Java™API for XML Processing Maintenance Release 1.6

Description:
Maintenance review of the JAXP 1.5 Specification

Maintenance Lead:
Joe Wang, Oracle Corporation

Feedback:
Please send comments to eg@jaxp.java.net

Rationale for Changes:

1. Use java.util.ServiceLoader

   JAXP defines a number of service provider interfaces to allow deployment with alternative parser implementations (service providers). Service providers are located by means of:

   1) Use a system property named after the corresponding factory name;

   2) Use the properties file "lib/jaxp.properties" in the JRE directory;

   3) Read JAR service file, for example, `META-INF/services/java.xml.datatype.DatatypeFactory`;

   4) Fall back to the system default implementation.

   This update to the JAXP specification updates the 3rd step so that it requires the use of the service provider loader facility defined by java.util.ServiceLoader. The rational for this is to allow for future modularization of the Java SE platform where service providers may be deployed by means other than JAR files and perhaps without the service configuration files. Note that the JAXP has always specified the use of the 'Services API' without reference to a specific API or service provider loading facility.

2. StAX 1.2, JSR 173 Stream API for XML MR3

   The javax.xml.stream APIs are updated to StAX 1.2, JSR 173 Streaming API for XML Maintenance Review 3.

3. API package org.w3c.dom.views

   API package org.w3c.dom.events was included by reference in JAXP 1.3. However, its dependent package org.w3c.dom.views was missing and should be added.

Announcement

Since JAXP version 1.1, JSR 206 has been distributed as a standalone technology and part of the Java SE at the same time. The JAXP API has been stable (no significant changes) for a long time and the need to use newer releases of the JAXP API with shipping releases of Java SE has mostly disappeared in recent years.

In accordance with JCP 2.9 Process Document, 2.1.4 Platform Inclusion , we are announcing the end of JAXP Standalone distribution. After MR3, JAXP 1.6, the technology that JSR 206 defines will be delivered as a part of the Java SE solely. Future changes in the JAXP API will be defined through the Platform JSR.

The subsumption of the JAXP API into the Platform JSR does not change any mechanisms defined in JAXP. The service provider interfaces are the same except that they will then be directly specified in the Platform JSR. Deployment of alternative implementations of the JAXP APIs will continue to be supported.

Proposed changes

1. Use java.util.ServiceLoader

Use java.util.ServiceLoader to replace the 3rd step. The followings are changes to Chapter 4. Pluggability Layer in JAXP Specification, Version 1.4.

1.1 SAX Plugability

Defined in the description of the following class and method:
public abstract class SAXParserFactory
public static SAXParserFactory newInstance()

1) The 3rd step:

Original Statement	New Statement
Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for the classname in the file META-INF/services/javax.xml.parsers.SAXParserFactory in jars available to the runtime.	Use the service-provider loading facilities, defined by the java.util.ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.

2) The 4th step:

Original Statement	New Statement
Platform default SAXParserFactory instance.	`Otherwise the system-default implementation is returned.`

1.2 DOM Plugability

Defined in the description of the following class and method:
public abstract class DocumentBuilderFactory
public static DocumentBuilderFactory newInstance()

1) The 3rd step:

Original Statement	New Statement
Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file META-INF/services/javax.xml.parsers.DocumentBuilderFactory in jars available to the runtime.	Uses the service-provider loading facilities, defined by the java.util.ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.

2) The 4th step:

Original Statement	New Statement
Platform default <code>DocumentBuilderFactory</code> instance.	Otherwise, the system-default implementation is returned.

1.3 XSLT Plugability

Defined in the description of the following class and method:
public abstract class TransformerFactory
public static TransformerFactory newInstance() throws TransformerFactoryConfigurationError

1) The 3rd step:

Original Statement	New Statement
Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file META-INF/services/javax.xml.transform.TransformerFactory in jars available to the runtime.	Use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.

2) The 4th step:

Original Statement	New Statement
Platform default TransformerFactory instance.	Otherwise, the system-default implementation is returned.

1.4 XPath Plugability

Defined in the description of the following class and method:
public abstract class XPathFactory
public static final static XPathFactory newInstance()

1) The 3rd step:

Original Statement	New Statement
The class loader is asked for service provider provider-configuration files matching javax.xml.xpath.XPathFactory in the resource directory META-INF/services. See the JAR File Specification for file format and parsing rules. Each potential service provider is required to implement the method: isObjectModelSupported(String objectModel) The first service provider found in class loader order that supports the specified object model is returned.	Use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used. Each potential service provider is required to implement the method isObjectModelSupported(String objectModel) . The first service provider found that supports the specified object model is returned.

	In case of ServiceConfigurationError an XPathFactoryConfigurationException will be thrown.

2) Other minor change:

Original Statement	New Statement
public static final XPathFactory newInstance()	public static XPathFactory newInstance()

1.5 Validation Plugability

Defined in the description of the following class and method:
public abstract class SchemaFactory
public static final static SchemaFactory newInstance(java.lang.String schemaLanguage)

1) The 3rd step:

Original Statement	New Statement
The class loader is asked for service provider provider-configuration files matching javax.xml.validation.SchemaFactory in the resource directory META-INF/services. See the JAR File Specification for file format and parsing rules. Each potential service provider is required to implement the method: isSchemaLanguageSupported(String schemaLanguage) The first service provider found in class loader order that supports the specified schema language is returned.	Use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used. Each potential service provider is required to implement the method isSchemaLanguageSupported(String schemaLanguage) . The first service provider found that supports the specified schema language is returned. In case of ServiceConfigurationError a SchemaFactoryConfigurationError will be thrown.

2) Other change:
Fix a typo in the description of class SchemaFactory, Schema Language section

Original Statement	New Statement
implentors	implementors

3) New class

public class SchemaFactoryConfigurationError
Thrown when a problem with configuration with the Schema Factories exists. This error will typically be thrown when the class of a schema factory specified in the system properties cannot be found or instantiated.

Synopsis:
public SchemaFactoryConfigurationError extends Error {
    public SchemaFactoryConfigurationError();
    public SchemaFactoryConfigurationError(java.lang.String message);
    public SchemaFactoryConfigurationError(java.lang.String message, java.lang.Throwable cause);
    public SchemaFactoryConfigurationError(java.lang.Throwable cause);
}

Inheritance Path:
java.lang.Object

java.lang.Throwable
java.lang.Error
javax.xml.validation.SchemaFactoryConfigurationError

Constructor Summary

Constructor and Description

SchemaFactoryConfigurationError`()`
Create a new `SchemaFactoryConfigurationError` with no detail message.

SchemaFactoryConfigurationError`(java.lang.String message)`
Create a new `SchemaFactoryConfigurationError` with the `String` specified as an error message.

SchemaFactoryConfigurationError`(java.lang.String message, java.lang.Throwable cause)`
Create a new `SchemaFactoryConfigurationError` with the given `Throwable` base cause and detail message.

SchemaFactoryConfigurationError`(java.lang.Throwable cause)`
Create a new `SchemaFactoryConfigurationError` with the given `Throwable` base cause.

1.6 Streaming API for XML Plugability

Defined in the description of the following classes and methods:
public abstract class XMLEventFactory
public static XMLEventFactory newInstance() throws FactoryConfigurationError

public abstract class XMLInputFactory
public static XMLInputFactory newInstance() throws FactoryConfigurationError

public abstract class XMLOutputFactory
public static XMLOutputFactory newInstance() throws FactoryConfigurationError

1) The 3rd step:

Original Statement	New Statement
Use the Services API (as detailed in the JAR specification), if available, to determine the classname. The Services API will look for a classname in the file META-INF/services/javax.xml.stream.XMLEventFactory in jars available to the runtime.	Use the service-provider loading facilities, defined by the ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.

2) The 4th step:

Original Statement	New Statement
Platform default XMLEventFactory instance.	Otherwise, the system-default implementation is returned.

3) Error:

Original Statement	New Statement
Throws: FactoryConfigurationError - if an instance of this factory cannot be loaded	Throws: FactoryConfigurationError - in case of service configuration error or if the implementation is not available or cannot be instantiated.

1.7 Datatype Plugability

Defined in the description of the following class:
public abstract class DatatypeFactory

1) The general statement:

Original Statement	New Statement
#newInstance() is used to create a new DatatypeFactory.The following implementation resolution mechanisms are used in the following order:	A new instance of the DatatypeFactory is created through the #newInstance() method that uses the following implementation resolution mechanisms to determine an implementation:

2) The 3rd step:

Original Statement	New Statement
The services resolution mechanism is used, e.g. META-INF/services/java.xml.datatype.DatatypeFactory. Any Exception thrown during the instantiation process is wrapped as a DatatypeConfigurationException.	Uses the service-provider loading facilities, defined by the java.util.ServiceLoader class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used. In case of service configuration error a javax.xml.datatype.DatatypeConfigurationException will be thrown.

2 StAX 1.2, JSR 173 Stream API for XML MR3

The revision 1.2 of Stream API for XML Processing for the Java Platform, that is, JSR 173 Maintenance Review 3, deprecated newInstance methods in the StAX factories and added newFactory methods. The followings are API changes of StAX 1.2 with the ServiceLoader changes above incorporated in the description of the newFactory methods. Refer to Change Log for JSR-000173 Streaming API for XML, Maintenance Review 3.

2.1 Deprecations

Class:
javax.xml.stream.XMLEventFactory

Method:
public static XMLEventFactory newInstance(java.lang.String factoryId,
java.lang.ClassLoader classLoader throws FactoryConfigurationError

Add the following deprecation notice:
@deprecated to maintain API consistency. All newInstance methods are replaced with corresponding newFactory methods. The replacement newFactory(String factoryId, ClassLoader classLoader) method defines no changes in behavior from this method.

Class:
javax.xml.stream.XMLInputFactory

Method:
public static XMLInputFactory newInstance(java.lang.String factoryId,
java.lang.ClassLoader classLoader throws FactoryConfigurationError

Add the following deprecation notice:
@deprecated to maintain API consistency. All newInstance methods are replaced with corresponding newFactory methods. The replacement newFactory(String factoryId, ClassLoader classLoader) method defines no changes in behavior from this method.

Class:
javax.xml.stream.XMLOutputFactory

Method:
public static XMLInputFactory newInstance(java.lang.String factoryId,
java.lang.ClassLoader classLoader throws FactoryConfigurationError

Add the following deprecation notice:
@deprecated This method has been deprecated because it returns an instance of XMLInputFactory, which is of the wrong class. Use the new method newFactory(java.lang.String factoryId, java.lang.ClassLoader classLoader) instead.

2.2 New factory methods

2.2.1 javax.xml.stream.XMLEventFactory

Method:
public static XMLEventFactory newFactory() throws FactoryConfigurationError

Create a new instance of the factory.

This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the XMLEventFactory implementation class to load:

- Use the javax.xml.stream.XMLEventFactory system property.
- Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the system property defined above.
- Use the service-provider loading facilities, defined by the `ServiceLoader` class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.
- Otherwise, the system-default implementation is returned.

Once an application has obtained a reference to a XMLEventFactory it can use the factory to configure and obtain stream instances.

Note that this is a new method that replaces the deprecated newInstance() method. No changes in behavior are defined by this replacement method relative to the deprecated method.

Throws:

`FactoryConfigurationError` - in case of service configuration error or if the implementation is not available or cannot be instantiated.

public static XMLEventFactory newFactory(java.lang.String factoryId, java.lang.ClassLoader classLoader) throws FactoryConfigurationError

Create a new instance of the factory. If the classLoader argument is null, then the ContextClassLoader is used.

This method uses the following ordered lookup procedure to determine the XMLEventFactory implementation class to load:

- Use the value of the system property identified by `factoryId`.
- Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the given `factoryId`.
- If `factoryId` is "javax.xml.stream.XMLEventFactory", use the service-provider loading facilities, defined by the `ServiceLoader` class, to attempt to locate and load an implementation of the service using the specified ClassLoader. If the classLoader is null, the default loading mechanism will apply: That is, the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.
- Otherwise, throws a `FactoryConfigurationError`.

Note that this is a new method that replaces the deprecated `newInstance(String factoryId, ClassLoader classLoader)` method. No changes in behavior are defined by this replacement method relative to the deprecated method.

@apiNote The parameter factoryId defined here is inconsistent with that of other JAXP factories where the first parameter is fully qualified factory class name that provides implementation of the factory.

Parameters:

`factoryId` - Name of the factory to find, same as a property name

`classLoader` - classLoader to use

Returns:

the factory implementation

Throws:

`FactoryConfigurationError` - in case of service configuration error or if the implementation is not available or cannot be instantiated.

`FactoryConfigurationError` - if an instance of this factory cannot be loaded

2.2.2 javax.xml.stream.XMLInputFactory

Method:
public static XMLInputFactory newFactory() throws FactoryConfigurationError

Create a new instance of the factory.

This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the XMLInputFactory implementation class to load:

- Use the javax.xml.stream.XMLInputFactory system property.
- Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the system property defined above.
- If `factoryId` is "javax.xml.stream.XMLInputFactory", use the service-provider loading facilities, defined by the `ServiceLoader` class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.
- Otherwise, the system-default implementation is returned.

Once an application has obtained a reference to a XMLInputFactory it can use the factory to configure and obtain stream instances.

Note that this is a new method that replaces the deprecated newInstance() method. No changes in behavior are defined by this replacement method relative to the deprecated method.

Throws:

`FactoryConfigurationError` - in case of service configuration error or if the implementation is not available or cannot be instantiated.

public static XMLInputFactory newFactory(java.lang.String factoryId, java.lang.ClassLoader classLoader) throws FactoryConfigurationError

Create a new instance of the factory. If the classLoader argument is null, then the ContextClassLoader is used.

This method uses the following ordered lookup procedure to determine the XMLInputFactory implementation class to load:

- Use the value of the system property identified by `factoryId`.
- Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the given `factoryId`.
- If `factoryId` is "javax.xml.stream.XMLInputFactory", use the service-provider loading facilities, defined by the `ServiceLoader` class, to attempt to locate and load an implementation of the service using the specified ClassLoader. If the classLoader is null, the default loading mechanism will apply: That is, the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.
- Otherwise, throws a `FactoryConfigurationError`.

Note that this is a new method that replaces the deprecated `newInstance(String factoryId, ClassLoader classLoader)` method. No changes in behavior are defined by this replacement method relative to the deprecated method.

@apiNote The parameter factoryId defined here is inconsistent with that of other JAXP factories where the first parameter is fully qualified factory class name that provides implementation of the factory.

Parameters:

`factoryId` - Name of the factory to find, same as a property name

`classLoader` - classLoader to use

Returns:

the factory implementation

Throws:

`FactoryConfigurationError` - in case of service configuration error or if the implementation is not available or cannot be instantiated.

`FactoryConfigurationError` - if an instance of this factory cannot be loaded

2.2.3 javax.xml.stream.XMLOutputFactory

Method:
public static XMLOutputFactory newFactory() throws FactoryConfigurationError

Create a new instance of the factory.

This static method creates a new factory instance. This method uses the following ordered lookup procedure to determine the XMLOutputFactory implementation class to load:

- Use the javax.xml.stream.XMLOuputFactory system property.
- Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the implementation class with the key being the system property defined above.
- Use the service-provider loading facilities, defined by the `ServiceLoader` class, to attempt to locate and load an implementation of the service using the default loading mechanism: the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.
- Otherwise, the system-default implementation is returned.

Once an application has obtained a reference to a XMLOutputFactory it can use the factory to configure and obtain stream instances.

Note that this is a new method that replaces the deprecated newInstance() method. No changes in behavior are defined by this replacement method relative to the deprecated method.

Throws:

`FactoryConfigurationError` - in case of service configuration error or if the implementation is not available or cannot be instantiated.

public static XMLOutputFactory newFactory(java.lang.String factoryId,
java.lang.ClassLoader classLoader) throws FactoryConfigurationError

Create a new instance of the factory. If the classLoader argument is null, then the ContextClassLoader is used.

This method uses the following ordered lookup procedure to determine the XMLOutputFactory implementation class to load:

- Use the value of the system property identified by `factoryId`.
- Use the properties file "lib/stax.properties" in the JRE directory. This configuration file is in standard java.util.Properties format and contains the fully qualified name of the

implementation class with the key being the given `factoryId`.

- If `factoryId` is "javax.xml.stream.XMLOutputFactory", use the service-provider loading facilities, defined by the `ServiceLoader` class, to attempt to locate and load an implementation of the service using the specified ClassLoader. If the classLoader is null, the default loading mechanism will apply: That is, the service-provider loading facility will use the current thread's context class loader to attempt to load the service. If the context class loader is null, the system class loader will be used.
- Otherwise, throws a `FactoryConfigurationError`.

Note that this is a new method that replaces the deprecated `newInstance(String factoryId, ClassLoader classLoader)` method. No changes in behavior are defined by this replacement method relative to the deprecated method.

@apiNote The parameter factoryId defined here is inconsistent with that of other JAXP factories where the first parameter is fully qualified factory class name that provides implementation of the factory.

Parameters:

`factoryId` - Name of the factory to find, same as a property name

`classLoader` - classLoader to use

Returns:

the factory implementation

Throws:

`FactoryConfigurationError` - in case of service configuration error or if the implementation is not available or cannot be instantiated.

`FactoryConfigurationError` - if an instance of this factory cannot be loaded

3. API package org.w3c.dom.views

This specification includes the following API package by reference:

`org.w3c.dom.views`

The org.w3c.dom.views package includes the following interfaces:

```
public interface AbstractView
      public DocumentView getDocument();

public interface DocumentView
      public AbstractView getDefaultView();
```

4. Compatibility for the ServiceLoader change

This specification mandates the use of java.util.ServiceLoader for finding service providers. Service providers across JAXP will now be located consistently following the process as defined in ServiceLoader. This change may represent some subtle difference from implementations of previous

versions of the specification where the provider-configuration file may have been located differently, for example, by using a different getXXX method of the ClassLoader than ServiceLoader. Applications that implement their own Classloaders shall therefore make sure that the ClassLoaders' getXXX methods are implemented consistently so as to maintain compatibility.

The StAX API, JSR 173, defined newInstance/newFactory method with a factoryId as a parameter. Since there was no constraint on what the value could be in the StAX specification, it implied it could be any arbitrary string. With this specification change, in the context of JAXP, the value of factoryId must be the name of the base service class if it is intended to represent the name of the service configuration file, that is, if it is not the name of a System Property.

5. End of JSR 206 Java™API for XML Processing

This MR, JAXP 1.6, will be the last of JSR 206 JAXP specification as a standalone. After this MR, JSR 206 will be subsumed into the Java Platform JSR, and JAXP will no longer exist as a standalone library.