.

XapRuntimeEnvironment interface

```
package com.virtualweaver.xotics.dialect.xape.engine;

public interface XapRuntimeEnvironment extends Cloneable {
  public Object clone();
```

Though it is an interface, XapRuntimeEnvironment is implemented exclusively by XAPE API.
Runtime environment is instanciated first on and for each Instance creation, and is often cloned to provide to some XAPE-APP element a specific environment separated from its parent.
Here below are its methods, which can be grouped in three categories :

- informations methods about XAPE general environment ;
- instance data management methods ;
- utility methods.

General informations

```
public XoEnvironment getXoEnvironment();

public String[] getShareSpaceNames() throws XapException;
public String[] getApplicationNames() throws XapException;
public boolean isMultiInstances(String appName) throws XapException;
public String[] getInstanceNames(String appName) throws XapException;

public String getApplicationName();
public String getInstanceName();
```

These methods give informations about which Applications are loaded, which Application Instances are running and which Share Spaces are created. Runtime environment provides also current Application and Instance names.

Instance data management

```
public XoDMInstance getDocument(String sharespace,
                                String docName) throws XapException;
public XoDMInstance getDocument(XapDocumentId did) throws XapException;
public XapDocumentId[] getDocumentNames() throws XapException;
public XoDMInstance bindDocument(String sharespace,
                                 String docName,
                                 URL url,
                                 Map initOptions,
                                 Object initObject,
                                 boolean force) throws XapException;
public XoDMInstance bindDocument(String sharespace,
                                 String docName,
                                 XoDMInstance document,
                                 Map initOptions,
                                 Object initObject,
                                 boolean force) throws XapException;
public XoDMInstance createDocument(String sharespace,
                                   String docName,
                                   String nsref,
                                   String prefix,
                                   String element,
                                   Map initOptions,
                                   Object initObject,
                                   boolean force) throws XapException;
public void unbindDocument(String sharespace,
                           String docName,
                           boolean release) throws XapException;

public URL getBaseUrl();
public String getCurrentState();
public String getNextState();
public void setNextState(String state);

public XoNode getXPathContextNode();
public void setXPathContextNode(XoNode node);
```

```
public Map getXPathVariables();
public void addXPathVariable(String name, Object value);
public void removeXPathVariable(String name);
```

The main source of data comes from XML documents which are bound to some specific Application Instance or shared in a Share Space. So, every document accessible to current Instance is accessed by document methods of Runtime environment. The environment can also create and bind new documents and can choose when to unbind some document, including shared documents.
XAPE Runtime environment manages also data relative to XPath requests : XPath variables and XPath context node. XAPE runtime environment uses these data on every XPath request evaluation.
Runtime environment sets and gets also the next instance state, and gives current state.
Each instance can be created with a base URL provided on application loading, used to resolve relative URL by utility method resolveUrl() of environment.

Utility methods

```
public void reloadApplication(String name) throws XapException;
public URL resolveUrl(String url) throws XapException;
public String resolveValue(String val) throws XapException;
public XapResponse sendRequest(XapRequest req) throws XapException;
public XoDMInstance bindDmdl(String sharespace,
                             String docName,
                             String nsref,
                             boolean force) throws XapException;
public XoDataType xpathRequest(String req) throws XapException;
public XoNode[] xpathRequestNodes(String req) throws XapException;
public void debug(String id, String element, String message);
public void log(String id, String element, String message);
```

Any instance can ask for reloading the main processing document for any already loaded application. Subsequent Instance creation is done with updated main XAPE-APP processing document. By default, it has no effect on mono-instance application already started, except if this mono-instance application reloads its own processing document : it can then end current instance and start a new instance again.
resolveURL() creates an absolute URL with a relative URL and URL provided by getBaseUrl().
resolveValue() replaces an XPath request enclosed in {} brackets in some input string by a text resulting from interpretation of the request.
XAPE runtime environment gives two methods to perform XPath requests. These methods apply string resolution on XPath request before evaluation.
debug() and log() are not used yet, an enhanced logging and tracing system is coming soon.

Element description

Structuring elements

Element <application>

This is the root element of an application document.

Content-Model :	
(functions?, state-start, state*, state-shutdown?, state-end)	
Attributes :	
id	Optional, can contains the name of the aplication

Element <state>

The main processing unit of an application. It represents a particular state in the application. A state's processing code is executed as long as state doesn't change. <state> content is processed as a cycle :

- starting by waiting for a request if specified,
- then executing optional <pre-process>,
- evaluating each <request-hnadler> condition and processing the first giving a result 'true',
- then finishing bu processing optional <post-process>.

If an exception is thrown during processing, <error-handler> element is processed if present.

Content-Model :	
(pre-process?, request-handler*, post-process?, error-handler?)	
Attributes :	
id	Required, the name of the state. As any ID, must be unique. Note that 3 state Ids are reserved : "xapStart", "xapEnd", "xapShutdown"
waitForRequest	Boolean to indicate whether this state must wait for a new request at each execution cycle. True by default
waitingTime	Time to wait for a request before continuing processing. -1 means no limit, and is default value

Element <state-start>

This element represents the state chosen to start an application processing. It works like a <state> with @id set to "xapStart".

Content-Model :	
(pre-process?, request-handler*, post-process?, error-handler?)	
Attributes :	
waitForRequest	Boolean to indicate whether this state must wait for a new request at each execution cycle. True by default
waitingTime	Time to wait for a request before continuing processing. -1 means no limit, and is default value

Element <state-end>

This element represents the last state processed. It works like a <state> with @id set to "xapEnd", excepted that it can be processed just once, and can not wait for a request. Jump to this state to end the application. Application processing jumps to this state when corresponding Instance receives a shutdown message and application document contains no <state-shutdown>.

Content-Model :
(pre-process?, request-handler*, post-process?, error-handler?)

Element <state-shutdown>

Application processing jumps to this optional state if a shutdown message is sent to this Instance. It works like a <state> with @id set to "xapShutdown", excepted that it can be processed just once, and can not wait for a request.

Content-Model :
(pre-process?, request-handler*, post-process?, error-handler?)

Element <pre-process>, <post-process>

These elements are processing containers, executed respectively at the beginning and the end of a state processing iteration.

Content-Model :	
Any processable elements	
Attributes :	
id	Optional, to identify a particular element

Element <next>, <next-end>, <next-shutdown>

These elements select the next state in an application processing. The state containing one of these elements is processed normally, the processing jumps to the new state after ending current iteration, then continues with the new state processing code. <next> can select any state, <next-end> jumps to <state-end> and <next-shutdown> jumps to <state-shutdown>.

Attributes :	
state	Required for <next> to set new state ID, fixed for <next-end> ans <next-shutdown>
id	Optional, to identify a particular element

Element <error-handler>

This element's content is processed when an exception occured in any processing container parent of this element. When exception is thrown, the processing is interrupted to start executing <error-handler> content. Just before, an element <exception>, from XAPE-Context dialect, is created or updated in Bag section of the private context, to contain relevant informations about exception thrown.

Content-Model :	
Any processable elements	
Attributes :	
id	Optional, to identify a particular element
exceptionName	Name of an element <exception> (from XAPE-Context dialect) created or updated to contain informations about the exception
exception	The Exception object thrown

Element <functions>

<functions> is just a container for <function> children elements.

Content-Model :	
(function*)	
Attributes :	
id	Optional, to identify a particular element

Conditional elements

Element <request-handler>

This element's content is processed when its @test, interpreted as an XPath expression, returns true. <request-handler> is designed to work as child of <state> and in case of several <request-handler> children, the first returning true only is processed.
A <request-handler> element can be accessed and processed directly by its ID, if specified in a XAPE request.

Content-Model :	
Any processable elements	
Attributes :	
id	Required to name each <request-handler>
test	An XPath expression whose result is interpreted as a boolean.

Element <condition>, <action>, <alt>

These elements represent an if-then-else expression. @test of <condition> is interpreted as an XPath expression, and evaluation result as a boolean. On true result, <action> child is processed, else <alt> is processed if present.

Content-Model :	
<condition> : (action, alt?, error-handler?) <action>, <alt> : Any processable elements	
Attributes :	
id	Optional, to identify a particular element
test (condition)	An XPath expression whose result is interpreted as a boolean.

Element <test>, <case>

These elements represent a switch-case expression. <test> is just a container of <case> elements followed by an optional <alt> child. @condition of each <case> child is interpreted as an XPath expression and evaluation result as a boolean. The first <case> returning true has its content processed. If no child returns true, the optional <alt> child is processed.

Content-Model :	
<test> : (case*, alt?, error-handler?) <case> : Any processable elements	
Attributes :	
id	Optional, to identify a particular element
condition (case)	An XPath expression whose result is interpreted as a boolean.

Tasks and functions

Element <reload-app>

Reloads the processing document of an already loaded application, if this document has been located by URL. This allows to update an existing application at any time of Engine execution, especially during runtime. After reloading, current instances of this application remain unchanged, but subsequent instance creations are made with updated document. There is two ways to load such application document :

- by XML config document of XapPlayer ;
- or by method loadApplication() of XapEngine.

In the first case, the document is loaded by URL, but with loadApplication() method, the document is loaded or created by the user, by any available method from XoFactory.
If <reload-app> doesn't find the location of original document, an exception is thrown.

Attributes :	
name	Required name of an already loaded application to update
id	Optional, to identify a particular element

Element <call-task>, <argument>

This element permits to call a Java class to perform some operation. This class must implement com.virtualweaver.xotics.dialect.xape.engine.XapTask interface. <call-task> has two attibutes, mutually exclusive : if @tclass is not set, @ref must refer to a <declare-task> element having an instance of XapTask class.
A XapTask object can accept arguments as string couples (name, value), defined by <argument> children.
At each call to <call-task> :

- if @tclass is set, arguments are set, XapTask.init() is called, then processing is performed ;
- if @ref is set, task is retrieved from <declare-task>, arguments are set, then processing is performed.

Content-Model :	
(argument*)	
Attributes :	
id	Optional, to identify a particular element
tclass	A XapTask object. Be careful, if you are using Xotics Editor to edit such property, to get specified XapTask class accessible by class loader.
ref	XPath expression to identify a <declare-task> element.

Element <declare-task>

This element avoids to instanciate as many XapTask object as <call-task> elements, by permitting to reuse a XapTask declared by this element and to initialize it only once. Possible arguments are retrieved and set to the task before calling init().

Content-Model :	
(argument*)	
Attributes :	
id	Required to allow <call-task> element to refer to this element
tclass	A XapTask object, usable by several <call-task> elements. Be careful, if you are using Xotics Editor to edit such property, to get specified XapTask class accessible by class loader.

Element <function>, <call-function>

<function> is a container of reusable processing instructions, which are processed at each call to <call-function>.

Content-Model :	
<Function> : Any processable elements	
Attributes :	
id	Required for <function> to identify the function
Select (call-function)	XPath expression to identify a <function> element.

Document managing

Element <bind-document>, <unbind-document>

Loads or unloads an XML document and adds it to or remove from a particular share space. In XAPE environment, a bound document is identified by a share space and a document name. A null share space value means Private Space.

Attributes :	
id	Optional, to identify a particular element
shareSpace	Name of the share space containing the document to (un)load. If not specified, the share space is the Private Space
docName	Required name to give to the document to bind or name of the document to unbind
Location (binding)	Required URL of the source XML document to bind
historySize (binding)	Size of the historization buffer of the document to bind (default -1)
Requestable (binding)	Tells whether the document to bind is XPath requestable (default true)
readOnly (binding)	Tells whether the document to bind is read-only (default false)
Force (binding)	If true, the document to bind replaces silently a possibly already bound document. The default is false, which means that if a document is already bound with (shareSpace, docName) identity, an exception is thrown

Element <copy-document>

Copy an already bound document to a new Share Space with new name. New document is cloned from source, and bound with the same attributes as <bind-document>.

Attributes :	
id	Optional, to identify a particular element
srcShareSpace	Name of the share space containing the document to copy. If not specified, the share space is the Private Space
srcDocName	Required name of the document to copy
trgShareSpace	Name of the share space containing the copied document. If not specified, the share space is the Private Space
trgDocName	Required name to give to the document to copy
historySize (binding)	Size of the historization buffer of the document to bind (default -1)
Requestable (binding)	Tells whether the document to bind is XPath requestable (default true)
readOnly (binding)	Tells whether the document to bind is read-only (default false)
Force (binding)	If true, the document to bind replaces silently a possibly already bound document. The default is false, which means that if a document is already bound with (shareSpace, docName) identity, an exception is thrown

Element <create-document>

Creates an XML document and binds it to a particular share space. In XAPE environment, a bound document is identified by a share space and a document name. A null share space value means Private Space.

Attributes :	
id	Optional, to identify a particular element
shareSpace	Name of the share space containing the document to create. If not specified, the share space is the Private Space
docName	Required name to give to the document to create
nsRef	Required namespace of the root element
rootElement	Required name of the root element
historySize	Size of the historization buffer of the document to create (default -1)
Requestable	Tells whether the document to create is XPath requestable (default true)
readOnly	Tells whether the document to create is read-only (default false)
Force	If true, the document to create and bind replaces silently a possibly already bound document. The default is false, which means that if a document is already bound with (shareSpace, docName) identity, an exception is thrown

Element <lock-document>

Locks a specific bound document during the processing of <lock-document> children.

Content-Model :	
Any processable elements	
Attributes :	
id	Optional, to identify a particular element
shareSpace	Name of the share space containing the document to lock. If not specified, the share space is the Private Space
docName	Required name of the document to lock

Element <undo-document>

Cancels last modification performed on bound document identified by (sharespace, docName), provided @historySize != 0 when document has been bound.

Attributes :	
id	Optional, to identify a particular element
shareSpace	Name of the share space containing the document to modify. If not specified, the share space is the Private Space
docName	Required name of the document to modify

Element <save-document>

Saves a bound document to a target location identified by URL.

Attributes :	
id	Optional, to identify a particular element
shareSpace	Name of the share space containing the document to save. If not specified, the share space is the Private Space
docName	Required name of the document to save
location	Required target URL to save the document
encoding	Optional XML encoding to save (default : ISO-8859-1)

Element <bind-dmdl>

Binds a DMDL document.

Attributes :	
id	Optional, to identify a particular element
shareSpace	Name of the share space containing the DMDL document to bind. If not specified, the share space is the Private Space
docName	Required name of the DMDLdocument to bind
Force	If true, the document to bind replaces silently a possibly already bound document. The default is false, which means that if a document is already bound with (shareSpace, docName) identity, an exception is thrown
nsRef	Required namespace of the desired DMDL document

Object managing

Element <for-each>

This container executes @select as an XPath expression, and processes its content for each XoNode object found in XPath result. For each iteration, the XoNode is set as XPath context node, which means that every XPath expression of children elements is evaluated with this context node.

Content-Model :	
Any processable elements	
Attributes :	
id	Optional, to identify a particular element
select	An XPath expression which should return a list of XoNode objects

Element <set-property>

This element sets value to a selection of XoObject propertie(s). Propertie(s) are selected by an XPath expression. Value is provided in its text (XML) representation.

Attributes :	
id	Optional, to identify a particular element
nullify	If this boolean is true, sets the propertie(s) to null
PCDATA	Value to set, in XML text representation
select	An XPath expression which must return a list of XoProperty objects

Element <add-child>

Adds an XoObject child to an XoContainer parent. This element has several ways to select the parent, but the child is always provided by the first child element of <add-child>, which can be a <create-object>, <remove-object> or <copy-object>. The possible next children of <add-child> work on the newly added child : for these children, the context node becomes the newly added child.

<add-child> Has three ways to add the new child :

- First, if @to is set, the child is added to the first object selected by XPath expression in @to, at index designated by @index ;
- Else, if @point is set, the child is added before or after the object selected by XPath expression in @point, depending on the value of @position which can be the constant BEFORE or AFTER ;
- Else, the child is added to the current context node, if it is an XoContainer, at index chosen by @index.

If none of these options works, an exception is thrown, which can be capted by an <error-handler> element.

Content-Model :	
((create-object | copy-object | remove-object), [any processable element]*, error-handler?)	
Attributes :	
id	Optional, to identify a particular element
to	XPath expression selecting the parent, if result contains several objects, the first only is used
index	Insert index, for a parent selection by @to or by the current context node
point	New child is added just before or after the object selected by this XPath expression. If expression in @point gives several objects, the first only is used
position	Constant BEFORE or AFTER (default), used with @point to choose to add new child before of after the one selected by @point

Element <create-object>

This element creates an XoObject, and puts it as current context node. This is the way to give the result to a <add-child>.

Attributes :	
id	Optional, to identify a particular element
nsRef	Namespace of the element to create
elementName	Local name of the element to create

Element <copy-object>

This element copy an existing XoObject, and puts it as current context node. This is the way to give the result to a <add-child>.

Attributes :	
id	Optional, to identify a particular element
select	XPath expression to select object to copy (in case of multiple result, the first object only is copied)
deep	If true (the default), copies entire sub-tree, else just copies this object

Element <remove-object>

This element removes an object which can be :

- First, either the child at index @index of the parent in current context node,
- Or selected by XPath expression of @select.

If none of these options work, an exception is thrown.

Attributes :	
id	Optional, to identify a particular element
select	XPath expression to select object to remove (in case of multiple result, the first object only is removed)
index	If not -1 (the default), the object to remove is the child at this index of the current XoContainer context node.

Element <move-child>

This element removes an object to add it to a new parent. The child to move is selected by an XPath expression in @from. <move-child> Has three ways to add the child moved :

- First, if @to is set, the child is moved to the first object selected by XPath expression in @to, at index designated by @index ;
- Else, if @point is set, the child is moved before or after the object selected by XPath expression in @point, depending on the value of @position which can be the constant BEFORE or AFTER ;
- Else, the child is moved to the current context node, if it is an XoContainer, at index chosen by @index.

If none of these options works, an exception is thrown, which can be capted by an <error-handler> element.
After having performed the move, children of <move-child> are processed with the object moved as context node, which means that every XPath expression in <move-child> children is executed with the moved object as context node.

Content-Model :	
Any processable elements	
Attributes :	
id	Optional, to identify a particular element
from	Required XPath expression selecting the child to move. If expression selects several objects, the first only is taken.
to	XPath expression selecting the target parent, if result contains several objects, the first only is used
index	Insert index, for a parent selection by @to or by the current context node
point	New child is added just before or after the object selected by this XPath expression. If expression in @point gives several objects, the first only is used
position	Constant BEFORE or AFTER (default), used with @point to choose to add new child before of after the one selected by @point

Mailing

Element <send-mail>

Sends an email defined with this element and its children.

Content-Model :	
(mail-recipient+, (mail-file | mail-text)*)	
Attributes :	
id	Optional, to identify a particular element
from	Required email of the sender
server	Required mail server name
subject	A string containing the subject of the message

Element <mail-recipient>

Used only as child of <send-mail>, this element identifies an email recipient. There must be at least one child <mail-recipient>.

Attributes :	
id	Optional, to identify a particular element
type	Required type of recipient, one of the values TO (the default), CC, BCC
address	Required recipient email address

Element <mail-text>

Used only as child of <send-mail>, this element is a part of the textual content of the mail. There can be several children of this type, each text content is concatenated to compose the entire text content.

Attributes :	
id	Optional, to identify a particular element
PCDATA	Text content of the mail

Element <mail-file>

Used only as child of <send-mail>, this element represents a file attachment. There can be as much files as needed.

Attributes :	
id	Optional, to identify a particular element
name	Required name to give to the attachment
location	Required URL of the file to attach

Miscellaneous

Element <set-variable>

This element creates or update an XPath variable stored in XAPE runtime environment and usable in every XPath request made during Application document processing. It's a very useful and used feature of XAPE-APP dialect. <set-variable> updates the XPath variable if it already exists. The value is the result of an XPath request.

Attributes :	
id	Optional, to identify a particular element
name	Required name of the variable
PCDATA	Required Xpath expression whose result is the value

Element <comment>

Writes a comment in standard output.

Attributes :	
id	Optional, to identify a particular element
PCDATA	A string written in standard output

Extending the dialect

XAPE-APP dialect doesn't aim to be exhaustive. It can (and should) be extended. There are two ways to create new reusable objects which can be integrated in a XAPE Application :

- by creating new XO object implementing a specific interface, XapProcessable, usable as XML element ;
- by creating new object (non XO) implementing another interface, XapTask, callable by XAPE-APP <call-task>.

There is no rule to choose one or the other way, it's a matter of personal preference, excepted if you want to take benefit of the tree structure of XML.

Creating new processing element

XapProcessable interface

Any XoObject (and derived) implementing interface XapProcessable can be integrated in a XAPE Application document as new XML element.

```
package com.virtualweaver.xotics.dialect.xape.engine;

public interface XapProcessable {

  public String getId();
  public void process(XapRuntimeEnvironment env) throws XapException;
}
```

Method getId() returns a unique ID to identify the object implementting this interface in a runtime context.
The main method is process(), which performs any operation with the help of XAPE Runtime Environment object. This runtime object represents the runtime context of the application being executed and is the link with the Engine itself. It provides useful functions detailed in next section.

A new XAPE processing element can be a simple XO object (XoObject XoTextContainer) or a container. If the element is an XoContainer, it is responsible to call process() method of its children.
A default implementation for such container is provided by the class AppProcessContainer which can be derived, as explained below.

AppProcessContainer class

This class is a default implementation for a processable container class. It can be derived or used as is.

```
package com.virtualweaver.xotics.dialect.xape.model.app;

public class AppProcessContainer extends XoContainerSupport implements
XapProcessable {


  public AppProcessContainer() {...}
  public String getId {...}
  public void setId(String id) throws PropertyVetoException {...}
  public void checkXoValidity() throws XoValidityException {...}
  public boolean isXoPropertyToWrite(String pname) {...}
  public boolean isXoObjectWelcome(XoObject xo, int index) {...}
  protected Object clone(AppProcessContainer clone) {...}
  public void process(XapRuntimeEnvironment env) throws XapException {...}

}
```

Every XapProcessable having an Id, AppProcessContainer makes it a JavaBean property.
By default, checkXoValidity() requires to set id, and isXoPropertyToWrite() writes it if id is not null.
Any XapProcessable child is welcome by isXoObjectWelcome().
This clone() method can be used by a derived clone() method to get a preconfigured AppProcessContainer clone.
Process() method executes process() method of its children and in case of exception thrown, executes any <error-handler> child if present or throw out the exception.

XAPE Task

A XAPE task is another way to extend XAPE-APP functionnalities, if you don't want to create a new XML element to implement you function. A Task has another useful feature : an initialization method, separated from processing method and possibly called once only.

XapTask interface

A XAPE Task must implement this interface :

```
package com.virtualweaver.xotics.dialect.xape.engine;

public interface XapTask extends XapProcessable {
  public void init(XapRuntimeEnvironment env) throws XapException;
  public void setArguments(Map arguments) throws XapException;
}
```

First, XapTask extends XapProcessable because a Task is a kind of processable object. A task can be configured with parameters passed with setArguments() method and a Map of couples (param name -> param value).

Once arguments are given to the task, init() method is called which permits to perform some init based on arguments. Arguments can also be set before calling process() method, it depends on which task handling element (from XAPE-APP dialect) is called :

- <declare-task> or <call-task> (with @tclass set) calls setArguments() on its task then calls init() ;
- whereas <call-task> (with @ref set) calls setArguments() on its task then calls process().

XapAbstractTask class

In order to simplify XAPE Task creation, an abstract task has been implemented to be derived, providing basic functions.

```
package com.virtualweaver.xotics.dialect.xape.task;
public abstract class XapAbstractTask implements XapTask {

  protected String taskLocation;

  public XapAbstractTask();
  public void setId(String id);
  public String getId();
  public void init(XapRuntimeEnvironment env) throws XapException;
  public abstract void process(XapRuntimeEnvironment env) throws
XapException ;
  public void setArguments(Map map) throws XapException;
  public String getArgument(XapRuntimeEnvironment env, String argKey,
boolean mandatory)
  throws XapException;
}
```

set/getId() are the accessors for XapProcessable required ID property.
Init() method does nothing, but process() is not implemented.
setArguments() stores argument map.
getArgument() provides an argument value, after having resolved it (via XAPE untime environment's method resolveValue()). Boolean param 'mandatory', if true, makes this method throwing a XapException if argument is not present.

XAP Engine

XapEngine class

XAP Engine is the main class of the API. It is responsible to load Applications, run Instances, send messages to Instances. This class is generaly not used directly. The developer could prefer XapPlayer class which adds to XAP Engine a configuration dialect making Engine configuration easier.

Here are the methods of XapEngine class :

```
package com.virtualweaver.xotics.dialect.xape.engine;

public final class XapEngine {
  public XapEngine(XoEnvironment env) throws XapException

  public void setParameter(String param, Object value) throws XapException
  public void addShareSpace(String spaceName) throws XapException
  public void loadApplication(XoDMInstance appDoc,
                              String name,
                              Map parameters) throws XapException
  public XapRepository getRepository()

  public void shutdown()
  public void startup() throws XapException

  public XapResponse sendRequest(XapRequest req) throws XapException
  public static XapRequest createShutdownMessage(XoEnvironment env,
                                                 String appName,
                                                 String instanceName)
                                                      throws XapException
  public static XapRequest createStartupMessage(XoEnvironment env,
                                                   String appName,
                                                   String instanceName)
                                                        throws XapException
  public static XapRequest createRequest(XoEnvironment env,
                                         byte type,
                                         String appName,
                                         String instanceName,
                                         String handler,
                                         String reqId,
                                         XapQueue responseQueue,
                                         XoObject contentObject)
                                         throws XapException

}
```

Configuration

The constructor uses an existing XO environment to load various dialects needed to operate.
Method getRepository() gives the name of loaded applications and running instances.

Init parameters

Method setParameter() sets a XAP Engine init parameter value. Currently, these parameters configure the Thread pool included in XAP Engine, precisely thread count boundaries in the pool and latency time before killing an idle thread between min and max boundaries. Parameter keys are defined in XapConstants :

Engine init parameter key	Description
XapConstants.ECP_MIN_PROCESSES	minimum number of threads in the pool, thread count can not be under this value, even if every thread is idle, valid value is a String or Integer representing a positive number less than Max_PROCESSES
XapConstants.ECP_MAX_PROCESSES	maximum number of threads in the pool, the pool can not have more threads than this value, even if every thread is running an application instance, valid value is a String or Integer representing a positive number more than MIN_PROCESSES
XapConstants.ECP_IDLE_TIME_BEFORE_END	waiting time before killing an idle thread, at the condition that thread count is not out of bounds, valid value is of type String or Long

Share Space

Processing and data are represented in XAPE by XML documents. Every document belongs to a Space, a group of documents. There is two kinds of Spaces, shared or private. Every Application Instance has its own Private Space, containing its processing document in XAPE-APP, its Context document and any other document bound by Instance processing instruction such as <bind-document>, all accessible only by the Instance.
Additional Spaces can be created, they are called Share Spaces, because documents belonging to such Space are accessible to any Application loaded. The method addShareSpace() of XapEngine permits to create a share space.

Start/stop

Starting XAP Engine consists in creating the pool of thread, then launching Applications loaded with XapConstants.ACP_AUTO_LAUNCH present in their loading parameters. This operation is performed by startup() method. Method shutdown() sends a shutdown system message (see below) to all running Instances.

Application loading

loadApplication() deploys a XAPE Application. The Application is represented by a document in XAPE-APP dialect. On loading, it is given a name, as unique ID inside XAP Engine. Loading operation implies several deployment parameters, provided in a Java Map :

Application loading parameter key	Description
XapConstants.ACP_MULTI_INSTANCES	required, tells whether this application can be instanciated multiple times, or just once. Value is a Boolean or a String : if false, only on instance is possible ; in this case, instance name is not required to identify it
XapConstants.ACP_AUTO_LAUNCH	optional, if this parameter is present, means to launch automatically an instance at Engine startup ; its value is then the name to give to the instance launched ; if application is mono-instance, the name is not taken in consideration
XapConstants.ACP_EXEC_MODE	Required, an Application Instance can be executed in its own dedicated thread, from Engine thread pool, or executed in the thread calling either sendRequest() or startup(), the two methods able to start an Instance or reactivate an Instance waiting for a request to continue its processing. Choose XapConstants.IN_DEDICATED_THREAD or XapConstants.IN_CALLERS_THREAD
XapConstants.ACP_BASE_URL	Optional, the base URL used to resolve relative URLs with XAPE runtime environment's method resolveUrl(). Valid value is an absolute URL object.
XapConstants.ACP_START_VARIABLES	Optional, this parameter represents a java Map containing XPath variables to add to XAPE runtime environment before runnig each new Instance. The Map stores couples of variable name and value. Each pair of key value is added like if using addXPathVariable() method of runtime environment.

Sending message

The unique way to communicate with an Instance from outside XAPE environment is to call sendRequest(), which needs to know Instance target by its Application and Instance names. This is also the way to create a new Instance : if target instance doesn't exists (and if Application is multi-instances), it is created before processing request.
This method uses two important objects, XapRequest and XapResponse, which are XO objects.

Format

A XAPE message is expressed in XML, as an XML document portion represented in Xotics Java API as an XO tree rooted by a XapRequest object. Here is an example of XAPE message XML structure (attributes are not shown) :

```
<ctx:request>
  <ctx:content>
    <any:contentObject>
  </ctx:content>
  <ctx:response>
    <any:response-content>
  </ctx:response>
</ctx:request>
```

The namespace ctx groups elements required to define a XAPE message. They come from XAPE Context dialect, described below.
<content> can contain the XML description of any request content ; it's up to the developer to design/create such content description elements. In XAPE servlet chapter there is an example of such specific description : request content is an XML description of a J2EE servlet request. With Xotics Editor, such request content can be an XML representation of Swing events.
<response> contains an XML representation of a response. Like for request content description, response format is totally free. <response> (aka XapResponse implementation) is just a container.
On each request received, an Instance integrates this subtree into its private Context document. Processing instructions can extract data from this request tree by XPath requests, then can build an XML response by adding custom elements as children of <response>.

XapRequest and XapResponse classes

XapRequest and XapResponse are XO objects in package com.virtualweaver.xotics.dialect.xape.engine. Both have common properties id and type inherited from XapMessage XO object. Here is a description of XapRequest and XapResponse properties (column Q/R means Q for XapRequest, R for XapResponse):

Property	Type	Q/R	Description
id	String	Q/R	ID of the request/message
type	byte	Q/R	Type of message (oneway, synchronous, asynchronous) as explain below
system	boolean	Q	A message can be addressed to XAP Engine itself, then it is called system
appName	String	Q	Target Application name
instanceName	String	Q	Target Instance name
handler	String	Q	If the message is not system, this is the ID of a <request-handler> element (XAPE-APP dialect) of target Instance processing document ; if specified, this handler is searched and executed if found. If the message is system, this property contains the type of system request : XapConstants. SYSTEM_MESSAGE_JUST_START for startup, XapConstants. SYSTEM_MESSAGE_SHUTDOWN for shutdown
responseQueue	XapQueue	Q	Message queue for receiving response, used for asynchronous message.

There are three kinds of requests :

Type of XAPE request	Description
XapConstants. REQ_TYPE_ONEWAY	Request expecting no response ; method sendRequest() returns null.
XapConstants. REQ_TYPE_SYNC	Request expecting a response ; sendRequest() waits until the response occurs, then returns a XapResponse object containing the response data.
XapConstants. REQ_TYPE_ASYNC	sendRequest() returns immediately, response will arrive in XapRequest.responseQueue object. Notice that this kind of request is possible only if target Application Instance is running in mode XapConstants.IN_DEDICATED_THREAD (i.e. in its own dedicated thread)

Creation

Even if XapRequest object and its subtree can be created by classical Xotics way (using XoRegistry.createXoObject() and XoUtilities.addChild()), it is recommended to use message creation methods of XapEngine :

- createRequest() to create request to an Instance target ; contentObject is a subtree representing effective request content data, created by Xotics classical way ;
- createShutdownMessage() to send a shutdown request towards a specific Instance ; target Instance jumps to <state-shutdown> if present or <state-end> otherwise ;
- createStartupMessage() to request the creation and start of a new Application Instance. A target instance is automatically created upon reception of any message ; this method creates a message asking just for launching a new Instance.

Note : a XAPE request is a document portion : there is no need (and no possibility) to create an XoDMInstance to contains this tree. This document portion is to be integrated intoan existing DM instance, the Private Context.

XAPE-Context

The Context is a private document bound to every Application Instance, containing current request subtree and a space to store any data. Here is an example describing its general structure, extract of XAPE Servlet :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<ctx:context
      xmlns:ns1="http://www.virtualweaver.com/XAPE/servlet/servlet200.dtd"
      xmlns:ctx="http://www.virtualweaver.com/XAPE/context/context200.dtd" >

  <ctx:request appName="App1" id="BC67#82" instanceName="BC67" type="SYNC">
    <ctx:content>
      <ns1:http-request
           method="GET"
           session="BC67"
           host="www"
           port="8090"
           path="/"
           secure="false">
        <ns1:cookie name="JSESSIONID">BC67</ns1:cookie>
        <ns1:header name="accept" >image/gif, */*</ns1:header>
        <ns1:header name="accept-language" >us</ns1:header>
        <ns1:header name="accept-encoding" >gzip, deflate</ns1:header>
        <ns1:header name="user-agent" >Mozilla/4.0</ns1:header>
        <ns1:header name="host" >www</ns1:header>
        <ns1:header name="connection" >Keep-Alive</ns1:header>
        <ns1:header name="cookie">JSESSIONID=BC67</ns1:header>
      </ns1:http-request>
    </ctx:content>
    <ctx:response id="BC67#82" type="SYNC" >
      <ns1:http-response contentType="text/html" status="0" />
    </ctx:response>
  </ctx:request>
  <ctx:bag>
    <ctx:file name="log" file="/usr/local/log1/log.dat"
  /ctx:bag>
</ctx:context>
```

Element description

Though XAPE-Context dialect is not designed to be stored as XML text on disk, it is a standard Xotics implementation, and described below as is. However, some properties can not store their value in XML text and so loose information in a saving operation.

Element <context>

This is the root element of a Private Context document. It has two parts : current request description and a multi-purpose space called bag, where Instance can add any XO object structure. Both are optional (an Instance can run without request), but <bag> is automatically created with Private Context.

Content-Model :
(request?, bag)

Element <request>

This element represents the current XAPE request. It is implemented by XapRequest XO object, and generally built with XapEngine.createRequest(). <request> has two main parts, the request content, expressed by custom XML structure, and a response container in order for Instance to add response data.

Content-Model :	
(content?, response?)	
Attributes :	
id	ID of current message, both to identify a request and to find corresponding response object (in a response queue) which has the same ID.
type	A value to choose among "SYNC", "ASYNC", "ONEWAY" attribute values
system	If true, this is a system message. There is two types of system messages : shutdown or startup
appName	Target Application name
instanceName	Target Instance name
handler	If @system is true, this takes the type of system message : Value is "system.JustStart" for startup and "system.Shutdown" for shutdown. If @system is false, this is optional and represent the ID of <request-handler> element of current processing state : if such element is found, it is directly processed, without evaluating other request handlers. If not found, nothing is done
responseQueue	This attribute is not storable as XML, but just used at runtime in a Java execution.

Element <content>

This element is the container for request content subtree, which can be any custom XML structure suitable to current application. Xotics Editor, powered by XAPE, receives as <content> child a description of events (Swing or not). XapServlet receives under <content> an XML representation of a J2EE HTTP servlet request. <content> is not required to be present, but if a request content subtree is provided when creating XapRequest structure, this must be the container.

Content-Model :
(ANY)+ any XO object or derived representing request extra data

Element <response>

This element represents the response possibly expected from a current XAPE request. It is implemented by XapResponse XO object, and generally built with XapEngine.createRequest(), as child of <request>/XapRequest element/object. <response> attributes are copied from its <request> container.

Content-Model :	
(ANY)? any XO object or derived representing response data	
Attributes :	
id	ID of current message, both to identify a request and to find corresponding response object (in a response queue) which has the same ID.
type	A value to choose among "SYNC", "ASYNC", "ONEWAY" attribute values

Element <bag>

This element is a general purpose container to hold any user XMLstructure, any user XO objects, used by Instance processing document.

Content-Model :
(ANY)? any XO object or derived, and specifically boolean, list, integer, string, double, object, file, text, exception, url elements

Elements <boolean>, <integer>, <double>, <string>, <object>

These elements serve to store basic objects. They can be added to the Bag to contain custom application data.

Attributes :	
name	Unique ID of element inside Private Context
type	A value of type depending on the element : boolean, int, double, String, Object

Element <list>

A simple named container to store any XO object in Context Bag.

Content-Model :	
(ANY)? any XO object or derived	
Attributes :	
name	Unique ID of element inside Private Context

Element <file>

An element to hold a java.io.File object.

Attributes :	
name	Unique ID of element inside Private Context
file	A value of type File

Element <url>

An element to hold a java.net.URL object.

Attributes :	
name	Unique ID of element inside Private Context
file	A value of type URL

Element <exception>

An element to hold a java.lang.Exception object.

Attributes :	
name	Unique ID of element inside Private Context
className	Class name of the exception
message	The message of the exception
value	Value containing the exception object, of type Exception

Element <text>

An element to hold a big text.

Attributes :	
name	Unique ID of element inside Private Context
PCDATA	Text content

XAP Player

XAPE Player is an executable extension of XAPE Engine. It uses an XML document to configure itself an deploy Applications. The player is moslty a XAPE Engine wrapper with enhanced features. It can be instanciated by its constructor to be used in any user program, or be executed via its main() method.

XapPlayer class

Here are the methods of XAPE Player class :

```
package com.virtualweaver.xotics.dialect.xape.player;

public XapPlayer(XoEnvironment env, URL initFile) throws XoException,
XapException;
public void startup() throws XapException;
public void shutdown() throws XapException;
public XapRepository getRepository();
public XapResponse sendRequest(XapRequest req) throws XapException;
public static void main(String[] args);
```

The player is launched either by running it in a main() method or by creating new instance with its unique constructor. With the constructor, the player can be integrated in any user program. The Servlet version of XAPE Engine is based on XapPlayer. Configuration from XML config file is set up by the constructor.
Executable version currently accepts as single argument the URL of XML config file.
XapPlayer execution can be controlled by Engine methods startup() and shutdown().
getRepository() returns an object informing about current loaded applications and running instances.
Method sendRequest() stays the unique way to communicate with application instances.

Player Configuration dialect (XAPE-Config)

This is the XML dialect used to configure XAPE Player and deploy XAPE Applications.

Overview

Here is an example of configuration document, the one used to launch Xotics Editor (just arranged to fit this page format) :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<config xmlns="http://www.virtualweaver.com/XAPE/config/config200.dtd">
  <load-dialect  dmcp="/com/virtualweaver/.../IMLDef.xml"/>
  <load-dialect dmcp="/com/virtualweaver/xotics/.../XepDef.xml"/>
  <deploy>
    <application name="EditorMainApp"
                 url="/data/XoEditorMainApp.xml"
                 baseUrl="/data/" autoLaunch="true"
                 multiInstances="false"
                 execMode="IN_DEDICATED_THREAD">
      <init-var name="ProfileBaseDir">fn:string('/usr/local/')</init-var>
    </application>
  </deploy>
</config>
```

Root element of the config document is <config>. Here, the document starts to specify which Xotics dialect implementations are needed to deploy applications. Then, each application deployment is described under element <deploy>.

Element description

Element <config>

Root element of a XAPE Player configuration document. Under this element are described :

- which dialects to load,
- specific Engine parameters,
- new share spaces,
- application deployments.

Content-Model :
(load-dialect*, param*, share-space*, deploy*)

Element <load-dialect>

Loads a Xotics dialect implementation in current Xotics environment. There is two ways to specify target dialect :

- either by URL of the Xotics JAR containing implementation classes (a JAR containing "Definition-Path:" special entry in its manifest) ;
- or by the classpath of the DMDL document of the dialect implementation.

These two attributes are mutually exclusive.

Attributes :	
dmcp	Classpath of the DMDL document defining Xotics dialect implementation
url	Location of the Xotics dialect implementation JAR

Element <param>

Specifies a XAPE Engine init parameter value. Currently, these parameters configure the Thread pool included in XAPE Engine, precisely thread count boundaries in the pool and latency time before killing an idle thread between min and max boundaries.

- MIN_PROCESSES : minimum number of threads in the pool, thread count can not be under this value, even if every thread is idle,
- MAX_PROCESSES : maximum number of threads in the pool, the pool can not have more threads than this value, even if every thread is running an application instance,
- IDLE_TIME : waiting time before killing an idle thread, at the condition that thread count is not out of bounds.

Attributes :	
name	Name of the parameter, a string whose value can be either MIN_PROCESSES, MAX_PROCESSES or IDLE_TIME.
PCDATA	Positive integer value, IDLE_TIME is expressed in milliseconds.

Element <share-space>

Creates a new share space before any Application Instance creation.

Attributes :	
name	Name of the share space to create

Element <deploy>

used to deploy a XAPE Application. This element is a container for one or several <application> elements. <application> element can be either a child of <deploy> or can be the root element of an external document. In this case, <deploy> uses @url to locate external document containing <application> description.
There can be as many <deploy> elements (children of <config>) as needed, though not really interesting.

Content-Model :	
(application+)	
Attributes :	
url	Optional, if <deploy> has no child, this attribute locates an external document containing <application> as root element.

Element <application>

Describes a XAPE Application to deploy. This element can be child of <deploy> element or the root element of a document accessed by @url of <deploy>.
For every Application, XAP Engine has the ability to create as much instances as needed or just one unique instance. The first case is called multi-instanciation and second case mono-instanciation. When loading a XAPE application, XAP Engine must know if this application will be mono-instance (that is, will be instanciated once for all) or multi-instance (instanciated as needed). This is done by @multiInstance.
An application instance can be created and started at Engine startup. This is specified by @autoLaunch. If the application is multi-instance, @instanceName must be specified to set the name of the instance to start automatically. By default, instance name of a mono-instance application is the same as application name.
There are two ways to execute an instance, depending on which thread is going to run the instance ; an instance can be executed :

- either by a dedicated thread, from the thread pool of the Engine,
- or by a user's thread, which performs processing at each message sending or at startup ; the thread is the one calling XapPlayer sendRequest() or startup().

Such execution mode is chosen at application loading time and can not be changed. @execMode selects this option among values IN_DEDICATED_THREAD or IN_CALLERS_THREAD.

Content-Model :	
(init-var| load-dialect)*	
Attributes :	
url	Require location of processing document in XAPE-APP dialect
name	Required ID of the application in XAPE Engine (must be unique)
multiInstance	Boolean to specify whether application is instanciated only once or instanciated as needed
autoLaunch	Boolean indicating to start an application instance automatically at startup
instanceName	an optional instance name which can be used to identify instance of a mono-instance application (by default, instance name for mono-instance application is the same as its application name), or the instance of a multi-instance application if @autoLaunch is true.
baseUrl	The base URL to resolve relative URLs in the application
execMode	A constant which can be either IN_DEDICATED_THREAD or IN_CALLERS_THREAD

Element <init-var>

Creates an XPath variable and stores it into the runtime environment of each instance at creation time. It can be used as init variable for application document.

Attributes :	
name	Required name of the variable
PCDATA	Required Xpath expression whose result is the value

XAPE Servlet

This is a J2EE Servlet version of XAP Engine. Every class or interface involved in this version is in the sub package servlet. Its main class is XapServlet, derived from HttpServlet, and containing a XAPE Player and a way to exchange data between HTTP client and XAPE Application.

Overview

Goals of XapServlet are to run a XapPlayer instance and to provide an interface between the player and the servlet request/response format. This is done with a special Java interface, XapServletRequestManager, which have a method for converting an HttpServletRequest into a XapRequest and another method for converting a XapResponse into an HttpServletResponse. A default implementation is provided, XapDefaultRequestManager, but developer can create other versions.
XapServletRequestManager can also use objects implementing XapServletRequestContentHandler interface, which are able to convert a specific input content designated by MIME type into an XML structure added as child of <content> in a XapRequest.
These different classes and other configuration informations are described in a XAPE Servlet config document, in XAPE-Servlet dialect.
If a XAP Application running in XAPE Servlet is multi-instances, Instance is created with each new HTTP session, to manage requests from that session. The name of the Instance is the session ID. It's life time is the same as its HTTP session. So, in order for XAP Engine to be informed about session ending, a specific session listener class, XapServletSessionListener, must be specified in the Servlet deployment file web.xml.

Deployment of XAPE servlet

Here below is the format of the web.xml file to provide when deploying XAPE applications with XAPE servlet. This web.xml file contains required informations only, you can add yours.

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

<!DOCTYPE web-app
    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
    "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>
 <display-name>{your display name}</display-name>
 <description>{your description}</description>

 <listener>
       <listener-class>
com.virtualweaver.xotics.dialect.xape.servlet.XapServletSessionListener
       </listener-class>
 </listener>
    <servlet>
      <servlet-name>XAPE_servlet</servlet-name>
      <servlet-class>
com.virtualweaver.xotics.dialect.xape.servlet.XapServlet
      </servlet-class>
```

```
      <init-param>
        <param-name>XapInitConfigURL</param-name>
        <param-value>
{your URL to XAPE player config file}
      </param-value>
      </init-param>
      <init-param>
        <param-name>XapInitMappingsURL</param-name>
        <param-value>
{your URL to request mapping file}
</param-value>
      </init-param>
    </servlet>

    <servlet-mapping>
      <servlet-name>XAPE_servlet</servlet-name>
      <url-pattern>{your pattern}</url-pattern>
    </servlet-mapping>
</web-app>
```

As you can see, this file is a template, text enclosed by {} must be set by the user. So, the J2EE web application is composed with XapServlet HTTP servlet class, with some required init params :

- XapPlayerConfigURL : URL of included XAPE Player Config file, which must contain application deployment directives ;
- XapServletConfigURL : URL of servlet config document, written in XAPE Servlet dialect ;
- XapInitRequestManagerClass : full class name of a specific request manager (XapServletRequestManager java interface), optional because XAPE servlet can use a default implementation (XapDefaultRequestManager).

A session listener is also specified ; XapServletSessionListener informs XAP Engine about session ending. When session ends, corresponding Instance (if from multi-instance Application) is required to end too.
To deploy a web app powered by XapServlet, copy this web.xml file in WEB-INF/ directory and replace parts in {}. This web.xml file is compliant to J2EE Servlet 2.3 specs and has been tested with Apache Tomcat on various plateforms.

Servlet Config dialect

Overview

This XML dialect describes two kinds of mapping needed to initialize XAPE servlet :

- mapping between a request URI (part of request URL between server and port and query string) and a target Application and optional direct handler ;
- mapping between a MIME type of request content and a XapServletRequestContentHandler object ;

First kind of mapping is required ; this is the only way to find a target application. MIME mapping is optional, XapServletRequestManager implementations can do all the parsing work.

A document in this dialect must be specified, with servlet init parameter XapServletConfigURL, to configure XapServlet.

Note : as you will see later in this XAPE Servlet chapter, this dialect contains also elements for XAPE request building. Entire dialect is also called XAPE Servlet dialect.

Here is an example of such document :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xape-servlet-config
xmlns="http://www.virtualweaver.com/XAPE/servlet/servlet200.dtd" >
  <request-mappings >
    <dispatch app="DynUpdater" path="/update" />
    <dispatch app="StaticRes" path="/img/*" />
    <dispatch app="StaticRes" path="*.xml" requestHandler=”xml_handler”/>
    <default-dispatch app="MainApp" />
  </request-mappings>
  <content-handlers>
    <content-handler contentType=”application/zip”
                     handlerClass=”com.dum.ZipHandler”/>
    <content-handler contentType=”text/xml”
                     handlerClass=”com.dum.XmlHandler”/>
    <default-content-handler handlerClass=”com.dum.DefaultHandler”/>
  </content-handlers>
</xape-servlet-config>
```

Element <xape-servlet-config> is always the root element of such document. <request-mappings> subtree is required ; it contains an ordered list of dispatching directives depending on the form of request URL path part. As you can see, wildcards are allowed in @path. Each <dispatch> element is evaluated in XML document order and target Application is selected as soon as input request path matches @path expression. It is also possible to select a direct request handler with dispatch @requestHandler. If no target Application is found, a possible <default-dispatch> selects Application receiving the request.
<content-handlers> is an optional container of <content-handler> elements, each specifying a particular XapServletRequestContentHandler class to parse a request whose content is of MIME type @contentType. This list is converted into a Java Map object and passed to method setRequestContentHandlers() of XapServletRequestManager interface.

Element description

This dialect has a wider usage than config purpose only. We describe here only elements involved in XAPE Servlet configuration.

Element <xape-servlet-config>

This is the root element of a XAPE servlet config document.

Content-Model :
(request-mappings, content-handlers?)

Element <request-mappings>

This is the container of <dispatch> elements. Notice that these elements are analized in document order. If no match is found, target Application specified in <default-dispatch> is returned, if this element is present.

Content-Model :
(dispatch*, default-dispatch?)

Element <dispatch>, <default-dispatch>

Gives the target Application to send the request if HTTP request URI (part of the request URL between server/port and query string) matches pattern in @path. This pattern allows wildcard '*' chars.

Here are some examples :

@path	Request URI	Matching ?
/img/	/req/img/im1.gif	yes
/img/	/img/im1.gif	yes
/doc*.xml	/req/doc1.xml	no
/doc*.xml	/doc12.xml	yes

<dispatch> elements are evaluated in document order. First matching stops evaluation. If no matching is found, <default-dispatch> provides target Application. <default-dispatch> is optional.

Attributes :	
path	Path pattern containing possible '*' wildcards. If request URI matches this pattern, corresponding <diaptch> element is selected. Required for <dispatch>, absent from <default-dispatch>.
app	Target Application name. Required for both <dispath> and <default-dispatch> elements.
requestHandler	Direct request handler of a target Application. This attribute is optional and concerns <dispatch> only. If present, a XAPE-APP <request-handler> element having @requestHandler as ID is searched in current <state> and processed if found.

Element <content-handlers>

Container of <content-handler> elements.

Content-Model :
(content-handler*, default-content-handler?)

Element <content-handler>, <default-content-handler>

<content-handler> provides a XapServletRequestContentHandler class to parse a request content of MIME type @contentType. <default-content-handler> returns the class to use to parse content if no other class is found. This directive is optional.
At init, XapServlet creates a Java Map from these mappings. Keys are MIME types and values are objects instanciated from XapServletRequestContentHandler classes. If a default mapping is provided, its key is just the string "defaultContentHandler".

Attributes :	
contentType	Mime type of an HTTP request content. Required for <content-handler>, absent for <default-content-handler>.
handlerClass	XapServletRequestContentHandler class name to choose, required for both elements.

Request manager

It's an interface and a default implementation to manage conversion of request and response between HTTP client and XAPE Application.

XapServletRequestManager interface

```
public interface XapServletRequestManager {

  public void setXoEnvironment(XoEnvironment env);
  public void setRequestContentHandlers(Map contentHandlers) throws
XapException;
  public SrvHttpRequest parse(HttpServletRequest req) throws XapException;
 public void sendBack(SrvHttpResponse xr, HttpServletResponse resp) throws
XapException;
```

Method setRequestContentHandlers() provides a mapping between particular MIME type and object of type XapServletRequestContentHandler. Given this mapping, method parse() creates and returns a XAPE servlet request object, sent by XapServlet in a XapRequest object to appropriate Application Instance.
Method sendBack() takes a XAPE Servlet response object to send response to HTTP client, using HTTP Servlet response object.

XAPE API provides a default implementation for this request manager, detailed below, but you can provide your own request manager class, with your own XAPE request structuration.

Default implementation

XapDefaultRequestManager class is the default implementation for XapServletRequestManager interface. It creates a XapRequest from HttpServletRequest and configure an HttpServletResponse from a XapResponse content before using HttpServletResponse output stream to send back the response.

XAPE request/response format

XapDefaultRequestManager class uses several XO objects, defined in XAPE-Servlet dialect, to create a request/response subtree. Here is an example of a XapRequest structure :

```
... xmlns:srv="http://www.virtualweaver.com/XAPE/servlet/servlet200.dtd"
      xmlns:ctx="http://www.virtualweaver.com/XAPE/context/context200.dtd"

  <ctx:request appName="App1" id="BC67#82" instanceName="BC67" type="SYNC">
    <ctx:content>
      <srv:http-request
           method="GET"
           session="BC67"
           host="www"
           port="8090"
           path="/"
           secure="false">
        <srv:cookie name="JSESSIONID">BC67</srv:cookie>
        <srv:header name="accept" >image/gif, */*</srv:header>
        <srv:header name="accept-language" >us</srv:header>
        <srv:header name="accept-encoding" >gzip, deflate</srv:header>
        <srv:header name="user-agent" >Mozilla/4.0</srv:header>
        <srv:header name="host" >www</srv:header>
        <srv:header name="connection" >Keep-Alive</srv:header>
        <srv:header name="cookie">JSESSIONID=BC67</srv:header>
      </srv:http-request>
    </ctx:content>
    <ctx:response id="BC67#82" type="SYNC" >
      <srv:http-response contentType="text/html" status="0" />
    </ctx:response>
  </ctx:request>
```

Namespace ctx represents XAPE-Context dialect, especially XapRequest and XapResponse objects. Namespace srv designates XAPE-Servlet dialect, providing <http-request> container, which must be the container for any HTTP servlet request content, and <http-response> container which contains HTTP servlet response content to send back to the HTTP client. This dialect provides also <cookie> to store a cookie, <header> to hold an HTTP header and <parameter> for HTTP request parameter.

Element Description

Element <http-request>

This container, child of a XAPE-Context <content> element, holds an http request content. Its implementation class is com.virtualweaver.xotics.dialect.xape.servlet.model.SrvHttpRequest. It contains HTTP cookie and header elements, then either a list of HTTP parameters or a custom content representing other input.

Content-Model :	
(cookie*, header*, (parameter* | content))	
Attributes :	
host	A string containing then name of the server
port	int representing TCP port of the connection
session	ID string of current HTTP session
secure	Boolean indicating if the connection is secure (made using HTTPS)
path	Part of the URL between host/port and query string
method	String containing HTTP method, which can be GET, POST, PUT, etc ...

SrvHttpRequest class contains also a property 'request' of type HttpServletRequest, not published as XML attribute, containing current HTTP Servlet request object, making it accessible to any XapProcessable object.

Element <http-response>

This container, child of a XAPE-Context <response> element, holds an http response content. Its implementation class is com.virtualweaver.xotics.dialect.xape.servlet.model.SrvHttpResponse. It contains HTTP cookie and header elements, then an optional response content.

Content-Model :	
(cookie*, header*, content?)	
Attributes :	
content-type	MIME content type of the response
status	int representing HTTP response status
redirect	String representing an redirection URL

SrvHttpRequest class contains also three important properties, not published as XML attributes :

- 'request', of type HttpServletRequest, containing current HTTP Servlet request object, making it accessible to any XapProcessable object ;
- 'response', of type HttpServletResponse, containing current HTTP Servlet response object, making it accessible to any XapProcessable object ;
- 'responseStream' of type java.io.OutputStream, an alternative to send response.

With default request manager implementation, there are two ways to send a response. When receiving XAPE response, the manager first seeks for a child of <http-response>/<content> child.

If present, child of <content> is considered as the root element of an XML document to send as HTTP response. Thus, it must be an XoRoot object, in order to be saved into Servlet output stream.
If absent, it is assumed that property responseStream contains the octet stream of the response. It permits to send any content which is not XML.

Element <content>

This container, which can be child of a <http-request> or <http-response> element, holds an XML structure representing a request or response content.

Content-Model :
(ANY) a unique child, implementing XoRoot if <content> is child of <http-response>

Elements <cookie>, <header>, <parameter>

These elements represents (name, value) pair used to hold HTTP cookie, header and parameter.

Attributes :	
name	Name of the cookie, header, parameter
PCDATA	value of the cookie, header, parameter