



XML Framework

Java API Reference



Apple Computer, Inc.
© 1999 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., except to make a backup copy of any documentation provided on CD-ROM.

The Apple logo is a trademark of Apple Computer, Inc. Use of the “keyboard” Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple-labeled or Apple-licensed computers.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for typographical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Macintosh, and WebObjects are trademarks of Apple Computer, Inc., registered in the United States and other countries. Enterprise Objects is a trademark of Apple Computer, Inc.

NeXT, the NeXT logo, OPENSTEP, Enterprise Objects Framework, Objective-C, and WEBSOCKET are trademarks of NeXT Software, Inc.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

Helvetica and Palatino are registered trademarks of Linotype-Hell AG and/or its subsidiaries.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

ORACLE is a registered trademark of Oracle Corporation, Inc.

SYBASE is a registered trademark of Sybase, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

Windows NT is a trademark of Microsoft Corporation.

All other trademarks mentioned belong to their respective owners.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this manual, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD “AS IS,” AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

XML

Package: `com.apple.webobjects.xml`

Introduction

The XML framework consists of two main classes—WOXMLCoder and WOXMLDecoder—which can encode and decode objects as XML. These classes can be used to archive and unarchive object data, or to parse and/or generate XML obtained from or destined for an external source (such as the World Wide Web). When working with such “foreign” XML, you describe the XML elements and properties and their mapping to objects in an XML-format “mapping model” that you can create with either a text editor or an XML editor.

The mapping model provides greater control over the decoding process and is typically used when you are encoding and decoding XML that is destined for, or originates from, an external source. When the WOXMLCoder and WOXMLDecoder are used as an archiving mechanism, the mapping model is usually not necessary. See [“The Format of the Mapping Model”](#) (page 4) for more information on the contents and structure of the mapping model.

When archiving and unarchiving custom objects using the WOXMLCoder and WOXMLDecoder without a mapping model, your custom classes need to:

- implement the single method declared in the WOXMLCoding interface (your implementation of this method is where you encode the custom class’s instance variables)
- implement the single-argument constructor described in the WOXMLDecoder class introduction

You don’t need to do the above if the object you are archiving and unarchiving—as well as any encapsulated objects—are an instance of String, Number (or a subclass, providing that the subclass doesn’t add any new instance variables), NSArray, NSDictionary, NSDate, NSData, or EOEnterpriseObject (or a subclass, providing that all instance variables are either attributes or relationships). You also don’t need to implement the above if you are working with a mapping model.

Exceptions raised by the underlying SAX parser are, for simplicity’s sake, wrapped in a WOXMLException object, greatly reducing the number of exceptions your code needs to catch.

The Format of the Mapping Model

The mapping model is a text file—created manually with a text editor or an XML editor—that consists of one or more `entity` elements, each of which can enclose zero or more `property` elements, all enclosed within a single `model` element.

The following is a simple mapping model:

```
<model>
  <entity name="Command" xmlTag="command">
    <property name="qty" xmlTag="quantity" attribute="YES"/>
    <property name="movie" xmlTag="movie"/>
    <property name="customer" xmlTag="customer"/>
  </entity>
  <entity name="MyMovie" xmlTag="movie">
    <property name="title" xmlTag="name" attribute="YES"/>
    <property name="dateReleased" xmlTag="date">
    <property name="roles" xmlTag="role">
    <property name="category" xmlTag="cat"/>
  </entity>
  <entity name="com.apple.yellow.foundation.EOGenericRecord" xmlTag="role">
    <property name="roleName" xmlTag="name" attribute="YES"/>
  </entity>
</model>
```

When creating a mapping model, be aware that mappings must be unique for a given property when decoding (that is, you cannot have two mappings for the same property). The same applies for XML tags when encoding: you cannot have two mappings for the same XML tag.

The “model” Tag

The `model` tag has no attributes.

The “entity” Tag

The `entity` tag has two required attributes and a number of optional ones:

`name=property`

(required) The name of the property on the key-value coding side (the name of the object, attribute, or dictionary key to which the XML data is to be mapped).

FRAMEWORK XML

`xmlTag=tag`

(required) The XML tag.

`unmappedTagsKey=key`

(optional) The name of the property on the key-value coding side to be used when, while parsing the XML for the entity, a tag is encountered for which there is no specified mapping in the mapping model.

`ignoreUnmappedTags="YES" | "NO"`

(optional) “YES” causes a `WOXMLException` to be thrown if `unmappedTagsKey` isn’t specified and, while parsing the XML for the entity, a tag is encountered for which there is no specified mapping in the mapping model. Note that an exception isn’t thrown if the object being decoded is an `NSDictionary`.

`contentsKey=key`

(optional) The property name to be used for text enclosed by the XML element being parsed. For example, if the mapping model contained the following:

```
...
<entity name="com.apple.yellow.foundation.NSMutableDictionary" xmlTag="text"
contentsKey="contents"/>
...
```

And if the following was encountered while parsing XML:

```
<text>Hello, <b>World!</b></text>
```

An `NSMutableDictionary` object would be created with a single key-value pair: the key “contents” would have as its value an array containing two elements: the string “Hello, “ and a dictionary with a single key-value pair: the key “b” would have as its value the string “World”.

The “property” Tag

Within an `entity` element, you can have zero or more `property` elements. The `property` tag also has two required attributes and a number that are optional:

`name=property`

(required) The name of the property on the key-value coding side (the name of the object, attribute, or dictionary key to which the XML data is to be mapped).

`xmlTag=tag`

(required) The XML tag.

FRAMEWORK XML

`attribute="YES" | "NO"`

(optional) Used during encoding, “YES” causes a given property “a” to be encoded like:

```
<foo a="someValue"> .. </foo>
```

instead of:

```
<foo> <a>someValue</a> .. </foo>
```

`forceList="YES" | "NO"`

(optional) Used for decoding only. “YES” causes a single enclosed element to be decoded as an NSMutableArray with a single object of the appropriate type.

Decoding an XML structure like the following:

```
<foo>
  <a>value1</a>
  <a>value2</a>
</foo>
```

results in a key-value coding call with key “foo” and value an NSMutableArray containing the two “<a>” elements. When there is only a single enclosed element, as in this example:

```
<foo>
  <a>some Value</a>
</foo>
```

decoding will result in a key-value coding call with key “foo” and value an object of type “a”. `forceList` alters this default behavior, causing the value in this instance to be an NSMutableArray containing a single object of type “a”.

`codeBasedOn="TAG"`

(optional) The `WOXMLCoder` normally uses the value of the property’s `name` attribute as the key during key-value coding. This is the desired behavior in most situations since the `name` attribute typically indicates the name of an instance variable in a custom class. When working with `NSDictionary` objects, however, you may instead want to use the value of the `xmlTag` attribute as the key. To do this, specify `codeBasedOn="TAG"` in the property’s list of attributes.

`reportEmptyValues="YES" | "NO"`

(optional) When an empty element is encountered while decoding XML (for instance, `<myElement></myElement>`), the `WOXMLDecoder` normally creates an empty `NSDictionary` object. Setting `reportEmptyValues` to “NO” prevents this empty object from being created.

FRAMEWORK XML

`outputTags="class" | "property" | "both" | "neither"`

(optional) Used when encoding, this attribute specifies which XML tag specified in the mapping model should be output for the given property. “property”, the default, specifies that the property’s XML tag should be output. “class” specifies that the XML tag associated with the property’s enclosing class should be output instead. “both” indicates that both the property and the class tags should be output, and “neither” indicates that neither XML tag should be placed in the XML.

To illustrate the use of the `outputTags` attribute, the following mapping model could be used to produce HTML tags from an `NSMutableDictionary` object:

```
<model>
  <entity name="com.apple.yellow.foundation.NSMutableDictionary"
xmlTag="text">
  <property name="p" xmlTag="ignore" outputTags="neither"/>
</entity>
  <entity name="Paragraph" xmlTag="p">
  <property name="text.contents" xmlTag="text" outputTags="class"/>
</entity>
  <entity name="ItemList" xmlTag="ul">
  <property name="contents" xmlTag="li" outputTags="both"/>
</entity>
  <entity name="Item" xmlTag="item">
  <property name="contents" xmlTag="ignore" outputTags="class"/>
</entity>
  <entity name="LineBreak" xmlTag="br"/>
  <entity name="ExternalLink" xmlTag="a">
  <property name="url" xmlTag="href" attribute="YES"/>
  <property name="anchorText" xmlTag="ignore" outputTags="neither"/>
</entity>
</model>
```

FRAMEWORK XML

WOXMLCoder

Inherits from: Object

Package: com.webobjects.appserver.xml

Class Description

Use this class to encode objects as XML. Encoding can take place either with or without a mapping model. The mapping model provides greater control over the encoding process and is typically used when you are encoding and decoding XML that is destined for, or originates from, an external source. When the `WOXMLCoder` and `WOXMLDecoder` are used as an archiving mechanism, the mapping model is usually not necessary. For more information on the mapping model, see the [“The Format of the Mapping Model”](#) (page 4) in the framework introduction.

When encoding without a mapping model, `WOXMLCoder` is able to encode any object as long as the object and all of the objects it encapsulates either implement the `WOXMLCoding` interface or are an instance of `String`, `Number` (or a subclass, providing that the subclass doesn't add any new instance variables), `NSArray`, `NSDictionary`, `NSDate`, `NSData`, or `EOEnterpriseObject` (or a subclass, providing that all instance variables are either attributes or relationships). During the encoding of an enterprise object, `WOXMLCoder` uses attribute information stored in the `EOModel` when assigning an XML type tag to an object. For objects that don't inherit from `EOEnterpriseObject`, the tag supplied by `WOXMLCoder`'s `encodeObjectForKey` method is used.

To encode an object, simply invoke the `encodeRootObjectForKey` method. To perform the reverse operation, generating an object from XML data, see the `WOXMLDecoder` class.

Method Types

Creating a WOXMLCoder

`coder`

`coderWithMapping`

Encoding an object graph

`encodeRootObjectForKey`

Implementing the WOXMLCoding interface

`encodeBooleanForKey`

`encodeDoubleForKey`

`encodeFloatForKey`

`encodeIntForKey`

`encodeObjectForKey`

Constructors

WOXMLCoder

`protected WOXMLCoder`

Description forthcoming.

Static Methods

coder

```
public static WOXMLCoder coder()
```

Creates and returns a new WOXMLCoder object.

coderWithMapping

```
public static WOXMLCoder coderWithMapping(String mappingURL)
```

Creates and returns a new WOXMLCoder object initialized with the mapping model specified by `mappingURL`. See [“The Format of the Mapping Model”](#) (page 4) for a complete description of the mapping model.

Note: Windows NT uses backslashes where other systems use forward slashes. When prepending the “file:” URL prefix to a path such as is returned by WResourceManager’s `pathForResourceNamed` method, on Windows NT the prefix must be “file:\\” while on all other platforms the prefix must be “file:///”. See the RelatedLinks example for one way to select the proper prefix based upon the underlying system.

Instance Methods

cr

```
protected void cr()
```

Description forthcoming.

CLASS WOXMLCoder

encodeBooleanForKey

```
public void encodeBooleanForKey(  
    boolean flag,  
    String key)
```

Invoke from within in your implementation of `WOXMLCoding`'s `encodeWithWOXMLCoder` method to append an element with XML tag `key` to the `WOXMLCoder` object's internal string buffer. The element's XML content is the string representation of `flag`—either `True` or `False`—and the element has an attribute named `type` with a value of `boolean`. For example, the following call to `encodeBooleanForKey`:

```
encodeBooleanForKey(true, "myTag");
```

causes the following text to be appended to the `WOXMLCoder`'s internal string buffer:

```
<myTag type="boolean">True</myTag>
```

encodedClassName

```
protected String encodedClassName()
```

Description forthcoming.

encodeDoubleForKey

```
public void encodeDoubleForKey(  
    double aDouble,  
    String key)
```

Invoke from within in your implementation of `WOXMLCoding`'s `encodeWithWOXMLCoder` method to append an element of type `key` to the `WOXMLCoder` object's internal string buffer. The element's content is the string value of `aDouble` and the element has an attribute named `type` with a value of `double`. For example, the following call to `encodeDoubleForKey`:

```
encodeDoubleForKey(1.23, "myTag");
```

causes the following text to be appended to the `WOXMLCoder`'s internal string buffer:

```
<myTag type="double">1.23</myTag>
```

CLASS WOXMLCoder

encodeFloatForKey

```
public void encodeFloatForKey(  
    float aFloat,  
    String key)
```

Invoke from within in your implementation of WOXMLCoding's `encodeWithWOXMLCoder` method to append an element of type `key` to the WOXMLCoder object's internal string buffer. The element's content is the string value of `aFloat` and the element has an attribute named `type` with a value of `float`. For example, the following call to `encodeFloatForKey`:

```
encodeFloatForKey(1.23, "myTag");
```

causes the following text to be appended to the WOXMLCoder's internal string buffer:

```
<myTag type="float">1.23</myTag>
```

encodeIntForKey

```
public void encodeIntForKey(  
    int anInt,  
    String key)
```

Invoke from within in your implementation of WOXMLCoding's `encodeWithWOXMLCoder` method to append an element of type `key` to the WOXMLCoder object's internal string buffer. The element's content is the string value of `anInt` and the element has an attribute named `type` with a value of `int`. For example, the following call to `encodeIntForKey`:

```
encodeIntForKey(123, "myTag");
```

causes the following text to be appended to the WOXMLCoder's internal string buffer:

```
<myTag type="int">123</myTag>
```

CLASS WOXMLCoder

encodeObjectForKey

```
public void encodeObjectForKey(  
    Object anObject,  
    String key)
```

Invoke from within in your implementation of WOXMLCoding's `encodeWithWOXMLCoder` method to append an element of type `key` to the WOXMLCoder object's internal string buffer. The element's content depends on `anObject`'s class. `anObject` must meet the same criteria outlined in `encodeRootObjectForKey`.

`encodeRootObjectForKey` relies upon this method to perform the actual encoding of objects.

encodedObjectInTag

```
protected void encodeObjectInTag(  
    String,  
    String,  
    String)
```

Description forthcoming.

encodeReferenceInTag

```
protected void encodeReferenceInTag(  
    int,  
    String,  
    String)
```

Description forthcoming.

encodeRootObjectForKey

```
public synchronized String encodeRootObjectForKeyForKey(  
    Object anObject,  
    String key)
```

Encodes `anObject` as XML and returns the resulting XML string. The encoded root object is tagged using `key`, and has a `type` attribute that indicates `anObject`'s class. `anObject` must be one of the following:

- an instance of `String`

CLASS `WOXMLCoder`

- an instance of `NSArray`
- an instance of `NSDictionary`
- an instance of `NSDate`
- an instance of `Number` (or a subclass, providing that the subclass doesn't add instance variables)
- an instance of `NSData`
- an object that implements the `WOXMLCoding` interface
- an instance of `EOEnterpriseObject` (or a subclass, providing that all instance variables are either attributes or relationships)

If `anObject` is not one of the above, `encodeRootObjectForKey` throws an exception.

encodeStringInTag

```
protected void encodeStringInTag(  
    String,  
    String,  
    String)
```

Description forthcoming.

escapeString

```
protected String escapeString(String escapeString)
```

Description forthcoming.

typeNeedsIndentation

```
protected boolean typeNeedsIndentation(Object)
```

Description forthcoming.

CLASS WOXMLCoder

xmlTagForClassNamed

protected String xmlTagForClassNamed(String className)

Description forthcoming.

xmlTagForPropertyKey

protected String xmlTagForPropertyKey(
String,
String)

Description forthcoming.

WOXMLDecoder

Inherits from:	Object
Implements:	NSKeyValueCoding
Package:	com.webobjects.appserver.xml

Class Description

Use this class to construct (“decode”) an object from XML data. Decoding can take place either with or without a mapping model. The mapping model provides greater control over the decoding process and is typically used when you are encoding and decoding XML that is destined for, or originates from, an external source. When the WOXMLCoder and WOXMLDecoder are used as an archiving mechanism, the mapping model is usually not necessary. For more information on the mapping model, see the [“The Format of the Mapping Model”](#) (page 4) in the framework introduction.

Decoding XML Without a Mapping Model

On its own, without a mapping model, WOXMLDecoder is able to decode any object as long as the object and all of its children either implement the WOXMLCoding interface or are an instance of String, Number (or a subclass, provided that the subclass doesn’t add instance variables), NSArray, NSDictionary, NSDate, NSData, or EOEnterpriseObject (or a subclass, providing that all instance variables are either attributes or relationships). To construct an object from XML data, invoke one of the `decodeRootObject` methods.

CLASS WOXMLDecoder

Objects that implement the `WOXMLCoding` interface must also implement a single-argument constructor that takes a `WOXMLDecoder` object as the single argument if they are to be decoded. Within this constructor you decode your object's instance variables using `WOXMLDecoder`'s various `decode...ForKey` methods.

The following simple "Person" class implements both the single-argument constructor needed to decode objects of this class and the `WOXMLCoding` interface.

```
import com.apple.webobjects.xml.*;
import com.apple.yellow.foundation.*;
import java.lang.*;
import java.net.*;
import java.math.*;

public class Person extends Object implements WOXMLCoding {
    String name;
    boolean married;
    int children;

    public Person() {
        name = "John Smith";
        married = true;
        children = 2;
    }

    public void encodeWithWOXMLCoder(WOXMLCoder coder) {
        coder.encodeObjectForKey(name, "Name");
        coder.encodeBooleanForKey(married, "MaritalStatus");
        coder.encodeIntForKey(children, "NumberOfChildren");
    }

    // constructor required for decoding
    public Person(WOXMLDecoder decoder) {
        name = (String)decoder.decodeObjectForKey("Name");
        married = decoder.decodeBooleanForKey("MaritalStatus");
        children = decoder.decodeIntForKey("NumberOfChildren");
    }
}
```

See the [XMLArchiving](#) example (accessible through the [WebObjects Info Center](#) under [Examples > WebObjects > Java > XMLArchiving](#)) for a more complete example.

Decoding XML With a Mapping Model

The mapping decoder gives you much greater control over the decoding process, since it operates under the direction of an XML-format mapping model that you create. This mapping model allows you to specify how XML elements and attributes are to be mapped to objects and object attributes (the mapping is performed using key-value coding). Because of the added power and flexibility the mapping model provides, the mapping decoder is particularly well-suited to decode XML that wasn't generated by the WOXMLCoder.

Suppose you had the following XML:

```
<command quantity="10">
  <customer><name>Ringle</name></customer>
  <fullMovie name="Alien">
    <date>1979-10-25 00:00:00 -0700</date>
    <cat>Horror</cat>
    <role name="Brett"></role>
    <role name="Kane"></role>
    <role name="Dallas"></role>
    <role name="Parker"></role>
    <role name="Lambert"></role>
    <role name="Ash"></role>
    <role name="Ripley"></role>
  </fullMovie>
</command>
```

Futher suppose that you wanted to decode the above XML into an object of the following class:

```
public class Command extends NSObject {
  public EOGenericRecord movie;
  public NSMutableDictionary customer;
  public int qty;
}
```

The WOXMLDecoder will do the job for you given the following mapping model:

```
<model>
  <entity name="Command" xmlTag="command">
    <property name="qty" xmlTag="quantity" attribute="YES"/>
    <property name="movie" xmlTag="movie"/>
    <property name="customer" xmlTag="customer"/>
  </entity>
```

CLASS WOXMLDecoder

```
<entity name="MyMovie" xmlTag="movie">
  <property name="title" xmlTag="name" attribute="YES"/>
  <property name="dateReleased" xmlTag="date">
  <property name="roles" xmlTag="role">
  <property name="category" xmlTag="cat"/>
</entity>
<entity name="com.apple.yellow.eocontrol.EOGenericRecord" xmlTag="role">
  <property name="roleName" xmlTag="name" attribute="YES"/>
</entity>
</model>
```

The above is a simple example; see the [RelatedLinks](#) example (accessible through the [WebObjects Info Center](#) under [Examples > WebObjects > Java > RelatedLinks](#)) for a more complete example illustrating the use of the WOXMLDecoder with a mapping model.

Interfaces Implemented

NSKeyValueCoding

takeValueForKey

valueForKey

Method Types

Creating a WOXMLDecoder

decoder

decoderWithMapping

Decoding XML

decodeRootObject

CLASS WOXMLDecoder

Reconstructing an object's contents without a mapping model

`decodeBooleanForKey`

`decodeDoubleForKey`

`decodeFloatForKey`

`decodeIntForKey`

`decodeObjectForKey`

Working with the XML parser

`parser`

`parserClassName`

`setParserClassName`

Constructors

WOXMLDecoder

`protected WOXMLDecoder()`

Description forthcoming.

Static Methods

decoder

`public static WOXMLDecoder decoder()`

Creates and returns a new WOXMLDecoder object.

CLASS WOXMLDecoder

decoderWithMapping

```
public static WOXMLDecoder decoderWithMapping(String mappingURL)
```

Creates and returns a new WOXMLDecoder object that decodes XML based upon the mapping model specified by `mappingURL`. For more information, see [“Decoding XML With a Mapping Model”](#) (page 19).

Note: Windows NT uses backslashes where other systems use forward slashes. When prepending the “file:” URL prefix to a path such as is returned by WOREsourceManager’s `pathForResourceNamed` method, on Windows NT the prefix must be “file:\\” while on all other platforms the prefix must be “file://”. See the [RelatedLinks](#) example for one way to select the proper prefix based upon the underlying system.

Instance Methods

addObjectToCache

```
protected void addObjectToCache(_private._DecodingNode childNode, Object anObject)
```

For internal use only.

decodeBooleanForKey

```
public boolean decodeBooleanForKey(String key)
```

Invoke this method from within in your single-argument constructor to set a boolean instance variable to the value of the key element within the XML being decoded. For example, to extract the marital status from the following XML:

```
<element type="Person" objectID="4">
  <Name type="java.lang.String" objectID="5">John Smith</Name>
  <MaritalStatus type="boolean">True</MaritalStatus>
  <NumberOfChildren type="int">2</NumberOfChildren>
</element>
```

You could use something similar to the following:

CLASS WOXMLDecoder

```
married = decoder.decodeBooleanForKey("MaritalStatus");
```

decodeDoubleForKey

```
public double decodeDoubleForKey(String key)
```

Invoke this method from within in your single-argument constructor to set an instance variable of type double to the value of the key element within the XML being decoded.

See Also: decodeIntForKey

decodeFloatForKey

```
public float decodeFloatForKey(String key)
```

Invoke this method from within in your single-argument constructor to set an instance variable of type int to the value of the key element within the XML being decoded.

See Also: decodeIntForKey

decodeIntForKey

```
public int decodeIntForKey(String key)
```

Invoke this method from within in your single-argument constructor to set an instance variable of type int to the value of the key element within the XML being decoded. For example, to extract the number of children from the following XML:

```
<element type="Person" objectID="4">
  <Name type="java.lang.String" objectID="5">John Smith</Name>
  <MaritalStatus type="boolean">True</MaritalStatus>
  <NumberOfChildren type="int">2</NumberOfChildren>
</element>
```

You could use something similar to the following:

```
children = decoder.decodeIntForKey("NumberOfChildren");
```

CLASS WOXMLDecoder

decodeObjectForKey

```
public Object decodeObjectForKey(String key)
```

Invoke this method from within in your single-argument constructor to set an instance variable to a newly constructed object whose class and content depends upon the value of the key element within the XML being decoded. The object being decoded must meet the same criteria outlined in `decodeRootObject`; if not, or if an error arises during the construction of the object, `decodeObjectForKey` throws a `WOXMLException`.

decodeRootObject

```
public synchronized Object decodeRootObject(NSData data)
```

```
public synchronized Object decodeRootObject(  
    org.xml.sax.InputSource inputSource)
```

```
public synchronized Object decodeRootObject(String XMLfileURL)
```

Decodes the indicated XML and constructs a corresponding object. `decodeRootObject` accepts XML either in an `NSData` object, in a file, or through an `InputSource`. If the XML resides within a file, `XMLfileURL` should be a URL that identifies the file containing the XML. If the XML resides within a `String` object, use a `StringReader` object to supply an `InputSource`, like this:

```
stringReader = new StringReader(xmlString);  
is = new InputSource(stringReader);  
// invoke setEncoding (on the input source) if the XML contains multibyte characters  
decodedObject = (NSMutableArray)myDecoder.decodeRootObject(is);
```

In the above example, `xmlString` is a `String` object that contains the XML for an encoded `NSMutableArray` object.

Each object encoded within the XML must have a `type` attribute that indicates the object's class. Each encoded object must be one of the following:

- an instance of `String`
- an instance of `NSArray`
- an instance of `NSDictionary`
- an instance of `NSDate`

CLASS WOXMLDecoder

- an instance of Number (or a subclass, providing that the subclass doesn't add instance variables)
- an instance of NSData
- an object that implements the WOXMLCoding interface
- an instance of EOEnterpriseObject (or a subclass, providing that all instance variables are either attributes or relationships)

Objects that implement the WOXMLCoding interface must also implement a special constructor; see [“Decoding XML Without a Mapping Model”](#) (page 17) for more information and an example.

If the parser is unable to parse the supplied XML, `decodeRootObject` throws a `WOXMLException` that encloses either a `SAXException` or an `IOException`.

getChildNodeType

```
protected String getChildNodeType(_private._DecodingNode childNode)
```

For internal use only.

handler

```
protected _private._DecodingHandler handler()
```

For internal use only.

parser

```
public org.xml.sax.Parser parser()
```

Returns the XML parser (instantiating one based upon the parser class name, if necessary). This method throws a `ClassNotFoundException` if the parser class cannot be located, and an `InstantiationException` or `IllegalAccessException` if the parser cannot be created. The default parser is the SAX parser (`com.ibm.xml.parsers.SAXParser`).

This method is invoked by `decodeRootObject`.

See Also: `parserClassName`, `setParserClassName`

CLASS WOXMLDecoder

parserClassName

```
public String parserClassName()
```

Returns the name of the XML parser's class. By default, this is "com.ibm.xml.parsers.SAXParser".

See Also: parser, setParserClassName

setParserClassName

```
public void setParserClassName(String className)
```

Sets `className` as the name of the class to be instantiated and used as the XML parser. The default parser class name is "com.ibm.xml.parsers.SAXParser". This method must be invoked before the parser is instantiated (by the `parser` method); once the parser has been instantiated, `setParserClassName` has no effect.

See Also: parser, parserClassName

takeValueForKey

```
public void takeValueForKey(  
    Object,  
    String)
```

Description forthcoming.

toString

```
public String toString()
```

Returns a String description of the receiver suitable for debugging purposes.

valueForKey

```
public Object valueForKey(String aKey)
```

Description forthcoming.

WOXMLException

Inherits from: `com.webobjects.foundation.NSForwardException`

Package: `com.webobjects.appserver.xml`

Class Description

This class serves solely to wrap a number of exceptions that can arise during the parsing process, reducing the number of exceptions your code has to catch. In particular, exceptions that are thrown by the SAX parser are encapsulated in WOXMLException objects by WOXMLDecoder and then re-thrown.

The WOXMLException class encapsulates both an exception and an optional text string that can be retrieved with `getMessage` (this message is also prepended to the text that is returned from `toString`).

Constructors

WOXMLException

```
public WOXMLException(String optionalMessage)
```

```
public WOXMLException(Throwable anException)
```

```
public WOXMLException(Throwable anException, String optionalMessage)
```

Creates and returns a new WOXMLException object. If `optionalMessage` is included, the message text can later be retrieved with `getMessage` and is prepended to the string returned from `toString`. If `anException` is supplied, the string returned by `toString` lists (among other things) `anException`'s class.

Instance Methods

getMessage

```
public String getMessage()
```

Returns the optional message supplied when the WOXMLException was created, followed by any optional message from the encapsulated exception.

toString

```
public String toString()
```

Returns a string representation of the WOXMLException object, including the `optionalMessage` (if one was supplied when the WOXMLException was created) and the name of the encapsulated exception.

WOXMLCoding

Implemented by: Custom objects that need to be encoded as XML

Package: com.webobjects.appserver.xml

Interface Description

When operating without a mapping model, the `WOXMLCoder` class is capable of encoding a predefined set of Java classes, any object that is an instance of `EOEnterpriseObject`, and any object that implements the `WOXMLCoding` interface. This interface consists of a single method, `encodeWithWOXMLCoder`, in which you encode your object's instance variables using `WOXMLCoder`'s various `encode...ForKey` methods.

If you'll be reconstituting objects from XML using `WOXMLDecoder`, your classes must have a constructor that takes a `WOXMLDecoder` object as its sole argument. This constructor should consist of a series of `decode...ForKey` method invocations that restore each of your object's instance variables.

The following simple "Person" class implements both the `WOXMLCoding` interface and the single-argument constructor needed to later decode objects of this class.

```
import com.webobjects.appserver.xml.*;
import com.webobjects.foundation.*;
import java.lang.*;
import java.net.*;
import java.math.*;

public class Person extends Object implements WOXMLCoding {
```

INTERFACE WOXMLCoding

```
String name;
boolean married;
int children;

public Person() {
    name = "John Smith";
    married = true;
    children = 2;
}

public void encodeWithWOXMLCoder(WOXMLCoder coder) {
    coder.encodeObjectForKey(name, "Name");
    coder.encodeBooleanForKey(married, "MaritalStatus");
    coder.encodeIntForKey(children, "NumberOfChildren");
}

// constructor required for decoding
public Person(WOXMLDecoder decoder) {
    name = (String)decoder.decodeObjectForKey("Name");
    married = decoder.decodeBooleanForKey("MaritalStatus");
    children = decoder.decodeIntForKey("NumberOfChildren");
}
}
```

See the [XMLArchiving example](#) (accessible through the [WebObjects Info Center](#) under [Examples > WebObjects > Java > XMLArchiving](#)) for a more complete example illustrating the use of the WOXMLCoding interface.

Instance Methods

classForCoder

```
public abstract Class classForCoder()
```

Description forthcoming.

INTERFACE WOXMLCoding

encodeWithWOXMLCoder

```
public abstract void encodeWithWOXMLCoder(WOXMLCoder aCoder)
```

Implement this method using WOXMLCoder's various `encode...ForKey` methods (invoked on `aCoder`) to encode your object's instance variables.

See Also: WOXMLCoder class

INTERFACE WOXMLCoding

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software.

Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITER

Greg Wilson

PRODUCTION EDITOR

Lorraine Findlay

COLOPHON