TIBCO ActiveMatrix BusinessWorks™ ActiveAspects Plug-in

User’s Guide

Software Release 1.1.0
May 2011

TIBCO®
The Power of Now®

Important Information

Contents

Preface

TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in (BWAA) extends the capabilities of TIBCO ActiveMatrix BusinessWorks by adding an Aspect Oriented Programming capability. This allows you to modularize and inject crosscutting concerns to enhance your BW processes at deploy time while keeping the original BW process intact. It exposes a JAVA API to build jar files that can alter the execution of a BusinessWorks Application.

Topics

Changes from the Previous Release of this Guide

This section itemizes the major changes from the previous release of this guide.

New Chapters Added

- Chapter 5, Object Sharing Between Java Activities and Advice Implementation provides information about the new API's which will be exposed to enable the user to use Object Sharing feature.
- Chapter 6, BWAA Palette contains details about the activity to resume a previously hibernated job.
- Chapter 7, Monitoring and Management describes the TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in monitoring and management features.

Related Documentation

This section lists documentation resources you may find useful. The documentation road map shows the relationships between the the books and online references in this product's documentation set.

TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in Documentation

The following documents form the TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in documentation set:

- *TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in Installation and Configuration* Read this manual for information on product installation.
- *TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in User's Guide* Read this manual to learn how to develop, build, and deploy aspects in ActiveMatrix BusinessWorks.
- *TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in API Reference* This manual gives information about the JAVA API for creating Advice Implementations in TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in.
- *TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in Release Notes* Read the release notes for the known issues.

Other TIBCO Product Documentation

TIBCO ActiveMatrix BusinessWorks is a pre-requisite for TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in and is used with other products. You may find it useful to read the documentation for the following TIBCO products:

TIBCO ActiveMatrix BusinessWorks

- *TIBCO ActiveMatrix BusinessWorks Concepts* Read this manual before reading any other manual in the documentation set. This manual describes terminology and concepts of ActiveMatrix BusinessWorks, and the other manuals in the documentation set assume you are familiar with the information in this manual.
- *TIBCO ActiveMatrix BusinessWorks Getting Started* This manual steps you through a very simple example of designing, deploying, and monitoring a ActiveMatrix BusinessWorks process.

- *ActiveMatrix BusinessWorks Process Design Guide* This manual describes how to create, edit, and test business processes using ActiveMatrix BusinessWorks.
- *ActiveMatrix BusinessWorks Palette Reference* This manual describes each of the palettes available in ActiveMatrix BusinessWorks.
- *TIBCO ActiveMatrix BusinessWorks Administration* This manual describes how to use TIBCO Administrator to deploy, manage, and monitor ActiveMatrix BusinessWorks processes.
- *TIBCO ActiveMatrix BusinessWorks Installation* Read this manual for information on installing one or more components of ActiveMatrix BusinessWorks and setting up a ActiveMatrix BusinessWorks domain.
- *TIBCO ActiveMatrix BusinessWorks Error Codes* This manual describes errors returned by ActiveMatrix BusinessWorks.
- *TIBCO ActiveMatrix BusinessWorks Release Notes* Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

Other TIBCO Products

- TIBCO Designer™ software: TIBCO Designer is an easy to use graphical user interface for design-time configuration of TIBCO applications.
- TIBCO Administrator™ software: TIBCO Administrator is the monitoring and managing interface for new-generation TIBCO products such as TIBCO ActiveMatrix BusinessWorks.
- TIBCO Adapter software
- Third-Party Documentation

Typographical Conventions

The following typographical conventions are used in this manual.

Table 1 General Typographical Conventions

Convention	Use
TIBCO_HOME	Many TIBCO products must be installed within the same home directory. This directory is referenced in documentation as *TIBCO_HOME*. The value of *TIBCO_HOME* depends on the operating system. For example, on Windows systems, the default value is `C:\tibco`. Incompatible products and multiple instances of the same product can be installed into different installation environments.
BW_*HOME*	TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in installs into the ActiveMatrix BusinessWorks directory within *TIBCO_HOME*. This directory is referenced in documentation as *BW_HOME*. The value of *BW_HOME* depends on the operating system. For example on Windows systems, the default value is `C:\tibco\bw\5.9`.
`code font`	Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example: Use `MyCommand` to start the foo process.
`bold code font`	Bold code font is used in the following ways: • In procedures, to indicate what a user types. For example: Type **`admin`**. • In large code samples, to indicate the parts of the sample that are of particular interest. • In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, `MyCommand` is enabled: `MyCommand [`**`enable`**` | disable]`
italic font	Italic font is used in the following ways: • To indicate a document title. For example: See *TIBCO ActiveMatrix BusinessWorks Concepts*. • To introduce new terms For example: A portal page may contain several portlets. *Portlets* are mini-applications that run in a portal. • To indicate a variable in a command or code syntax that you must replace. For example: `MyCommand` *PathName*

Table 1 General Typographical Conventions (Cont'd)

Convention	Use
Key combinations	Key name separated by a plus sign indicate keys pressed simultaneously. For example: Ctrl+C. Key names separated by a comma and space indicate keys pressed one after the other. For example: Esc, Ctrl+Q.
📝	The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.
💡	The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result.
⚠	The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken.

Connecting with TIBCO Resources

How to Join TIBCOmmunity

TIBCOmmunity is an online destinaton for TIBCO customers, partners, and resident experts—a place to share and access the collective experience of the TIBCO community. TIBCOmmunity offers forums, blogs, and access to a variety of resources. To register, go to http://www.tibcommunity.com.

How to Access All TIBCO Documentation

After you join TIBCOmmunity, you can access the documentation for all supported product versions here:

http://docs.tibco.com/TibcoDoc

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, please contact TIBCO Support as follows.

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:

  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:

  https://support.tibco.com

  Entry to this site requires a user name and password. If you do not have a user name, you can request one.

Chapter 1 Overview

This chapter introduces TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in.

Topics

Introduction

TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in (BWAA) extends the capabilities of TIBCO ActiveMatrix BusinessWorks by adding an Aspect Oriented Programming capability. This allows you to modularize and inject crosscutting concerns to enhance your ActiveMatrix BusinessWorks processes at deploy time while keeping the original ActiveMatrix BusinessWorks process intact. It exposes a JAVA API to build applications, which are called advice implementations that can alter the execution of a ActiveMatrix BusinessWorks Application.

This chapter provides an overview of features of the product, BusinessWorks ActiveAspects Plug-in Concepts, BusinessWorks ActiveAspects Plug-in Resources, and AOP terminology.

Chapter 2 describes the Point Cut Query Language used for writing point cut expressions.

Chapter 3 describes the Asynchronous Advice Implementations.

Chapter 4 describes the Hibernate/Resume feature of TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in.

Chapter 5 describes the JAVA Object sharing between BW Java activities and Advice implementations.

Chapter 6 provides information about the BWAA palette.

Chapter 7 describes the Monitoring and Management feature.

Appendix A, ActivityTypes provides information about the activity types of all TIBCO ActiveMatrix BusinessWorks activities that are useful for the user to generate expressions in Point cut query language.

Appendix B, Developing gXML Applications provides information for developing gXML applications.

Aspect Oriented Programming (AOP) Terminology

It is important for the user to understand the terms used in the document about the product. Table 2 describes the AOP terminology used throughout this document.

Table 2 AOP Terminology

Name	Description
Process Aspect	An XML resource that can alter the behavior of a BW application by injecting user defined code at specific points within a process. It contains the declaration of most of the concepts in this list.
Advice Implementation	User developed Java class that executes one or more cross-cutting concerns around an activity.
Advice Implementation Instance	A java object instance of an advice implementation.
Advice	A configured Advice Implementation that is placed around a given activity. Advices are defined in aspect files. Advice is a design-time concept.
Advice Instance	A particular instance of an advice. Advice Instance is a run-time concept.
Process Join Point	A specific point in a process that supports injection of user defined advices.
Point Cut	An expression or query that selects join points based on certain conditions. Point cuts are defined in aspect files.
Target Activity	An activity whose input, output or exception messages are intercepted by an advice.
Aspect Library	A JAR file that contains one or more aspects.
Aspect Implementation Library	A JAR file that contains one or more aspect implementation files.

Table 2 AOP Terminology

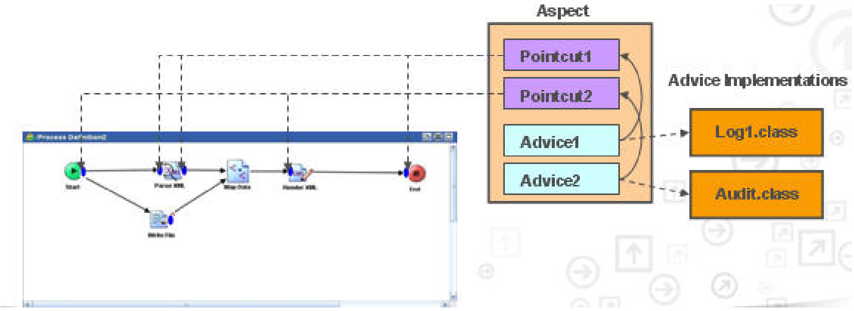
Name	Description
aspectPath	Path where aspect libraries are located.

Overview

A Process-Oriented Aspect (POA) alters the execution of a process by injecting **Advices**, which are user defined code, in specific points of the process called **Join Points**. The selection of the Join Points is made based on expressions called **Point Cuts**.

An Aspect is the collection of Point Cuts and Advices. Aspects implement features that cut across different layers of a BW application (i.e. across different BW processes). One of the key characteristics of the POA style programming is that these features can be developed, packaged and deployed independent of BW Applications. This provides a very flexible model and allows a different user, the Aspect Developer to develop business logic that can alter post design-time, the behavior of a BW application.

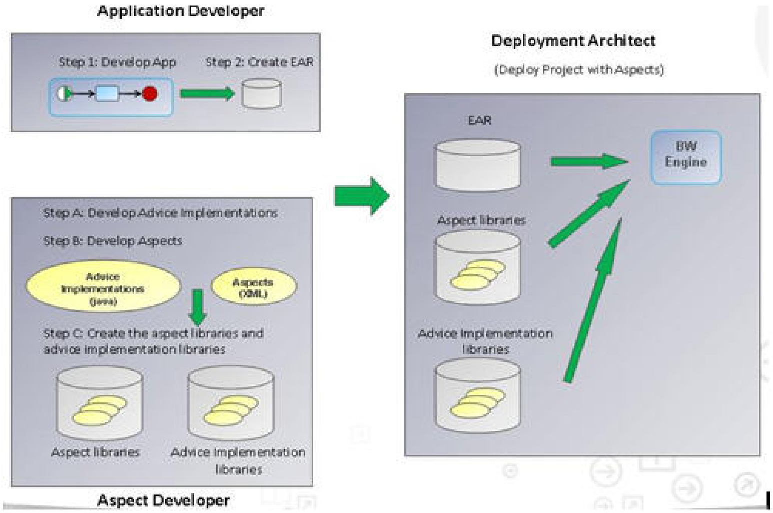
Figure 1 ActiveMatrix BusinessWorks ActiveAspects Behavior



Aspects are packaged in Aspect Libraries, which are JAR files. These are different from the JAR files that contain Advice Implementations. These JAR files *must* be available in the aspect path in order to allow the aspects to be loaded by the BW engine. The aspect injection process happens at run-time through a process called **In-Memory XML Weaving**.

If the Aspect Libraries are not available in the aspectPath, the BW engine executes the BW application as is (i.e. without altering the behavior defined by the ActiveMatrix BusinessWorks developer). ActiveMatrix BusinessWorks provides a platform for developing both ActiveMatrix BusinessWorks applications as well as process-oriented aspects that can be injected in these applications.

Figure 2 ActiveMatrix BusinessWorks ActiveAspects Plug-in Process


Application Developer
Step 1: Develop App
Step 2: Create EAR
Deployment Architect
(Deploy Project with Aspects)
EAR
BW Engine
Aspect libraries
Advice implementation libraries
Step A: Develop Advice Implementations
Step B: Develop Aspects
Advice Implementations (Java)
Aspects (XML)
Step C: Create the aspect libraries and advice implementation libraries
Aspect libraries
Advice implementation libraries
Aspect Developer

Roles and Responsibilities

Application Developer	This developer of an TIBCO ActiveMatrix BusinessWorks application creates processes and generated an EAR file for deployment. The application developer is aware of the potential injection of POAs but is not responsible for developing them.
Advice Implementation Developer	This developer develops the Java code that gets injected into the TIBCO ActiveMatrix BusinessWorks processes via the POAs. In general, this developer may have very little information about the ActiveMatrix BusinessWorks application where the aspects are injected. This user's responsibility is to create a robust piece of code that addresses a crosscutting concern for the TIBCO ActiveMatrix BusinessWorks application.
Aspect Developer	This developer creates the aspects that are applied to one or more TIBCO ActiveMatrix BusinessWorks applications. This user must have a deep knowledge about the TIBCO ActiveMatrix BusinessWorks application as well as the advices that are available, that can be injected in the TIBCO ActiveMatrix BusinessWorks application. In some organizations the Aspect Developer may also play the role of the Advice Implementation Developer. For simplicity, this document assumes that there is only one user, the Aspect Developer, who plays both roles.
Deployment Architect	Deploys both a TIBCO ActiveMatrix BusinessWorks application and its associated aspects.

Process Join Point

A Process Join Point is a well defined point in a process flow where a special event occurs. A special event could be the beginning of the execution of an activity or the end of the execution of an activity. By inserting advices in join points, a TIBCO ActiveMatrix BusinessWorks user can alter the execution of the process. Table 3 describes the join points supported by TIBCO BusinessWorks.

Table 3 TIBCO ActiveMatrix BusinessWorks Supported Join Points

Activity	Functionality
Before Activity	The advice runs before an activity is executed.
After Returning Activity	The advice runs after an activity is successfully completed.
After Throwing Activity	The advice runs after an activity throws an exception.
After Activity	The advice executes after an activity either completes successfully or it throws an exception. This is equivalent to JAVA's `finally` construct.

TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in

TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in consists of resources.

Process Aspect

Process Aspect is a new concept in TIBCO ActiveMatrix BusinessWorks that provides a way to alter the execution of a process by injecting user defined code in specific points of a process. In general, Aspects are defined, packaged and deployed completely independent of TIBCO ActiveMatrix BusinessWorks applications (like EAR files).

Two main parts of an Aspect are:

- **One or more Point Cuts** - Provide the selection or the query logic for selecting the Join Points where Advices are inserted.
- **One or more Advices** - Provide the user a defined code that will be executed at a specified Point cut.

In TIBCO ActiveMatrix BusinessWorks, an Aspect is implemented with an XML file that has a "`bwaspect`" extension.

The following snippet shows the pseudo-schema of the TIBCO ActiveMatrix BusinessWorks aspect:

```
<aspect xmlns="http://schemas.tibco.com/bw/poa"
targetNamespace="xsd:anyURI"
xmlns:xsi="xsd:anyURI" xsi:schemaLocation="xsd:anyURI"
order=" xs:unsignedByte"?>
<documentation>…</documentation>?
<pointcut name="NCName"> +
<documentation>...</documentation>?
 <query queryLanguage="xsd:anyURI"?>...</query>
</pointcut>
<advice name="NCName" pointcut="NCName"> +
<documentation>...</documentation>?
<activity where="Before | AfterReturning | AfterThrowing | After"
exceptionType="xsd:string">
<implementation.java className="xsd:String">
</activity>
<properties>
<property name = "NCName">…</property>
```

```
</properties>
</advice>
</aspect>
```

An aspect has a required `@targetNamespace` attribute, whose value *must* be unique in the context of a BW engine. At the BW engine initialization time, if two or more aspects are found with the same `targetNamespace`, it throws a `BWAspectConfigurationException` and exits. An aspect also has an optional `@order` attribute, which is used to establish an execution order for advices that are inserted in the same join point.

A point cut is used to select all the join points where advices are injected. The selection is implemented by performing a query on the BW Project infoset. For details refer to Chapter 2, Point Cut Query Language.

A point cut has a required `@name` attribute, whose value must be unique in the context of an aspect. At the BW engine initialization time, if more than one point cuts are found with the same name, the BW Engine throws a `BWAspectConfigurationException`.

A common usage of an aspect is to contain more than one point cut. This is useful especially when the aspect contains multiple advice definitions. At run-time, when advices are injected in the business processes, only the point cuts that are actually referenced by advices are used. Hence, all the point cuts not referenced by advices are discarded by the engine at run-time.

An advice is a configured user defined code that gets injected in a process at run-time, thereby changing its behavior. The location of where the advice gets injected is specified by referencing a point cut, via the required `@pointcut` attribute.

An advice cannot be inserted anywhere in a process other than the valid point cuts.

Currently supported join points can only alter the activity behaviors. There are four flavors of join points. They are,

- `Before`
- `AfterRunning`
- `AfterThrowing`
- `After`

An `AfterThrowing` advice can also specify an exception type `QName` via the `@exceptionType` attribute. Use this attribute to select a particular type of exception. **If this attribute is not set, then any exception thrown by the activity will trigger the execution of the advice.**

An advice has a required `@name` attribute, whose value must be unique in the context of an aspect. At run-time, the BW engine throws a `BWAspectConfigurationException` if it finds two or more advices with the same name. An advice also has an implementation, which is the actual code that gets executed at run-time.

Currently TIBCO ActiveMatrix BusinessWorks supports only JAVA for defining advice implementations.

An advice implementation Java class, which is specified in the `@className` attribute of the implementation element, must implement a specific contract. For more information about this contract and the API that is available to advice implementations, refer Advice Implementations.

The BW engine does not halt its execution if an exception such as a `BWAspectConfigurationException`, is thrown during its initialization process.

Aspects, point cuts, and advices can have an optional documentation element that can be used to store comments associated with these entities.

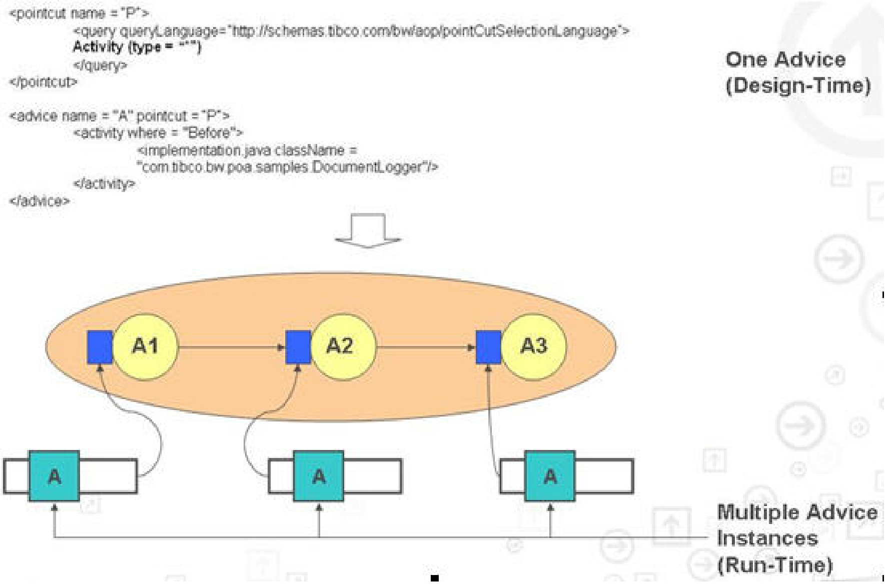
Advices, Advice Instances and Advice Implementation Instances

This section describes the difference between an advice and an advice instance. **Advice is a *design-time* concept whereas Advice Instance is a *run-time* concept.** An advice defined at design-time in an aspect XML file, has a configuration that includes a reference to a point cut. At run-time, one advice instance is created for every join point that is selected by the point cut. Unless Scoping is specified this is the default behavior. For details refer to, Scopes.

The advice instance is actually the entity that is executed at run-time, not the advice.

Figure 3 shows an advice that gets instantiated and injected before all activities. Since the process has three activities, three advice instances are actually created at run-time.

Figure 3   Advice Instances Created at Run-Time

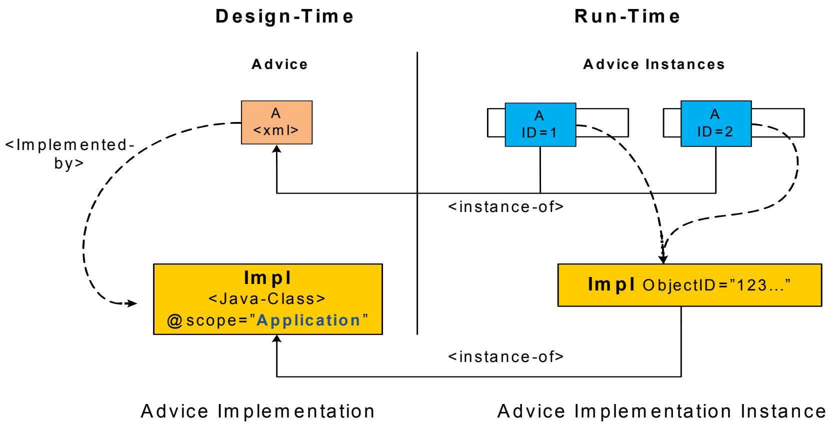


```
<pointcut name = "P">
        <query queryLanguage="http://schemas.tibco.com/bw/aop/pointCutSelectionLanguage">
        Activity (type = "*")
        </query>
</pointcut>

<advice name = "A" pointcut = "P">
        <activity where = "Before">
                <implementation.java className =
                "com.tibco.bw.poa.samples.DocumentLogger"/>
        </activity>
</advice>
```

At run-time, the TIBCO ActiveMatrix BusinessWorks engine may not always create three `DocumentLogger` instances.. This mainly depends on the scope attribute of the `DocumentLogger` class. **If the scope is "Application", only one java object instance of `DocumentLogger` is actually created at run-time per TIBCO ActiveMatrix BusinessWorks application.** This is very important to understand since it represents the key difference between an advice instance and an advice implementation instance. **The instance of the advice implementation (such as the object of the java class that provides the implementation of the advice) is not the same as the advice instance.** Three advice instances may in fact share the same advice implementation java object instance.

Figure 4 shows two advice instances sharing the same advice implementation instance.

Figure 4 Two Advices Sharing Same Advice Implementation Instance



Advice Configuration Properties

Advice implementations (i.e. Java classes) can define configuration properties. These properties provide a way for advices to configure the execution of their associated Java class. This is very useful especially when different advices that share the same implementation, want to execute it using different parameter values. For details about how advice implementations can define configuration properties, refer Advice Implementation Properties.

For each configuration property defined in the advice implementation there can be a value set in the advice configuration. If an advice implementation property is not set in an advice, the default value that is specified in the java class, if exists, is used.

Each advice has a `<properties>` element that contains these property values.

Advice implementation properties that are not required do not need to be set in advice configurations". For details see Advice Implementation Properties.

Example 1 **An advice with two properties:**

```
<advice name = "Advice1" ...>
```

```
<activity where = "Before">
<implementation.java className =
"com.tibco.bw.poa.samples.JMSPropertyChangerWithConfig"/>
</activity>
<properties>
<property name = "propertyToModify">Bar</property>
<property name = "propertyToModifyValue">BarValue1</property>
</properties>
</advice>
```

In this case, the advice implementation (i.e. the `JMSPropertyChangerWithConfig` java class) defines two configurable member variables:

- `propertyToModify`
- `propertyToModifyValue`

These two properties are visible and configurable from an advice, in this case Advice1. The `@name` attribute of the `<property>` element *must* match the name of the advice implementation property. If a match is not found, the BW Engine throws an `AspectException` at the time the engine gets initialized.

Another advice could share the same implementation and configure it in a different way.

Example 2 **Advice2 configures its implementation to mutate the same property "Bar" but with a different value:**

```
<advice name = "Advice2" ...>
<activity where = "Before">
<implementation.java className =
"com.tibco.bw.poa.samples.JMSPropertyChangerWithConfig"/>
</activity>
<properties>
<property name = "propertyToModify">Bar</property>
<property name = "propertyToModifyValue">BarValue2</property>
</properties>
</advice>
```

Example 3 **Advice3 configures its implementation to mutate a different property altogether:**

```
<advice name = "Advice3" ...>
<activity where = "Before">
<implementation.java className =
"com.tibco.bw.poa.samples.JMSPropertyChangerWithConfig"/>
</activity>
```

```
<properties>
<property name = "propertyToModify">Abc</property>
<property name = "propertyToModifyValue">Xyz</property>
</properties>
</advice>
```

The advices cannot share the same "advice implementation *instance*". They can certainly share the same "advice implementation".

Advice Ordering

Advices get injected in processes based on the point cuts that are associated with them. At run-time multiple advices can be injected in the same join point. Sometimes, the order in which these advices get executed is very important and needs to be controlled by the Aspect Developer. The BW engine executes the advices in a specific order, which is computed based on a priority order associated with each advice.

Executing of advices also depends on the process execution flow; which decides the availablity of input or output XML document which is to be used by the advice.

Since the advices that are injected in a specific join point can be specified either in the same or in different aspect files, the priority order of an advice is computed based on:

1. The value of the `@order` attribute that is specified on the aspect that defines the advice.
2. The location (such as the position) of the advice element inside the aspect definition (such as the location inside the XML file). An advice with a position that is closer to the root element `<aspect>` has a higher priority than an advice that is located farther than the root element.

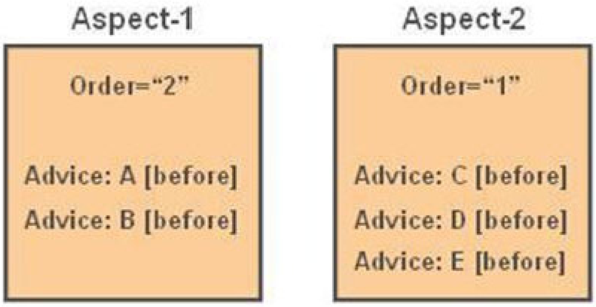
The optional `@order` attribute that can be specified on an aspect is used to establish a priority order between different aspects that are applied to a BW project. All advices defined in the same aspect share the same `@order` attribute value. This value can range between 1 and 255. The lower the number the higher the priority of the aspect. If the `@order` attribute is not specified, 255 is used by default, which means that the aspect has the lowest priority.

Although the `@order` attribute is `xsd:unsignedByte`, 0 is an invalid value. A validation error is thrown if used.

The order in which the advices are specified in an aspect file is very important. The BW engine uses this order to execute the advices that are run when a specific event occurs. This is true regardless of the event (for example, before executing an activity, after an activity returns successfully, and so on). For example, if a target activity throws an exception and there are "after" advices as well as "after throwing" advices injected after the target activity, the order in which these advices execute is influenced by the order in which they appear in the aspect file.

Figure 5 shows two aspect definitions - Aspect-1 and Aspect-2, each one defining a few "`Before`" advices. The order of Aspect-1 is "2" and the order of "Aspect-2" is 1, which means that Aspect-2 has a higher priority order than Aspect-1.

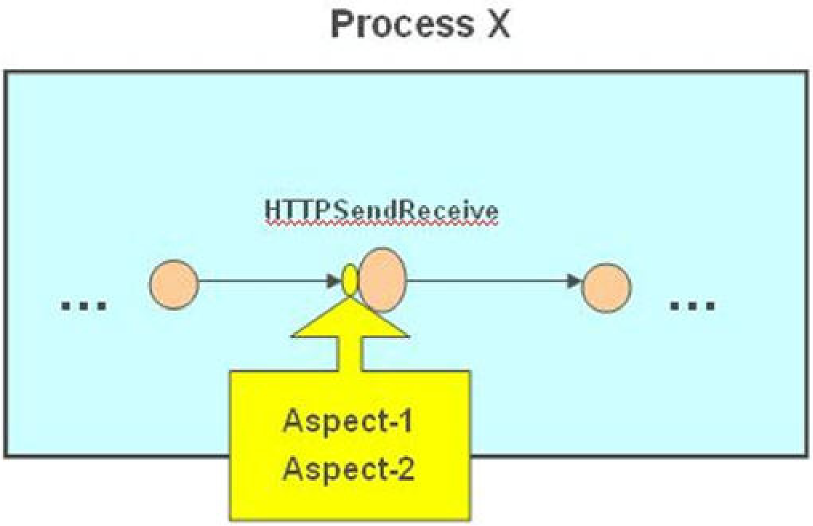
Figure 5 Aspect Definitions



Assume that the order in which these advices appear in the aspect definitions is exactly the other in which they are shown in this diagram (For example, A is before B, C is before D, and D is before E).

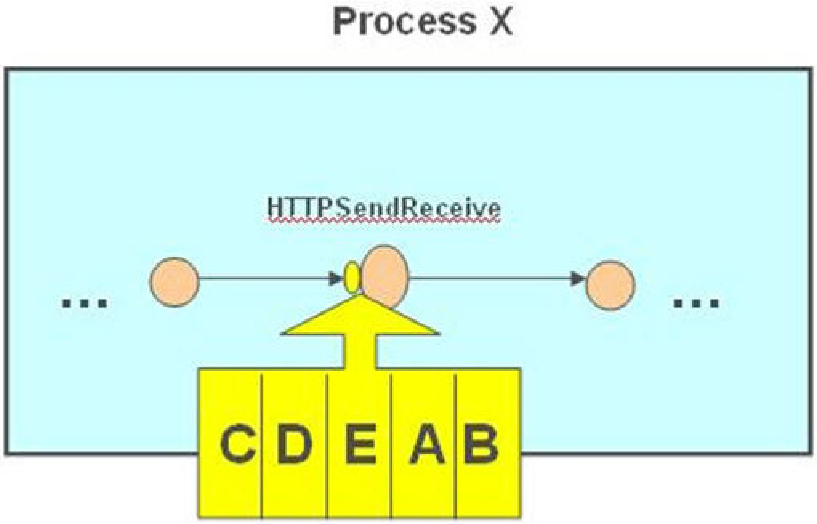
Figure 6 shows that after evaluating the point cuts associated with these advices, the BW Engine injects these two aspects before a specific activity of a process.

Figure 6 Injecting Two Aspects Before Specific Activity in a Process



Since Aspect-2 has a higher priority than Aspect-1, all its "Before" advices will execute before the advices defined in Aspect-1. The order of execution of the advices defined in the same aspect is given by the order in which they appear in the aspect XML file. Therefore, the order in which these advices get executed is shown in Figure 7.

Figure 7 Order of Execution of the Advices



If two aspects have the same order, then their corresponding advices execute in an order that is not deterministic.

Advice Execution Model

The advices that are injected in a specific join point execute always in sequence. As shown in Advice Ordering, the order in which these advices execute is computed based on the way the aspects are configured as well as the type of the join point.

Here are the details about how these aspects get executed by focusing on other characteristics that are not related to ordering.

It is important to note that an advice always has access to the XML Document as well as the Process Context that is available in the join point where it is injected. The Process Context can provide access to other XML documents that were contributed by the previous activities (such as activities that executed before the join point).

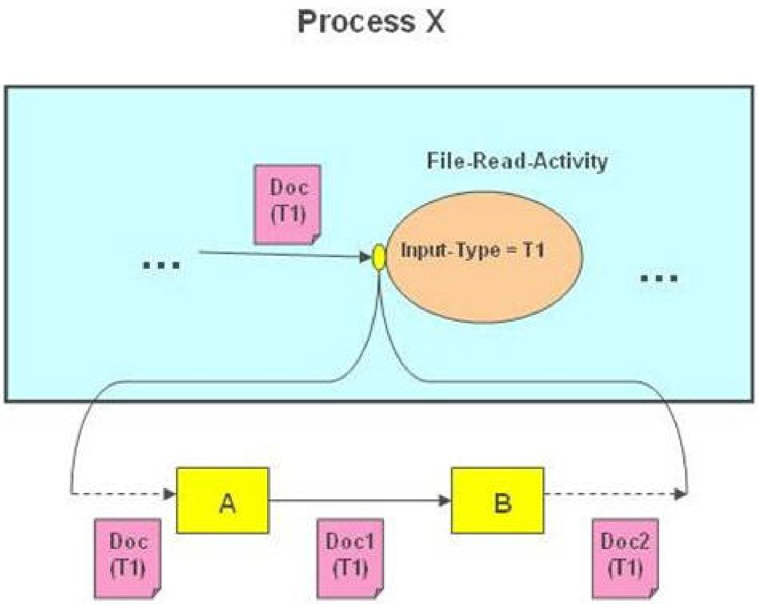
Advices do not contribute to the Process Context new XML documents that are visible to TIBCO ActiveMatrix BusinessWorks activities. Only activities can do that. XML documents contributed by advices are visible only to other advices that execute downstream in the process.

Do not abuse this way of sharing data between advice instances that execute in the same process.

The information available to an aspect for each of the supported join point is:

- The XML Document that is passed to a "`Before`" advice is the same XML Document that had been passed to the target activity, if the advice was not injected in the process. **The advice can alter the document but it cannot change its structure.** The XML Document that is produced by the advice must be valid against the same schema (such as the schema of the input element), which is defined by the target activity. The same rule applies to all the subsequent advices that are inserted in the same join point. This means that all the XML Documents that flow through all the "`Before`" advices that are injected in the same join point share the same schema. This is shown Figure 8.

Figure 8 XML Document Passed to a "Before" Advice

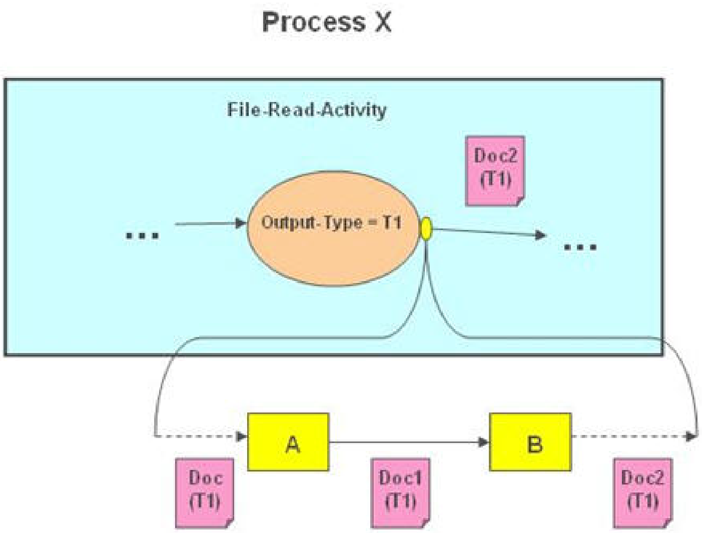


In Figure 8, the File-Read-Activity's input type is T1. The advices that are injected before this activity receive at run-time the XML Documents, that are valid against the same schema (such as, they are instance of T1).

- The XML Document passed to an "`After Returning`" advice is the XML Document that is generated by the target activity, if the activity has an output type or it is a null object otherwise. **The advice can alter the document but it cannot alter its structure.** The XML document that is produced by the advice must be valid against the same schema (such as, the schema of the output type), which is defined by the target activity.
- The same rule applies to all the subsequent advices that are inserted in the same join point. This implies that all the XML Documents that flow through

all the "`After Returning`" advices that are injected in the same join point, share the same schema. This is shown in Figure 9.

Figure 9 XML Document Passed to An "After Returning" Advice



- The XML Document that is passed to an "`After Throwing`" advice, is the XML Document that represents the exception thrown by the target activity.

⚠ The advice can alter the document (for example, change the exception message, add more information to the exception, and so on) but cannot alter its structure.

- The XML document that is produced by the advice must be valid against the schema of the output exception type, which basically means that the advice cannot change the exception type that is thrown by the activity.

  Since an activity can report multiple exception types, an "`After Throwing`" advice has to support multiple XML schema types, which usually makes it difficult to develop. If an advice is interested in a particular exception type, it can use the `@exceptionType` attribute to specify its `QName`. The value of this attribute has the following format:

```
<exceptionType> ::= <exceptionTypeQName> (", "
<exceptionTypeQName> )*

<exceptionTypeQName> ::= "{" <exceptionTypeNamespace> "}"
<exceptionTypeLocalName>
```

where,

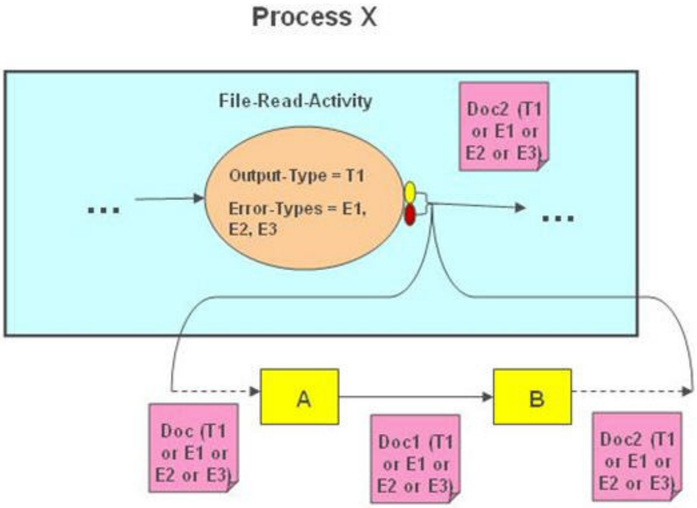
- `<exceptionTypeNamespace>` is the target namespace of the XML Schema that defines the exception type.
- `<exceptionTypeLocalName>` is the name of the exception type.

This syntax supports multiple `QName` values incase an advice may want to register its interest in more than one exception type.

To provide a flexible way to select a wide range of exception types, both the exception type namespace as well as the exception type name support `"*"` wildcards. This is the only wildcard supported.

In such cases, the advice is executed only when an exception instance of that type is thrown by the target activity. All the subsequent advices that are inserted in the same join point follow these rules. The "`After Throwing`" advices should not throw back the exception that is passed to them. Instead, they should return the exception message when they finish executing.. If an exception is thrown by an "`After Throwing`" advice, the engine treats it as any other exception thrown by other types of advices. Figure 10 shows an example of two "`After Throwing`" advices.

Figure 10 Example of Two "After Throwing" Advices

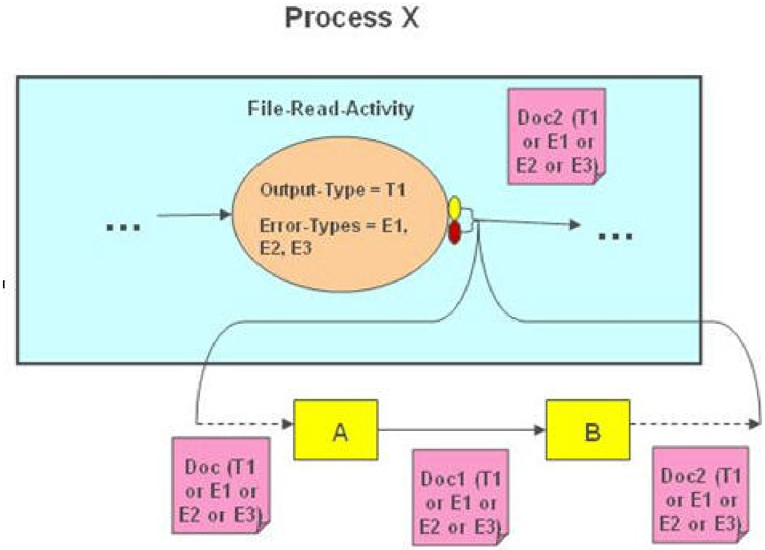

Process X
File-Read-Activity
Doc2 (T1 or E1 or E2 or E3)
Output-Type = T1
Error-Types = E1, E2, E3
...
...
A
B
Doc (T1 or E1 or E2 or E3)
Doc1 (T1 or E1 or E2 or E3)
Doc2 (T1 or E1 or E2 or E3)

- The XML Document that is passed to an "`After`" advice represents either the output of the target activity or one of its exceptions. The advice can alter the document (for example, change the output message, change the exception message, and so on) but it cannot alter its structure. The XML document produced by the advice must be valid against either the schema of the output

XML document or the schema that describes the exception thrown by the activity. This basically means that the same type of document that is passed to an "`After`" advice is passed to any other subsequent "`After`" advices that might run in the same join point. While writing an "`After`" advice a developer should be careful since it needs to handle different semantics (for example, successful returns as well as multiple exception types).

An "`After`" advice cannot use the `@exceptionType` attribute to register its interest in a particular exception type. If an exception is thrown by an "`After`" advice, the BW engine treats it as any other exception thrown by other types of advices. Figure 11 shows an example of two "`After`" advices.

Figure 11 Example of Two "After" Advices


Process X
File-Read-Activity
Output-Type = T1
Error-Types = E1, E2, E3
...
...
Doc2 (T1 or E1 or E2 or E3)
A
B
Doc (T1 or E1 or E2 or E3)
Doc1 (T1 or E1 or E2 or E3)
Doc2 (T1 or E1 or E2 or E3)

Any exception thrown by an advice is propagated as a `RuntimeException` and handled by the BW engine as thrown by the target activity. The engine processes it based on the business logic defined in the process containing the target activity.

Packaging and Deployment of Aspects

This section describes the structure of an `aspect JAR` and the deployment process of `aspect libraries.`

Aspects are packaged together in JAR files, that are referred to as "`aspect JARs`".

An `aspect JAR` is a JAR file that contains a file with the "`.AMF`" (Aspect Manifest File) extension in the `META-INF` folder. **There can be only one file with the "`.AMF`" extension in that folder.**

The Aspect Manifest File contains information about where all the aspects are located inside the aspect JAR file. These locations can be specified in two ways:

- By specifying a folder name inside the JAR file. All aspects that are part of that folder will be loaded by the BW engine. The BW engine will not locate aspects in the subfolders. The root folder is specified using `""` (which is an empty string).
- By specifying the full name (like including the folder location) of an individual aspect inside the JAR file.

The AMF file is an XML file that *must* be valid against the Aspect Manifest File XML Schema. When loading an aspect JAR file, the BW engine validates the Aspect Manifest File against this schema. If validation errors are found, the BW engine throws an `AspectException`.

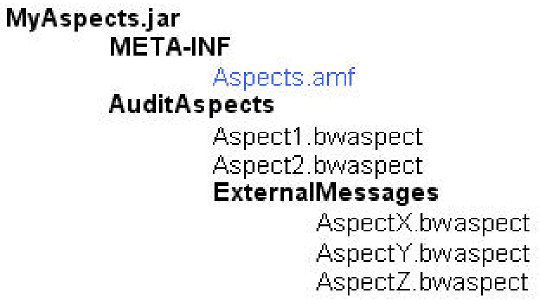
The following snippet shows the Aspect Manifest File pseudo-schema:

```
<bwpoa xmlns = "http://schemas.tibco.com/bw/poa/manifest" ...>
<aspects>
<aspectsFolder>...</aspectsFolder>*
<aspect>...</aspect>*
</aspects>
</bwpoa>
```

A JAR file that contains aspects but does not contain an Aspect Manifest File, is not recognized by the BW engine as an aspect JAR. Hence none of the aspects defined in that JAR are loaded by the BW engine.

Examples of Aspect Manifest File

An aspect JAR with the following structure:



```
MyAspects.jar
    META-INF
        Aspects.amf
    AuditAspects
        Aspect1.bwaspect
        Aspect2.bwaspect
        ExternalMessages
            AspectX.bwaspect
            AspectY.bwaspect
            AspectZ.bwaspect
```

If the aspects.amf file is the following:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<bwpoa
```

```
xmlns = "http://schemas.tibco.com/bw/poa/manifest"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation = "http://schemas.tibco.com/bw/poa/manifest
bwpoa.xsd">
<aspects>
<aspectsFolder>auditAspects</aspectsFolder>
</aspects>
</bwpoa>
```

Then only `Aspect1.bwaspect` and `Aspect2.bwaspect` are loaded by the BW engine. The aspects specified in the ExternalMessages folder are not loaded.

However, if the aspects.amf file is the following:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<bwpoa
xmlns = "http://schemas.tibco.com/bw/poa/manifest"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation = "http://schemas.tibco.com/bw/poa/manifest
bwpoa.xsd">
<aspects>
<aspectsFolder>auditAspects</aspectsFolder>
<aspectsFolder>auditAspects/ExternalMessages</aspectsFolder>
</aspects>
</bwpoa>
```

Then,

all the aspects defined in this aspect JAR are loaded by the BW engine. The same behavior could be accomplished by specifying some or even all of the aspects individually. For example, the following manifest file would have the same result as the previous one:

```
<?xml version = "1.0" encoding = "UTF-8"?>
<bwpoa
xmlns = "http://schemas.tibco.com/bw/poa/manifest"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation = "http://schemas.tibco.com/bw/poa/manifest
bwpoa.xsd">
<aspects>
<aspect>auditAspects/Aspect1.bwaspect</aspect>
<aspect>auditAspects/Aspect2.bwaspect</aspect>
<aspectsFolder>auditAspects/ExternalMessages</aspectsFolder>
</aspects>
</bwpoa>
```

Deploying Packaged Aspects in BW Engine

Multiple aspect JARs can be deployed in a BW Engine. When the engine starts up, all these aspect JARs are loaded and their aspects are weaved into the BW Project.

All aspect JARs loaded by the BW engine at run-time must be located in the same folder.

The name of the folder is specified through the following java system property:

`aspectPath`

This property can be specified in the `bwengine.tra`. For example:

```
# BW Aspect Definition Files
java.property.aspectPath
%BW_HOME%/examples/poa/Scenario1/Aspects
```

Same in the case of folders specified inside aspect JARs, the BW engine does not look at sub-folders when loading aspect JARs. The engine loads only the JARs that are part of this top level folder.

The engine writes out information about all advices that are woven in the TIBCO ActiveMatrix BusinessWorks project. For each process, the engine analyses the advices that are configured to be woven and skips the ones that cannot be woven due to product limitations. After finishing weaving of a process, the engine logs out all advices that were skipped. For more information about the advices that cannot be skipped, read the release notes or contact TIBCO Support.

Advice Implementations

TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in provides support for executing advices developed in Java. TIBCO ActiveMatrix BusinessWorks defines both a Java API as well as a set of annotations that have to be used for creating Advice Implementations.

Relationship between an advice and its implementation is very important. **An advice is a configured instance of an advice implementation**. Multiple advices can be implemented with the same implementation. Therefore, there is an N-to-one relationship between advices and their associated java implementation.

An advice implementation provides the business logic of an advice.

Multiple advices can be implemented with the same Java class.

Java Annotations for Advertising Advice Implementation Metadata

TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in defines a couple of java annotations that are used for advertising aspect specific metadata. These annotations are defined in the package `com.tibco.bw.poa.runtime.annotation`. This section describes these annotations.

The @AdviceImpl Java Annotation

This annotation is used for tagging java classes as advice implementations. This annotation is specified on a class, in the following way:

```
@AdviceImpl
public class MyAdviceImplementation … {
}
```

Table 4 describes the 5 optional parameters.

Table 4 @AdviceImpl Java Annotation Optional Parameters

Name	Type	Description
`scope`	String	Specifies the instantiation scope of the advice implementation.
`dataAccess`	String	Advertises the data access mode (for example, read/only vs. read/write).

Table 4 @AdviceImpl Java Annotation Optional Parameters

Name	Type	Description
`hibernatesJobs`	Boolean	Advertises whether the advice implementation hibernates jobs
`targetKind`	String	Advertises the kind of the target, or join point where the advice implementation gets instantiated.
`targetFilter`	String	Advertises an activity type name, which can be used to further narrow down the scope of the target.

scope

Advice implementations get instantiated in different scopes, which can be controlled using the `scope` parameter. The valid values of this parameter are the following:

Values	Description
ADVICE	The advice implementation is instantiated once for every advice instance that is using it.
APPLICATION	One instance of the advice implementation is created for an application. In TIBCO ActiveMatrix BusinessWorks 5.9, the application equals to a BW Project, which means that one instance of the advice is created in a BW engine.

If the scope parameter is set to a different value than the one mentioned above, the BW engine throws an `AspectException` at run-time, when the engine gets initialized.

This example of an advice implementation that uses the application scope:

```
@AdviceImpl (
scope = "APPLICATION"
)
public class MyAdviceImplementation … }
}
```

The default value of this parameter, which is used when the parameter is not explicitly set by the user, is "ADVICE".

dataAccess

Advice implementations access the data in a way that can be classified in two categories:

- Advice implementations that need **read-only** access to the data
- Advice implementations that need **read-write** access to the data

An advice implementation advertises the category it belongs to by setting the `dataAccess` parameter. The valid values of this parameter are the following:

Values	Description
READ-WRITE	The advice implementation mutates the data, hence needs read/write access to it.
READ-ONLY	The advice implementation *does not* mutate the data, hence needs read-only access to it.

If the `dataAccess` parameter is set to a different value than the ones mentioned above, the BW engine throws an `AspectException` at run-time, when the engine gets initialized.

Following is an example of an advice implementation that mutates the XML document:

```
@AdviceImpl (
dataAccess = "READ-WRITE"
)
public class MyAdviceImplementation … {
public void execute(N input, AspectProcessContext context) {
// the advice implementation mutates the data here…
}
}
```

The default value of this parameter, which is used when the parameter is not explicitly set by the user, is "READ-ONLY". Hence, an advice implementation that mutates the data must explicitly set the dataAccess parameter to "READ-WRITE". If an advice implementation that mutates the data does not set this parameter appropriately, a `ClassCastException` is thrown at run-time, when the advice implementation tries to use a mutable XML processing contex.

hibernatesJobs

Advice implementations can hibernate and resume Jobs. If an advice implementation takes advantage of this feature, it must explicitly declare that it does it through the `hibernatesJobs` parameter. The valid values of this parameter are:

Values	Description
true	The advice implementation calls an API to hibernate a job.
false	The advice implementation does not call an API to hibernate a job.

Following is an example of an advice implementation that hibernates jobs:

```
@AdviceImpl (
hibernatesJobs = true
)
public class MyAdviceImplementation … {
public void execute(N input, AspectProcessContext context) {
…
if (need_to_hibernate_job) {
context.setHibernateJobEnabled(0);
}
}
}
```

The default value of this parameter, which is used when the user doesn't explicitly set it, is `false`. **By default, advice implementations do not hibernate jobs.** If an advice implementation attempts to call the API to hibernate a job without explicitly setting the `hibernatesJobs` parameter to `true`, an `AspectException` is thrown at run-time. This exception may result in the job to be terminated abnormally, depending on how exceptions are handled at the process level.

When accessing the incoming XML document, advice Implementations may or may not be dependent on a particular schema. For example, an advice implementation that changes the value of a JMS property in a JMS message sent by the `JMSQueueSendReceive` activity is dependent on the format (like XML schema) of the `JMSQueueSendReceive` activity's input message.

For this advice implementation to work, it has to be hardcoded to expect the XML document in this specific format.

Activities that are hardcoded to a specific XML schema can only be instantiated and executed in a context where the incoming document conforms to that particular schema.

A way is provided to the advice implementation developer to declare a dependency of a particular instantiation context through the use of two parameters defined as part of the **`@AdviceImpl` annotation: `targetKind` and `targetFilter`**.

targetKind

This is used for specifying the kind of the target or join point where the advice is instantiated. This parameter can have one of the following values:

Values	Description
ACTIVITY-BEFORE	The advice implementation must be instantiated only before activities.
ACTIVITY-AFTER-RETURNING	The advice implementation must be instantiated only after an activity, on the path that executes when the activity returns successfully.
ACTIVITY-AFTER-THROWING	The advice implementation must be instantiated only after an activity, on the path that executes when the activity throws an exception.
ACTIVITY-AFTER	The advice implementation must be instantiated only after an activity, on the path that executes regardless on whether the activity returns successfully or throws an exception
"" (empty string)	The advice implementation is not dependent on a particular context .

If the `targetKind` parameter is set to a different value than the ones mentioned above, the BW engine throws an `AspectException` at run-time, when the engine gets initialized.

The default value of this parameter, which is used when the user doesn't explicitly set it, is `""`. By default, advice implementations do not depend on a particular context (can be executed anywhere).

targetFilter

When the `targetKind` parameter is set, the developer of the advice implementation can also specify a target filter, using the `targetFilter` parameter, which is used to further narrow down the scope of the instantiation context. Using this parameter, the developer can specify the type of the activity around which the implementation can be instantiated. For example, the developer can specify the advice implementation that can only be instantiated in the context of a `FileReadActivity`. The value of the parameter (the filter) is the same as the value of the `activity()` primitive's "`type`" parameter that is defined as part of the Point Cut Query Language.

The default value of this parameter, which is used when the advice implementation developer does not explicitly set it, is `""`. By default, advice implementations can be executed in any context.

Following is an example of an advice implementation that must be instantiated and executed before `JMSQueueSend` activities.

```
@AdviceImpl(
targetKind = "ACTIVITY-BEFORE",
targetFilter = "bw.JMSQueueSendActivity"
)
public class JMSPropertyChanger ... {
}
```

When the Aspect engine gets initialized, the engine checks these two parameters and uses them to validate the aspect configurations. If an advice is about to be instantiated in a context that is not valid, the engine throws an exception.

An exception is thrown at engine initialization time and not at run-time during the processing of an incoming request.

This is very important as it ensures that the engine failure is fast when aspects are not properly configured.

A more complicated example shows how all these parameters can be used together. The following is a singleton advice implementation that requires read-write access to the data, that hibernates jobs and must be instantiated and executed before `JMSQueueSend` activities:

```
@AdviceImpl(
scope = "APPLICATION",
dataAccess = "READ-WRITE",
hibernatesJobs = true,
targetKind = "ACTIVITY-BEFORE",
targetFilter = "bw.JMSQueueSendActivity"
)
public class JMSPropertyChanger ... {
}
```

Enable Aspect Engine Logging

To enable tracing for Aspect Engine, set the `AspectEngine.Trace` property to `true` in a `cfg` file and this file should be passed as an argument to the BusinessWorks engine.

The @Property Java Annotation

This annotation is used for marking public member variables defined in advice java implementations as properties (like advice implementation properties).

Following is an example of an advice implementation property:

```
public class AdviceImplExample<I, U, N extends I, A extends I, S,
T, X>
 extends SyncAdvice<I, U, N, A, S, T, X> {
@Property
public String currency = "$";
...
}
```

Advice Implementation Properties

Advice implementations can define configuration properties that are used for configuring the behavior and execution of the java class. For example, an advice implementation that changes the value of a JMS Property may choose to define the name of the JMS property using an advice implementation property. TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in makes these advice implementation properties visible and configurable from advices. This means that two advices that have the same implementation (that are implemented with the same java class) can specify different values for the same property. In the example in The @Property Java Annotation, there could be two or more advices that want to mutate JMS Properties. These advices can share the same implementation and each one can configure it with a different JMS Property name.

@Property Java Annotation Optional Parameters

Name	Type	Desription
required	Boolean	Whether the property is required to be set in the aspect file.

required

The advice implementation developer can declare an advice property either required or optional by setting the required parameter.

A required advice property must be set in every advice that uses the implementation class that defines it. The ActiveMatrix BusinessWorks engine throws an exception at the time the engine gets initialized, if it finds a required advice implementation property that is not set in an advice.

Values	Description
true	The advice property is required.
false	The advice property is optional.

The default value, which is used when the "required" parameter is not set, is `true`. Therefore, by default, advice properties are required to be set in aspect files.

Advice implementations advertise configuration properties through the use of the `@Property` annotation. The following example shows an advice implementation with two properties:

- `package com.tibco.bw.poa.samples`
- `import com.tibco.bw.poa.runtime.annotation.Property`

```
@AdviceImpl(
dataAccess="READ-WRITE",
targetKind="ACTIVITY-BEFORE",
targetFilter="bw.JMSQueueSendActivity"
)
public class JMSPropertyChangerWithConfig<I, U, N extends I, A
extends I, S, T, X> extends SyncAdvice<I, U, N, A, S, T, X> {
@Property
public String propertyToModify ="foo";

@Property (required = false)
public String propertyToModifyValue = "defaultFooValue";
...
}
```

Restrictions Imposed by TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in for Advice implementation Properties

- Only properties of String type are supported. If the Aspect Engine detects a property of a type different than String, it throws an exception at initialization time.
- Member variables that are exposed as advice configuration properties must be declared public. If the BW Engine detects a property that is not declared public, it throws an exception at initialization time.

At run-time, when an advice is instantiated, the BW engine injects in the advice instance the property values that are specified in the aspect XML file. **Setter and Getter methods do not have to be available in the java class to provide access to the member variables.** The BW engine can inject these property values by accessing the member variables directly.

All advices that share the same implementation instance (for example when using an implementation configured with an `APPLICATION` scope) must have the same property values. The BW Engine configures the implementation instance at the time the first advice that is using it gets instantiated. If a subsequent advice that uses the same implementation instance is instantiated, the BW engine validates that all its properties have the same values as the properties set on the advice implementation instance. If a mismatch is found, the BW Engine throws an exception.

To avoid unnecessary null pointer exceptions at run-time, it is highly recommended that all advice implementation properties have default values specified in the java class.

Refer Advice Configuration Properties about how property values are set in aspect files.

Scopes

At run-time, during its initialization process, the BW Engine weaves aspects into BW processes. As part of this process, the BW engine instantiates advices and injects them in different join points inside processes. Since each advice has an implementation associated with it, which is represented as a Java class, the BW engine either creates an instance of this class or takes one from a pool of already created object instances. ActiveMatrix BusinessWorks 5.9 supports binding multiple advice instances to the same advice implementation java class instance through the concept of scoping.

The two Advice Implementation Scopes supported are:

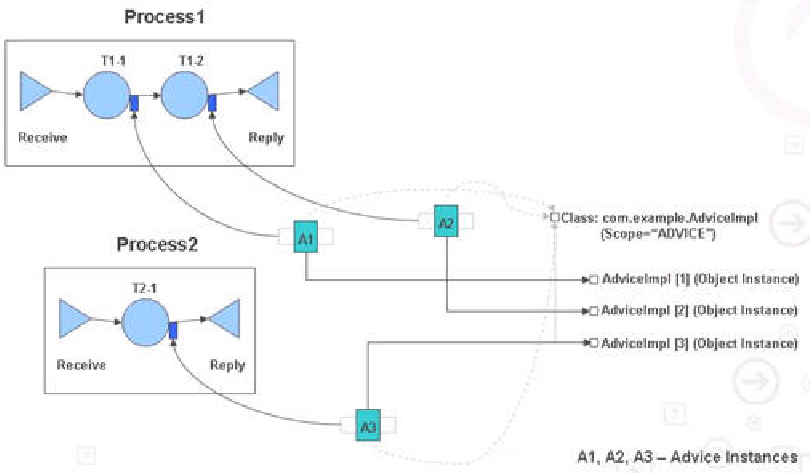
- Advice
- Application

Since scope is an attribute of the advice implementation, it is configured at the java class level in an annotation. This means that all advices using the same advice implementation java class have the same scoping configuration.

Advice Scope Mode

The **Advice** scope is used when a new advice implementation object instance must be created for every advice instance. This mode is usually used when multiple advices that share the same implementation do not need to share any state. Figure 12 shows this Advice Scope mode.

Figure 12 Advice Scope Mode


Process1
T1.1
T1.2
Receive
Reply
A1
A2
Class: com.example.AdviceImpl
(Scope="ADVICE")
Process2
T2.1
Receive
Reply
AdviceImpl [1] (Object Instance)
AdviceImpl [2] (Object Instance)
AdviceImpl [3] (Object Instance)
A3
A1, A2, A3 – Advice Instances

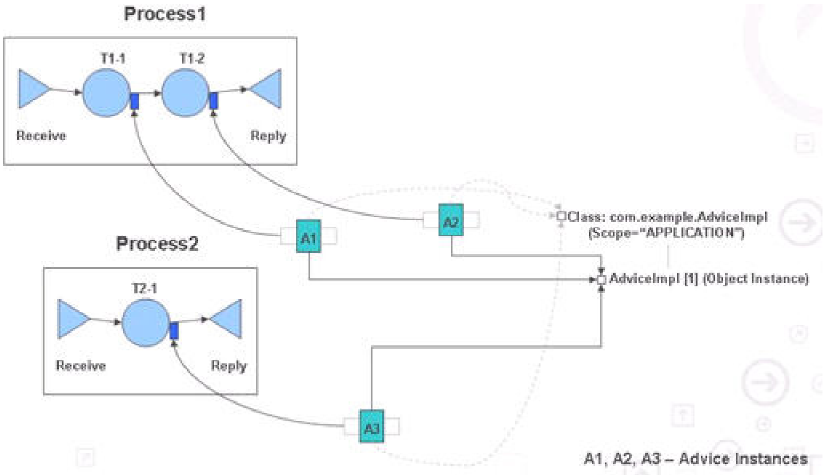
Despite one advice implementation object instance being created for every advice instance, the developer of the advice implementation java class will still have to be aware of data concurrency issues.

Since an advice instance runs in a multi threaded environment potentially serving multiple process instances (like jobs) at the same time, each job is executed in a different thread.

Application Scope Mode

The **Application** scope is used when the developer of the advice implementation java class wants to ensure that only one advice implementation object instance is created for the entire application. In TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in, this means one object instance per engine. Hence, all advice instances that are implemented with the same implementation share the same advice implementation object instance. Figure 13 shows the Application Scope mode.

Figure 13 Application Scope Mode


Process1
T1-1
T1-2
Receive
Reply
Process2
T2-1
Receive
Reply
A1
A2
A3
Class: com.example.AdviceImpl
(Scope="APPLICATION")
AdviceImpl [1] (Object Instance)
A1, A2, A3 – Advice Instances

XML Document Access

When it executes, an advice implementation has access to the XML document that is available in a particular context (like, join point). This XML Document is build based on a schema, which is again dependent on the context. For example, when it runs before an activity, the XML Document must be valid against the XML Schema that defines the input type of the activity.

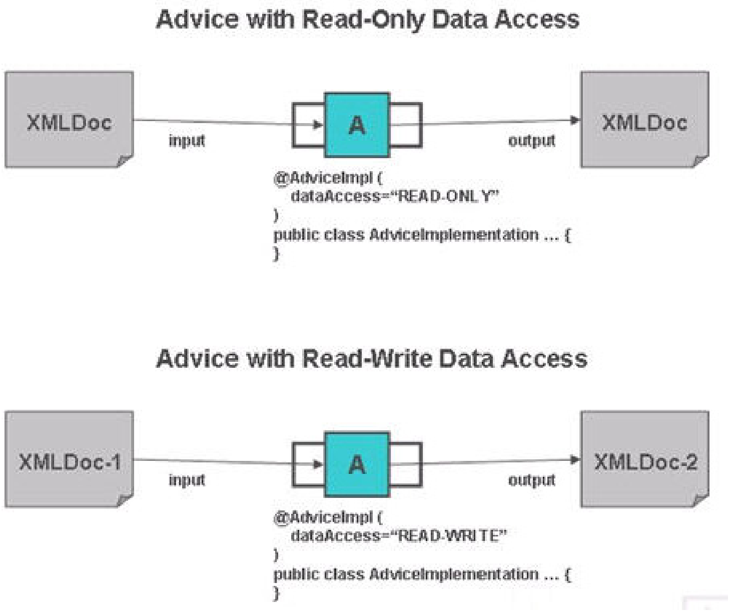
In order to provide a deterministic behavior that allows the engine to fail fast when configuration issues are detected, TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in provides a declarative way of specifying metadata about advice implementations. This metadata is primarily driven by two main questions a developer of an advice implementation needs to answer.

- Does the advice implementation mutate the XML Document?
- Does the advice implementation expect the document to be valid against a particular XML Schema?

Not all advice implementations mutate the incoming XML Document. If the engine knows that an advice does not mutate the XML document, it can perform some optimizations, such as not requiring a revalidation of the document after the execution of the advices. In TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in, an advice cannot mutate the incoming XML Document unless it explicitly states that it does it, in the advice implementation metadata. For details about how to configure an advice to allow the mutation of the XML Document, refer The @AdviceImpl Java Annotation.

Figure 14 shows two advices, one that mutates the XML Document and another that does not mutate the XML Document.

Figure 14 Read-Only vs. Read-Write Data Access



Advice with Read-Only Data Access

```
@AdviceImpl (
   dataAccess=“READ-ONLY”
)
public class AdviceImplementation ... {
}
```

Advice with Read-Write Data Access

```
@AdviceImpl (
   dataAccess=“READ-WRITE”
)
public class AdviceImplementation ... {
}
```

The XML Document is passed as an input parameter to the `execute()` method that is defined as part of the Advice Implementation Java class. The following is the signature of the method:

```
public N execute(N inputDoc, AspectProcessContext context)
throws AspectException;
```

The XML Document is the first input parameter (like, "inputDoc"). TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in uses gXML (Generic XML) as the data model, which allows it to support multiple underlying XML tree models such as DOM, Axiom, etc. For more information about gXML and general information on how to manipulate an XML Document with gXML, refer **References**.

In order to manipulate an XML Document, the advice implementation must get access to the gXML processing context object, which is instance of `org.gxml.sa.GxProcessingContext`. This object can be retrieved by an advice implementation from the advice's context, in the following way:

```
GxProcessingContext<I,U,N,A,S,T,X> pContext =
getAdviceContext().getGxProcessingContext();
```

This object is then used for traversing and pulling information from the XML Document. This object cannot be used for mutating an XML Document, though. In order to mutate an XML Document, the advice implementation needs to get a mutable gXML processing context, which is instance of `org.gxml.sa.GxProcessingContextMutable`. Since this class extends `GxProcessingContext`, the way to retrieve a mutable processing context is very similar to the way to retrieve the immutable processing context (like, note the extra type casting):

```
GxProcessingContextMutable<I,U,N,A,S,T,X> pContext =
GxProcessingContextMutable)getAdviceContext().getGxProcessingConte
xt();
```

If an advice implementation type casts the returned value of `getGxProcessingContext()` to a mutable processing context without explicitly setting the `@AdviceImplementation` or `dataAccess` parameter to "`READ-WRITE`", the previous call throws a `ClassCastException` at run-time. That is since the engine injects a mutable processing context in an advice instance only if its implementation is mutable.

Packaging and Deployment of Advice Implementations

Advice implementations are packaged in JAR files and need to be available in the `CLASSPATH` at run-time in order for the engine to properly instantiate advices. It does not really matter how these advice implementations are packaged in JAR files. What is important is that all the advice implementations (like, java classes) are referenced by advices to be available in the `CLASSPATH` at run-time, at the time the BW engine gets initialized.

TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in does not support loading a JAR file in the `CLASSPATH` at run-time, after the engine has been initialized. Therefore, unless it gets restarted, the BW engine cannot execute an advice implemented with a java class that was not available in the `CLASSPATH` at initialization time.

The TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in installer creates a "`lib`" folder under `BWAA_HOME` and adds it to the `CLASSPATH`. This folder can be used for storing all advice implementation libraries (that is, JAR files).

Chapter 2 Point Cut Query Language

This chapter defines Point Cut Query Language and provides the necessary information needed to build complex expressions.

Topics

Introduction

TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in defines a simple query language used for writing point cut expressions. This language consists of a small set of primitives. Each one of these primitives is designed to narrow down the scope of the search based on certain conditions (for example, search for all File Read activities, search for all processes that have their namespace starting with "http://example.com/", etc.).

Query Language Primitives

The query language defines the following four primitives:

- **activity** (selection-expression) - Selects all activities that match a specific expression.
- **process** (selection-expression) - Selects all processes that match a specific expression.
- **project** (selection-expression) - Narrows down the scope of the search to projects that match a specific expression.
- **engine** (selection-expression) - Narrows down the scope of the search to engines that match a specific expression.

These primitives can be combined to form complex expressions. The selection-expression, which is specified as part of these primitives, has a generic syntax that is not dependent on the actual primitive. Each primitive defines a set of properties that can be used in the expression. For example, the `activity()` primitive defines a `type` property that can be used to filer out activities based on their type. The evaluation of the expression results in the selection of a set of join points.

If there are no join points selected, then the application runs without executing any advices.

Following is the Backus-Naur Form (BNF) definition of the point cut expression.

```
<point-cut-expression> :: = <primitive> (  "&amp;&amp;" <primitive>
)*

<primitive> ::=  "activity(" <selection-expression> ")"  |
"process(" <selection-expression> ")"  |

"project(" <selection-expression> ")"  |  "engine("
<selection-expression> ")"

<selection-expression> ::= <selection-expression-part> ( <and-or>
<selection-expression-part> )*

<selection-expression-part> ::=
<selection-expression-part-simple> |
<selection-expression-part-wrapped>

<selection-expression-part-simple> ::=  ( <propertyName>  "="
<propertyValue> )  |
      ( "!("  <property-name> "=" <property-value> ")" )

<selection-expression-part-wrapped> ::=   "("
<selection-expression-part-simple> ")"

<and-or> ::= "&amp;&amp;"  |  "||"
```

`<propertyName>` - see the tables

`<propertyValue>` - see the table

Despite all these primitives being optional, one of them must always be specified in a point cut expression.

- **If the `activity()` primitive does not appear in the expression then,** the BW engine selects all activities that are defined as part of the project. In this case the BW Engine treats this as having activity (`name="*"`) in the expression.
- **If the `process()` primitive does not appear then,** the BW Engine treats this as having process (`name="*"`) in the expression.
- **If the `project()` primitive does not appear then,** the BW Engine treats this as having project (`name="*"`) in the expression.
- **If the `engine()` primitive does not appear then,** the BW Engine treats this as having engine (`name="*"`) in the expression.

Properties Defined for activity() primitive

Property Name	Property Type	Property Value	Description
name	String		The name of the activity.
type	String		The type of the activity.
kind	String	`"event-source"` | `"signal-in"` | `"activity"`	The flavor of the activity (for example, event-source, signal-in or regular activity).
description	String		The description of the activity.

type

"type" is an ID that uniquely identifies a particular type of activity. To get the ID of a particular activity, refer to the table available in Appendix A, ActivityTypes. The rest of this section describes the algorithm for building these IDs. This is useful to know since there are activities that do not get shipped out of the box with TIBCO ActiveMatrix BusinessWorks and therefore they are not mentioned in Appendix A, ActivityTypes.

This ID is computed in the following way:

```
<type> ::= "bw.<resource-type-suffix>"
```

where

the resource-type-suffix is given by the sequence of the characters that appear after the last "." character in the `<pd:resourceType>` element, which is serialized as part of every activity's configuration. To find the `resource-type-suffix` for a specific activity, the user has to open a process definition that contains that activity and check the `<pd:resourceType>` element that is serialized as part of its configuration.

For example, here is the configuration XML of a File Read Activity that appears in a process definition:

```
<pd:activity name="Read File">
        <pd:type>com.tibco.plugin.file.FileReadActivity</pd:type>

<pd:resourceType>ae.activities.FileReadActivity</pd:resourceType>
        <pd:x>224</pd:x>
        <pd:y>173</pd:y>
        <config>
            <encoding>binary</encoding>
        </config>
        <pd:inputBindings>
            <ns1:ReadActivityInputClass>
                <fileName>
                    <xsl:value-of
select="&quot;C:\test\foo.xml&quot;"/>
                </fileName>
            </ns1:ReadActivityInputClass>
        </pd:inputBindings>
    </pd:activity>
```

This example shows that the `resource-type-suffix` of the File Read Activity is "`FileReadActivity`". This means that this activity's type is "`bw.FileReadActivity`" in Point cut query language.

kind

"`kind`" is used to filter out activities based on their flavors. There are three flavors of activities, each one identified with a specific ID:

- **`event-source`** - An activity that is an event source or process starter (for example, File Event Source, JMS Queue Receiver, and so on).
- **`signal-In`** - An activity that is a signal-in (for example, File Signal-In, and so on).

- **`activity`** - An activity that is neither an event-source nor a signal-in (for example, File Read Activity, HTTP Send Receive Activity, and so on).

Since these properties are defined for the `activity() primitive`, they can only be used in the context of this primitive.

Properties Defined for process() primitive

Property Name	Property Type	Property Value	Description
name	String		The name of the process.
tns	String		The target namespace of the process.
kind	String	`"sub-process"` | `"regular"` |	The flavor of the process.
description	String		The description of the process.

kind

"`kind`" is used to filter out processes based on their flavors. There are two flavors of processes, each one identified with a specific ID:

- **`sub-process`** - A process that doesn't have an event source or process starter.
- **`regular`** - A process that has an event source or process starter.

Like the properties defined for the `activity()` primitive, the properties defined for the `process()` primitive can only be used in the context of this primitive.

Properties Defined for project() primitive

Property Name	Property Type	Property Value	Description
name	String		The name of the project.

This primitive is used for filtering out projects based on their name. This can be useful in organizations where all aspects are developed, packaged, and deployed together.

Properties Defined for engine() primitive

Property Name	Property Type	Property Value	Description
name	String		The name of the engine.

This primitive is used for filtering out engines based on their names. Similar to `project()` primitive, this primitive can be useful in organizations where all aspects are developed, packaged, and deployed together.

Use of Escape Character

While using double quote as part of the property value in a pointcut query, use backward slash ('\') as an escape character.

For example, activity (description= "this is how \" is used as part of property value").

Escape characters should be used for all the xml predefined entities (<, >, &, ' and ") in the aspect file.

Wildcard Support

In order to provide more flexibility and to simplify the writing of point cut expressions, the query language supports wildcards (`"*"`) in property values. These can be used either to specify the entire property value (e.g. `name="*"`) or to specify a part of it (for example, `name = "JMS*"`).

Property names are case insensitive and property values are case sensitive.

This means that activity (`Name = "foo"`) evaluates the same as activity (`name = "foo"`). However, activity (`name = "foo"`) does not evaluate the same as activity (`name = "Foo"`).

Parentheses Support

Parentheses up to level two are supported in selection-expression of this query language.

Correct Expression (level one):

```
activity ( (name="JMSReceiver" || name= "HTTPReceiver" ) &amp;&amp;
(type="bw.JMSQueueEventSource" || type = "bw.httpEventSource" ) )
```

Invalid Expression (level three):

```
process (name = "Test*" || (name = "notify" || (name =
"waitnotifyprocess*" &amp;&amp; tns = "http://waitnotifytns/*")))
```

Supported Operators

The following operators are supported.

EQUAL Comparison operator “=” (EQUAL) can be used while comparing property values. For details, refer to Examples of Point Cuts Defined Using Query Language.

NOT EQUAL Comparison operator “!=” (NOT EQUAL) can now be used while comparing property values. For details, refer to 8.

Examples of Point Cuts Defined Using Query Language

1. Select all the FileEventSource activities.

```
<pointcut name = "allFileEvsActivities">

<query queryLanguage =
"http://schemas.tibco.com/bw/poa/pointCutSelectionLanguage">

activity ( type = "bw.FileEventSource" )

</query>

</pointcut>
```

2. Select all activities of type JMS.

```
<pointcut name = "allJMSActivities">
<query queryLanguage =
"http://schemas.tibco.com/bw/poa/pointCutSelectionLanguage">
activity ( type = "bw.JMS*" )
</query>
</pointcut>
```

3. Select all FileRead and FileWrite activities.

```
<pointcut name = "fileActivities">
<query queryLanguage =
"http://schemas.tibco.com/bw/poa/pointCutSelectionLanguage">
activity ( type = "bw.FileReadActivity" || type =
"bw.FileWriteActivity" )
</query>
</pointcut>
```

4. Select all the event source activities.

```
<pointcut name = "eventSourceActivities">
<query queryLanguage =
"http://schemas.tibco.com/bw/poa/pointCutSelectionLanguage">
activity ( kind = "event-source" )
</query>
</pointcut>
```

5. Select all activities that have "`@TODO`" in their description.

```
<pointcut name = "allTODOActivities">
<query queryLanguage =
"http://schemas.tibco.com/bw/poa/pointCutSelectionLanguage">
activity ( description = "*@TODO*" )
</query>
```

```
</pointcut>
```

6. Select all the activities that belong to processes that have their namespace starting with "`http://example.org/`".

```
<pointcut name = "purchaseOrderActivities">
<query queryLanguage =
"http://schemas.tibco.com/bw/poa/pointCutSelectionLanguage">
process ( tns = "http://example.org/*" )
</query>
</pointcut>
```

7. Select all the File Write Activities with name starting with "`FileRead`", that are part of a process whose target namespace starts with `http://example.org/`. Select only the processes that are part of projects whose names start with "`HR`".

```
<pointcut name = "complex">
<query queryLanguage =
"http://schemas.tibco.com/bw/poa/pointCutSelectionLanguage">
activity (name = "FileRead*" &amp;&amp; type =
"bw.FileReadActivity") &amp;&amp; process ( tns =
"http://example.org/*" ) &amp;&amp; project ( name = "HR*" )
</query>
</pointcut>
```

8. Select all the activities with name starting with "`file`". Do not select all "`FileWrite`" activities.

```
<pointcut name = "DoNotFileWriteActivities">
<query queryLanguage =
"http://schemas.tibco.com/bw/aop/pointCutSelectionLanguage">
activity ( name = "file*" &amp;&amp; type  !=
"bw.FileWriteActivity" )
</query>
</pointcut>
```

Chapter 3 Asynchronous Advice Implementations

This chapter describes the Asynchronous Advice Implementations.

Topics

Introduction

The asynchronous model is designed for advice implementations that take some time to execute. These are usually advice implementations that communicate with external systems, perform input/output operations or perform tasks that can potentially bring down the performance of a ActiveMatrix BusinessWorks application.

An asynchronous advice implementation does *not* execute its business logic on the engine job thread. In other words, the ActiveMatrix BusinessWorks engine does *not* hold the job thread until an asynchronous advice completes its execution. This allows the engine to execute advices and/or activities that might exist in the process on *parallel tracks*, while the asynchronous advice is executing. The engine *does not*, however, continue executing the next advice in the join point, or the next activity on the same track before the asynchronous advice completes its execution.

A typical asynchronous advice implementation gets a thread from a thread pool in its `execute()` method and starts executing its business logic on it. Right after that it calls `AdviceController->setPending()` with the appropriate timeout and returns from its `execute()`. After its business logic completes and before the advice thread finishes executing, the advice implementation calls `AdviceController->setReady()` with the result object. Once the job thread is available to finish executing the asynchronous advice, the engine calls its `postExecute()`method by passing the result object received in the `setReady()`call. The advice implementation gets the opportunity to do any final job related cleanup operations before returning the final result object back to the engine.

If the advice implementation does not complete its execution in the allotted time, which is specified in the `setPending()` call, the engine *times out* the advice implementation by calling its `cancelled()` method. An advice implementation that does not communicate with external systems, does not perform input/output operations and tasks that may take some time to execute can be implemented as synchronous advice implementation.

Working of Asynchronous Advices in BW Engine

Execution Model (Successful Execution)

Successful execution - `execute()` and `postExecute()`

- The engine calls `execute()` to start the execution.
- The advice gets a thread from a pool and runs its business logic on it.
- The advice sends the signals back to the BW engine, when it finishes executing.
- The engine calls `postExecute()` to finish executing the implementation.

The BW engine's job thread is not blocked until the advice finishes executing and produces its result.

- Parallel tracks, if available in the process, are executed on the jobs thread.
- The engine does not execute the next advice in the pipeline.

Execution Model (timeout)

Timed out execution: `execute()` and `cancelled()`

- The engine calls `execute()` to start the execution.
- The advice gets a thread from a pool and runs its business logic on it.
- Before the advice returns from `execute()`, it sets a timeout.
- When a timeout occurs, the engine calls `cancelled()` to finish executing the implementation.
- The advice releases any outstanding resources.

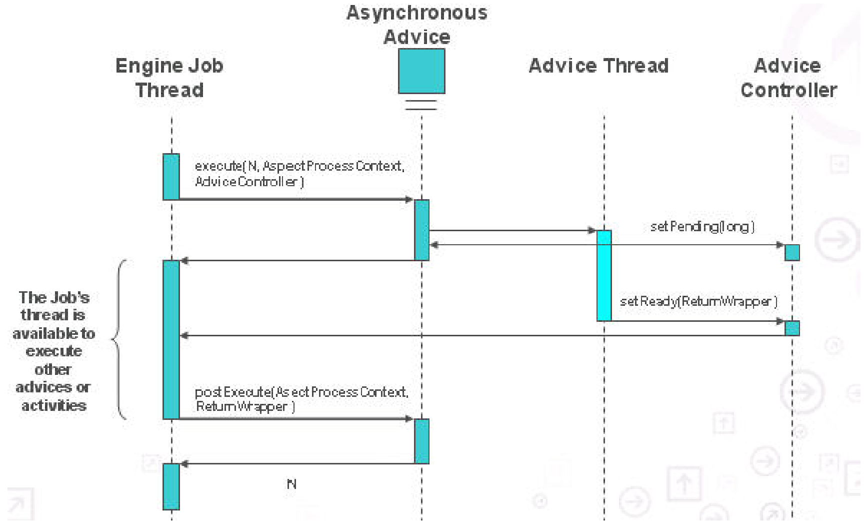
AspectProcessContext cannot be used by asynchronous advices on the parallel thread

- All the logic that requires access to this object should be moved to `execute()`, `postExecute()` or `cancelled()`.

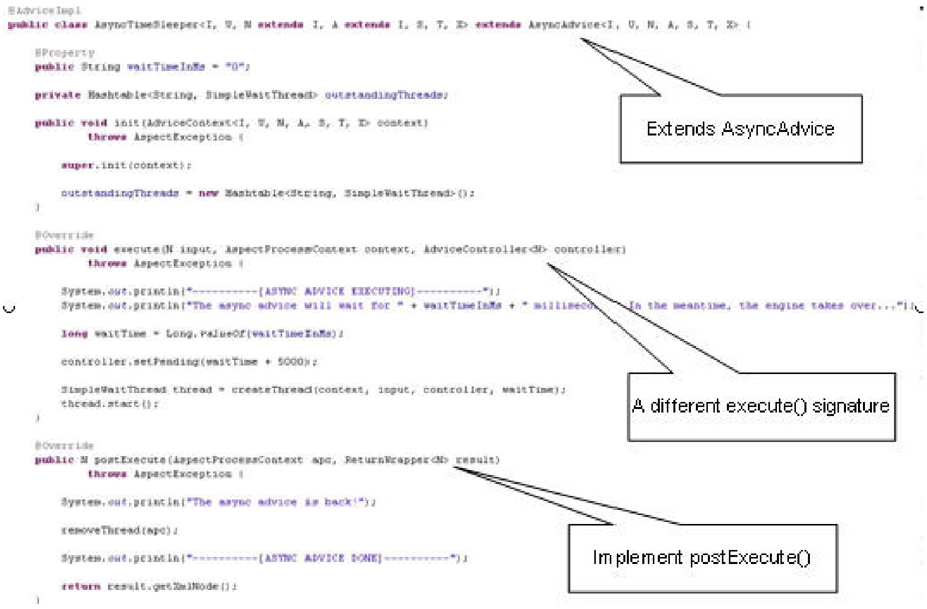
Threading Model: Asynchronous Advice Implementations

Figure 15 shows the threading model.

Figure 15 Threading Model


Engine Job Thread
Asynchronous Advice
Advice Thread
Advice Controller
execute(N, AspectProcessContext, AdviceController)
setPending(long)
setReady(ReturnWrapper)
The Job's thread is available to execute other advices or activities
postExecute(AspectProcessContext, ReturnWrapper)
N

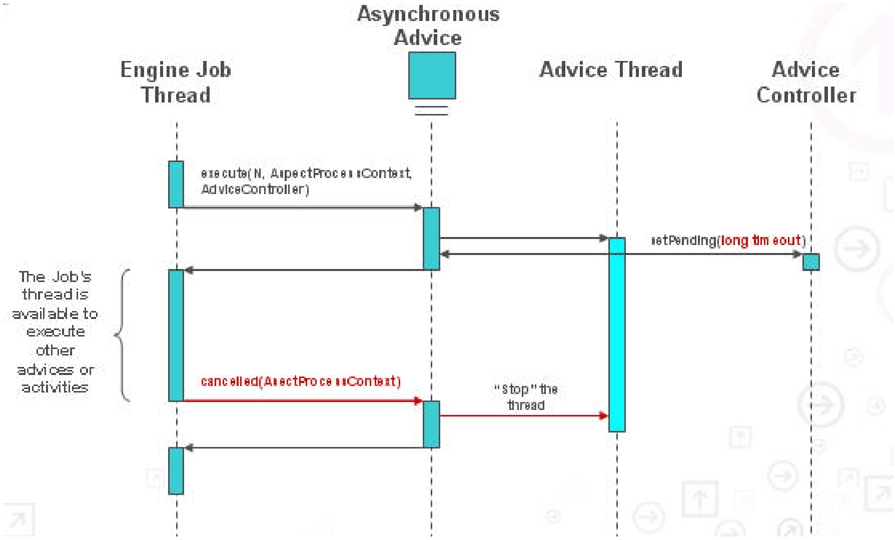
Asynchronous Advice Example



```
@AdviceImpl
public class AsyncTimeSleeper<I, U, N extends I, A extends I, S, T, X> extends AsyncAdvice<I, U, N, A, S, T, X> {

    @Property
    public String waitTimeInMs = "0";

    private Hashtable<String, SimpleWaitThread> outstandingThreads;

    public void init(AdviceContext<I, U, N, A, S, T, X> context)
            throws AspectException {

        super.init(context);

        outstandingThreads = new Hashtable<String, SimpleWaitThread>();
    }

    @Override
    public void execute(N input, AspectProcessContext context, AdviceController<N> controller)
            throws AspectException {

        System.out.println("-----------[ASYNC ADVICE EXECUTING]-----------");
        System.out.println("The async advice will wait for " + waitTimeInMs + " milliseco    In the meantime, the engine takes over...");

        long waitTime = Long.valueOf(waitTimeInMs);

        controller.setPending(waitTime + 5000);

        SimpleWaitThread thread = createThread(context, input, controller, waitTime);
        thread.start();
    }

    @Override
    public N postExecute(AspectProcessContext apc, ReturnWrapper<N> result)
            throws AspectException {

        System.out.println("The async advice is back!");

        removeThread(apc);

        System.out.println("-----------[ASYNC ADVICE DONE]-----------");

        return result.getXmlNode();
    }
```

Threading Model: Asynchronous Advice Implementations (Timeout)

Figure 16 Threading Model (Timeout)


Asynchronous Advice
Engine Job Thread
Advice Thread
Advice Controller
execute(N, AspectProcessContext, AdviceController)
setPending(long timeout)
The Job's thread is available to execute other advices or activities
cancelled(AspectProcessContext)
"Stop" the thread

Summary

- An advice that takes a long time to execute should probably be asynchronous.
- An advice that uses input/output operations should probably be asynchronous.
- When implementing the `cancelled()` method, perform a graceful stop of the advice thread.
- An Asynchronous Advice should always use a thread pool.
- An Asynchronous Advice should always use a timeout and must be configured as property.
- Do not use `AspectProcessContext` on the advice (parallel) thread.

Chapter 4 Hibernate Resume

This chapter describes the Hibernate Resume feature of TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in.

Topics

Features

The main features of hibernate resume are:

- An advice implementation can '**Hibernate**' a process in its execute method.
- A process is said to be hibernated when the process state is written out to the disk (or database) and the process is completely removed from memory.
- Hibernation internally does use the logic similar to checkpointing for TIBCO ActiveMatrix BusinessWorks.
- A hibernated process can be started at a later stage in time using the `ResumeHibernatedProcess` TIBCO Hawk command exposed by TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in on the BW engine.

Comparing Checkpointing and Hibernate

Table 5 shows the comparison between checkpointing (TIBCO ActiveMatrix BusinessWorks) and Hibernate (TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in)

Table 5 Comparing Checkpointing and Hibernate

Checkpointing	Hibernate
Stores Process State.	Stores Process State.
Can only be used in the form of Checkpoint Activity or implicit part of Transaction.	Any Advice Implementation can hibernate at any activity.
Job continues till completion.	Job stops executing and is removed from the memory.
User cannot modify the checkpointed data.	At the time when job is resumed, the advice implementation that hibernated the job, can modify the XML document that is available in the join point.
Only crashed or error jobs are restarted.	User manually decides which jobs to resume.
A restarted job will continue from the next activity.	A resumed job will continue from the same activity (or next activity depending on which join point the job was hibernated).

Defining a Hibernate Advice Implementation

To define a hibernate advice implementation, you must:

- Advertise in the following annotation that it is using the hibernate feature.

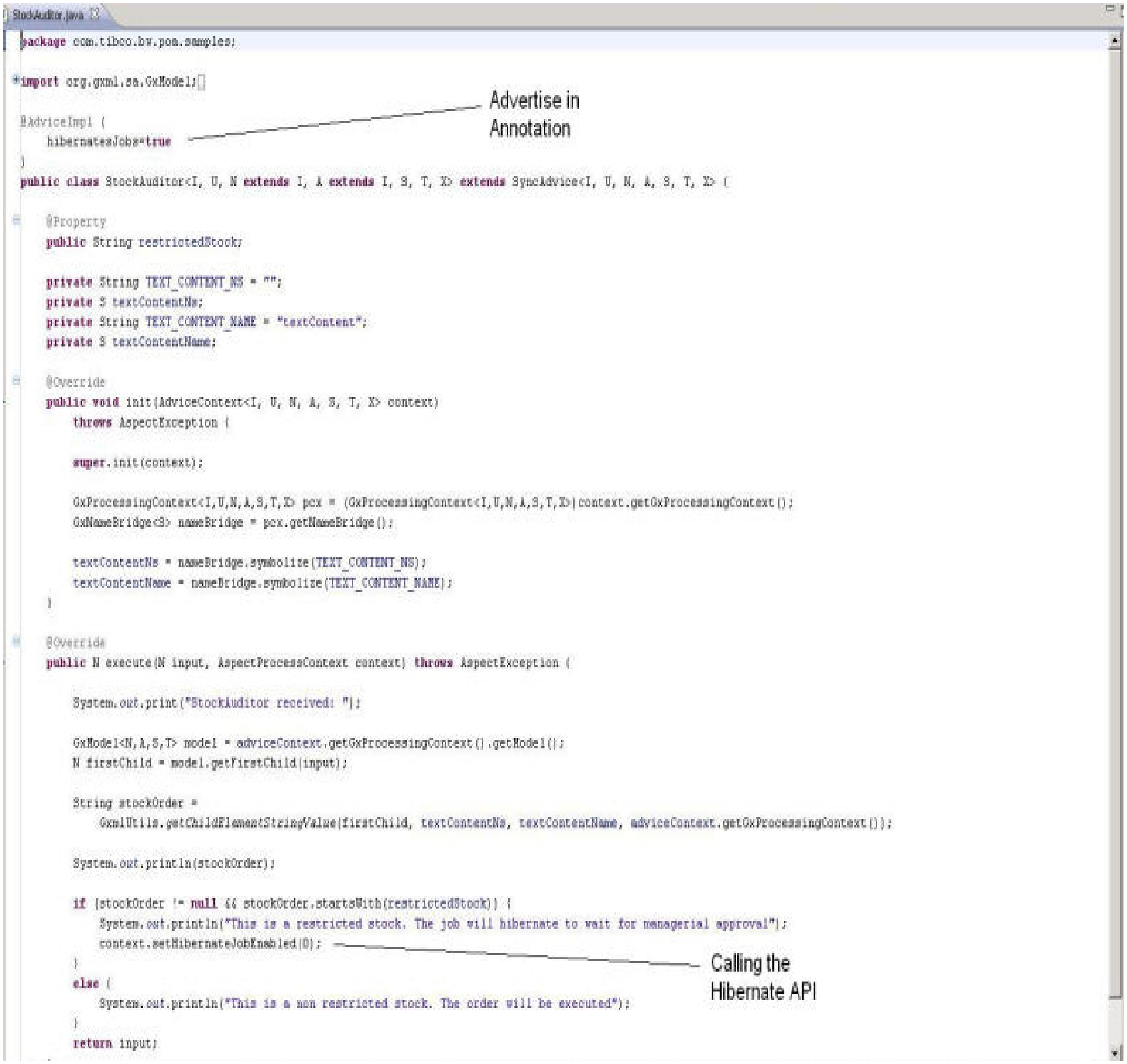
  ```
  @AdviceImpl (
  hibernatesJobs=true
  )
  ```

- Should call the `setHibernateJobEnabled(<timeDelay>)` on the `AspectProcessContext` object.

timeDelay

This `<timeDelay>` attribute enables the user to set a time after which the job will be hibernated. This facility is provided for asynchronous activities on the parallel paths to complete execution. A time delay set to zero (0) will hibernate the job immediately without waiting.

Example of Hibernate Advice Implementation

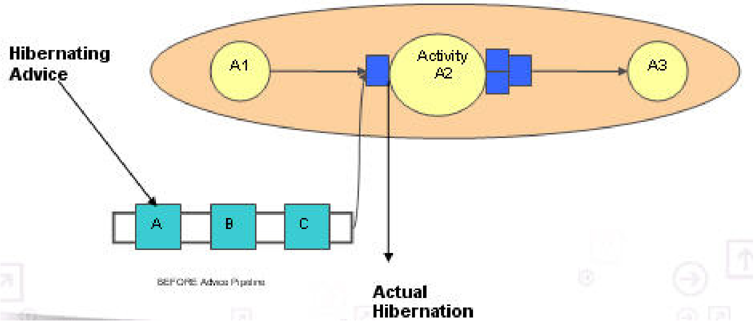


```
package com.tibco.bw.poa.samples;

import org.gxml.sa.GxModel;

@AdviceImpl (
    hibernatesJobs=true
)
public class StockAuditor<I, U, N extends I, A extends I, S, T, X> extends SyncAdvice<I, U, N, A, S, T, X> {

    @Property
    public String restrictedStock;

    private String TEXT_CONTENT_NS = "";
    private S textContentNs;
    private String TEXT_CONTENT_NAME = "textContent";
    private S textContentName;

    @Override
    public void init(AdviceContext<I, U, N, A, S, T, X> context)
        throws AspectException {

        super.init(context);

        GxProcessingContext<I,U,N,A,S,T,X> pcx = (GxProcessingContext<I,U,N,A,S,T,X>)context.getGxProcessingContext();
        GxNameBridge<S> nameBridge = pcx.getNameBridge();

        textContentNs = nameBridge.symbolize(TEXT_CONTENT_NS);
        textContentName = nameBridge.symbolize(TEXT_CONTENT_NAME);
    }

    @Override
    public N execute(N input, AspectProcessContext context) throws AspectException {

        System.out.print("StockAuditor received: ");

        GxModel<N,A,S,T> model = adviceContext.getGxProcessingContext().getModel();
        N firstChild = model.getFirstChild(input);

        String stockOrder =
            GxmlUtils.getChildElementStringValue(firstChild, textContentNs, textContentName, adviceContext.getGxProcessingContext());

        System.out.println(stockOrder);

        if (stockOrder != null && stockOrder.startsWith(restrictedStock)) {
            System.out.println("This is a restricted stock. The job will hibernate to wait for managerial approval");
            context.setHibernateJobEnabled(0);
        }
        else {
            System.out.println("This is a non restricted stock. The order will be executed");
        }
        return input;
```

Execution of a Hibernate Advice

The job may not get hibernated right after an advice calls `setHibernateJobEnabled(long)`. Hence, the engine must finish executing all advices that are part of the same join point before initiating the hibernate procedure. Even after that, the job may still not immediately get hibernated right. The advice that triggers the hibernation may choose to delay it, in order to give chance to any asynchronous advice or any asynchronous activity that might exist in the job on parallel tracks to finish executing. To accomplish that, the advice implementation must call `setHibernatedJobEnabled(long)` method by passing a `timeDelay` greater than 0. A 0 `timeDelay` ensures that the hibernation is initiated right after the engine finishes executing all advices that are running in that particular join point.

Figure 17 Hibernate Advice Execution



Resuming the Hibernated Job

TIBCO Hawk methods have been exposed for listing and resuming the hibernated jobs. A job is resumed from the next point of execution. For example, for a "Before" hibernated advice, the job will resume from activity execution.

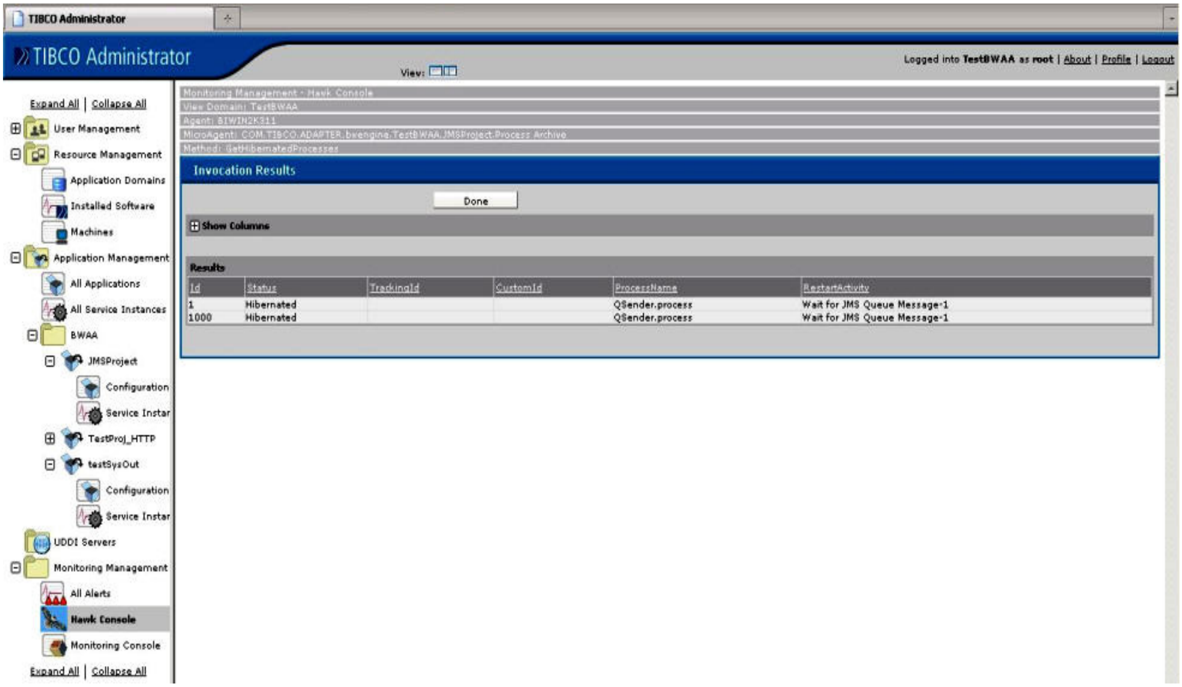
The new 'Resume' activity also allows you to resume a hibernated job. For details refer, Chapter 6, BWAA Palette.

The TIBCO Hawk methods are:

- `GetHibernatedProcesses()` - returns a Tabular Data with all hibernate jobs information
- `ResumeHibernatedProcesses(long jobId)` – resume

A job hibernated on one engine can be resumed on a different instance of the engine. However, the engine must be running prior to calling the `ResumeHibernatedProcess()`.

Figure 18 GetHibernatedProcesses Dialog



Monitoring Management - Hawk Console
View Domain: TestBWAA
Agent: BIWIN2K311
MicroAgent: COM.TIBCO.ADAPTER.bwengine.TestBWAA.JMSProject.Process Archive
Method: GetHibernatedProcesses

Invocation Results

Results

Id	Status	TrackingId	CustomId	ProcessName	RestartActivity
1	Hibernated			QSender.process	Wait for JMS Queue Message-1
1000	Hibernated			QSender.process	Wait for JMS Queue Message-1

Example of Resuming a Job

Figure 19 shows an example of resuming a job from TIBCO Administrator Hawk console.

Figure 19 Resuming a Job

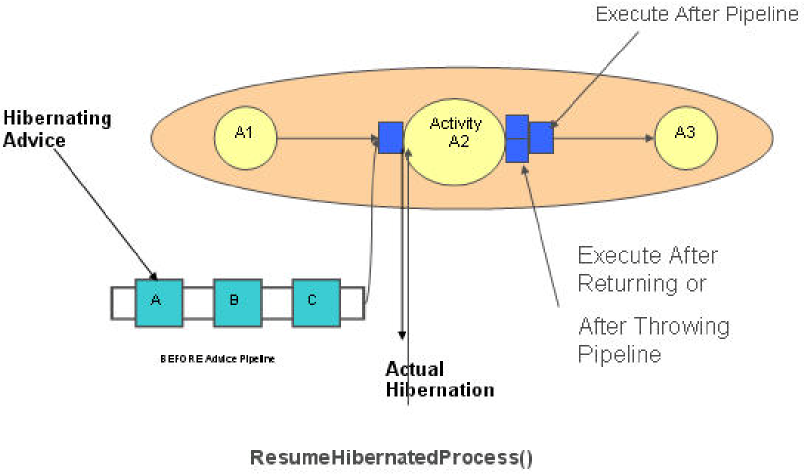
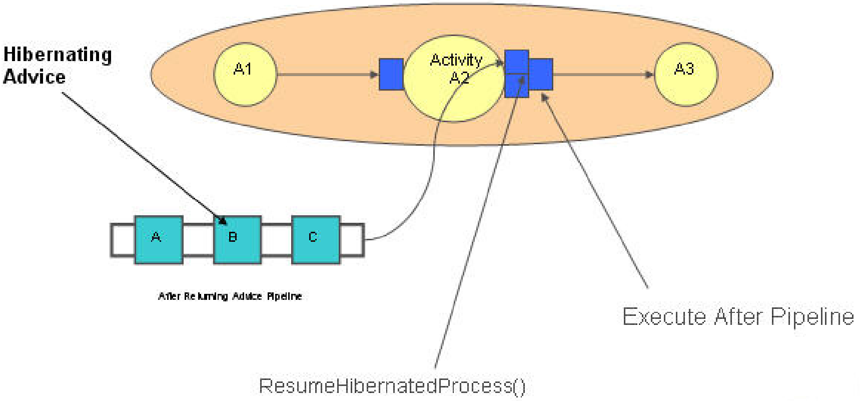


Figure 20 Resuming a Job (After Returning)



Using a Database for Hibernation

A database can be specified just like in checkpointing. A JDBC Shared Resource needs to be in the project which has to hibernate jobs to a database.

Set the following BW Engine Properties:

- `Engine.Hibernate.UseDatabase` – Set to true to use a database
- `Engine.Hibernate.Database.Configuration` – Path to JDBC shared resource

Table Names can be specifed using similar properties like Checkpointing. All databases supported for TIBCO ActiveMatrix BusinessWorks Checkpointing can be used for Hibernation.

If the database is not set, then the job state is stored in a file under the `working/<hostname or enginename>/hibernate` folder.

Modifying the Hibernated Data

The hibernating advice implementation can modify the hibernated data when the job is resumed.

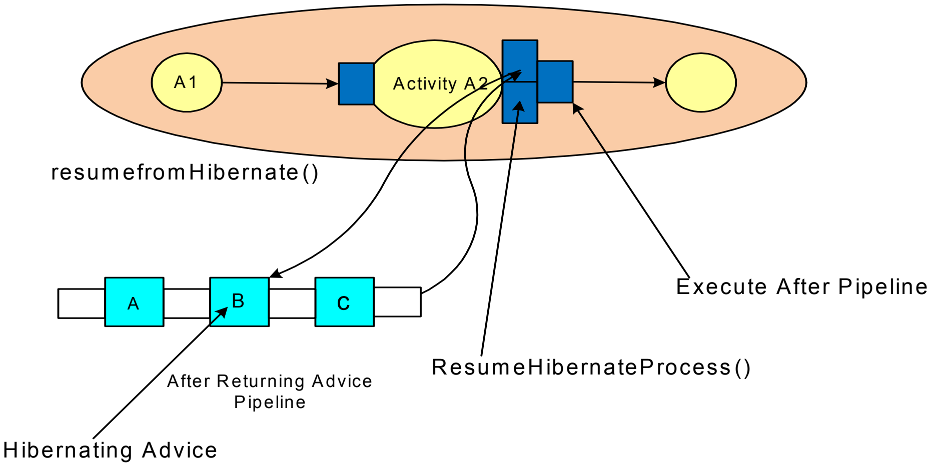
The advice implementation needs to override the following method:

```
public void resumeFromHibernate(N input, AspectProcessContext
context)
```

It should specify appropriate data-Access if it has to mutate the hibernated document. The engine calls this method only on the advice that has triggered the hibernation.

Figure 21 shows the User level view for resumefromHibernate()

Figure 21 User Level View for resumefromHibernate()



Chapter 5 Object Sharing Between Java Activities and Advice Implementation

This chapter provides information about the new API's which will be exposed to enable the user to use this Object Sharing feature.

Topics

Overview

TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in provides a feature for the activities of the JAVA palette in TIBCO ActiveMatrix BusinessWorks and Advice Implementations to pass objects between each other by means of defining a "Java Object Reference" type for Input and/or Output schema of the activities involved.

The feature enables you to pass JAVA objects between:

- Advice Implementations and Java Activities
- An enhanced support for Advice Implementations with another Advice Implementation
- JAVA Activities with other JAVA Activities in a BW project

An Advice Implementation or a JAVA Activity can check-in and check-out objects from the engine data structure by using API's. You have to pass a unique key which will act as a unique identifier for the object in the engine data structure.

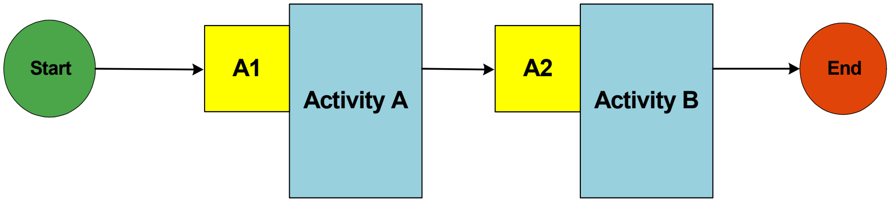
You must pass the key between advice implementations and activities manually. It is not a part of this feature. You must also ensure that the activity/advice implementation which needs to check-out the object, already has the key to it.

Also, the features provided by TIBCO ActiveMatrix BusinessWorks or TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in may or may not be used to accomplish this.

User Scenarios

This section summarises basic supported and unsupported user scenarios. Table 6 and Table 7 shows the user scenarios for activities and advices with implementations.

Figure 22 Basic User Scenario



Following is the basic user scenario as shown in Figure 22.

- **Activity A** and **Activity B** are supported **Java Activities**
- **A1** and **A2** are **Advices (with Implementations)** applied on **Activity A** and **Activity B** respectively
- **Activity A** and **Advice A1** will check-in JAVA objects.
- **Activity B** and **Advice A2** can check-in and check-out JAVA objects.

Table 6 User Scenarios for Activities

Action 1	Key, Value	Action 2	Key, Value	Result
Supported				
Activity A Check-in	foo, obj1	Activity B Check-out	foo	obj1 returned by Activity B.
Activity A Check-in	foo, obj1	Advice A2 Check-out	foo	obj1 returned by Advice A2.
Activity A Check-in	foo, obj1	Activity B Check-in	foo, obj2	obj1 is replaced by obj2 in the map.
Unsupported				
Activity A Check-in	foo, obj1	Advice A2 Check-out	foo, obj2	

Table 7 shows the supported and unsupported scenarios for Advices with implementations.

Table 7 User Scenarios for Advices with Implementations

Action 1	Key, Value	Action 2	Key, Value	Result
Supported				
Advice A1 Check-in	foo, obj1	Activity B Checkout	foo	obj1 returned by Activity B.
Advice A1 Check-in	foo, obj1	Advice A2 Checkout	foo	obj1 returned by Advice A2.
Advice A1 Check-in	foo, obj1	Advice A2 Check-in	foo, obj2	obj1 is replaced by obj2 in the map.
Unsupported				
Advice A1 Check-in	foo, obj1	Activity B Check-in	foo, obj2	

Summary

- Any advice implementation or JAVA activity can check-in objects with a unique-key.
- Any other advice implementation or supported JAVA activity with the knowledge of this unique key can check-out that object.

Limitations

- **When an object is checked-in with a unique-key and the same unique-key is used to check-in a different object**: It is only supported provided the original entity that checked-in the object is of the same type. For example, an advice implementation can overwrite an object checked-in by another advice implementation but NOT by an activity.
- An activity can overwrite an object checked-in by another activity but NOT an advice implementation.

API's and New Interfaces

The following interface has been added in the TIBCO ActiveMatrix BusinessWorks:

public interface `JavaProcessContext`

The public methods available in this interface are:

- **public void** `storeProcessObject` (String key, Serializable obj)
- **public** Serializable `getProcessObject` (String key)
- **public** Serializable `removeProcessObject` (String key)

Implement a `setJavaProcessContext` (JavaProcessContext object) in the Java class in which you wish to use the feature. The engine will invoke this method before any other methods are invoked on the JAVA class.

Advice Implementation developers need not use this interface for storing and retrieving objects. They should continue to use similar methods defined on the `AspectProcessContext` interface.

Use Cases

There are two cases that happen while retrieving:

- If an advice is trying to retrieve an object with the unique-key *foo*, the engine first looks up if an advice which has been executed for the same process earlier has checked-in an object with key *foo*. If it does not find an object, the engine then looks up if an activity has checked-in an object with key *foo*. If the engine does not find an object in both cases, it returns a NULL.
- Similarly, if an activity tries to retrieve an object with the unique-key foo, the engine first looks up if an activity which has been executed in the same process earlier has checked-in an object with key foo. If it does not find an object, the engine then looks up if an advice has checked-in an object with key *foo*. If the engine does not find an object in both cases, it returns a NULL.

Chapter 6 BWAA Palette

BWAA palette contains the activity to resume a previously hibernated job.

Topics

Resume

Activity

This activity fetches the hibernated job from the filesystem or database, loads it into the memory and continues to execute it from the point of hibernation.

Configuration

The Configuration tab has the following fields.

Field	Global Var?	Description
Name	No	The name to appear as the label for the activity in the process definition.
Description	No	Short description of the activity.

Input

The input for the activity is the following.

Input Item	Datatype	Description
jobID	Long	This is the job id to resume a previously hibernated job. **Note**: There is no static configuration for this activity as jobID is the only parameter required.

The activity will throw an `ActivityException` (`JobNotFoundException`), if there is no job with input job id to be resumed.

Output

There is no output for this activity.

However, Output tab displays the stack trace of exception detail whenever Resume activity throws an exception.

Chapter 7 Monitoring and Management

This chapter describes the TIBCO ActiveMatrix BusinessWorks ActiveAspects Plug-in monitoring and management features.

Topics

Introduction

The Monitoring and Management feature is provided by a monitoring interface, which is implemented by a TIBCO Hawk microagent. This section describes the monitoring interface methods. These methods can be called via

- TIBCO Hawk AMI protocol and
- JMX

The TIBCO Hawk Display, the TIBCO ActiveMatrix BusinessWorks monitor servlet, and any other TIBCO Hawk application can use the AMI protocol to invoke the monitoring interface methods on a running engine.

It is important to note that:

- Wild card characters are not supported
- For each application the Advice Instance ID starts with zero (0).
- If there are no asynchronous advices running, the total elapsed and total execution time will be same.
- In order to provide **no value for the Long type**, **specify** "**-1**" as the value. For example, -1 as an input for AdviceInstanceID returns all the available AdviceInstanceIDs data.

getAdviceInstances

Microagent Method

Description This method provides general information about all advice instances that are created in the BW engine.

Index Names `ProcDefName, ActivityName, AdviceInstanceID`

In Index

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValue Choices
`AdviceInstanceID`	Long	null	true	The identifier of the advice instance.	None	None
`ProcDefName`	String	null	true	The name of the process definition.	None	None
`ActivityName`	String	null	true	The name of the activity adjacent to the advice.	None	None
`ActivityClass`	String	null	true	The name of the activity class.	None	None

`Where`	String	null	true	The location where the advice instance is running (for example, before activity, after activity, after returning activity, after throwing activity)	None	`ACT_BEFORE`
						`ACT_AFTER_RET,`
						`ACT_AFTER_THR,`
						`ACT_AFTER`

`AdviceNs`	String	null	true	The namespace of the advice.	None	None
`AdviceName`	String	null	true	The name of the advice.	None	None
`AdviceImpl`	String	null	true	The advice implementation name.	None	None

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValue Choices
`Sync`	Boolean	null	true	Whether the advice implementation is synchronous.	None	True, False
`DataAccess`	String	null	true	The data access mode (for example, read-only, read-write)	None	READ-ONLY, READ-WRITE
`HibernatesJobs`	Boolean	null	true	Whether the advice instance is capable of hibernating jobs.	None	True, False

Out Index

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValueChoices
`AdviceInstanceID`	Long	null	true	The identifier of the advice instance.	None	None
`ProcDefName`	String	null	true	The name of the process definition.	None	None
`ActivityName`	String	null	true	The name of the activity adjacent to the advice.	None	None
`ActivityClass`	String	null	true	The name of the activity class.	None	None
`Where`	String	null	true	The location where the advice instance is running (For example, before activity, after activity, after returning activity, after throwing activity)	None	ACT_BEFORE ACT_AFTER_RET, ACT_AFTER_THR, ACT_AFTER

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValueChoices
`AdviceNs`	String	null	true	The namespace of the advice.	None	None
`AdviceName`	String	null	true	The name of the advice.	None	None
`AdviceImpl`	String	null	true	The advice implementation name.	None	None
`Sync`	Boolean	null	true	Whether the advice implementation is synchronous.	None	True, False
`DataAccess`	String	null	true	The data access mode (for example, read-only, read-write)	None	READ-ONLY, READ-WRITE
`HibernatesJobs`	Boolean	null	true	Whether the advice instance is capable of hibernating jobs.	None	True, False

Output Example

Name	Output
`AdviceInstanceID`	• 10000 • 10001
`ProcDefName`	• Folder1/P1.process • P2.process
`ActivityName`	• ReadActivity • HttpSend
`ActivityClass`	• bw.FileRead • bw.HTTPSendReceive
`Where`	• ACT_BEFORE • ACT_AFTER_RET

Name	Output
`AdviceNamespace`	• http://aspects.com/audit • http://aspects.com/log
`AdviceName`	• Audit • Log
`AdviceImpl`	• com.example.AuditAdvice • com.example.LogAdvice
`Sync`	• True • False
`DataAccess`	• READ-WRITE • READ-ONLY
`HibernatesJobs`	• False • False

getAdviceInstanceMetrics

Microagent Method

Description This method provides various metrics for a particular advice instance.

Index Names `AdviceInstanceID`

In Index

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValue Choices
`AdviceInstanceID`	Long	null	true	The identifier of the advice instance.	None	None

Out Index

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValue Choices
`AdviceInstanceID`	Long	null	true	The identifier of the advice instance.	None	None
`ExecutionCount`	Long	null	true	The number of times this advice instance has been executed by the engine.	None	None
`ErrorCount`	Long	null	true	The number of times this advice instance has thrown an exception.	None	None
`RunningCount`	Long	null	true	The number of times this particular advice instance is currently running.	None	None

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValue Choices
`TotEla`	Long	null	true	Total wall-clock time used by all calls of this advice instance (milliseconds) includeing the waiting time for asynchronous advices.	None	`None`
`TotExe`	Long	null	true	Total wall-clock time used by all calls of this advice instance (milliseconds) not including the waiting time for asynchronous advices.	None	None
`AvgEla`	Long	null	true	Average elapsed time of all completed advice instances (milliseconds).	None	None
`MinEla`	Long	null	true	Minimum elapsed time of all completed advice instances (milliseconds).	None	None
`MaxEla`	Long	null	true	Maximum elapsed time of all completed advice instances (milliseconds).	None	None
`LastEla`	Long	null	true	The elapsed time of the last completed advice instance (milliseconds).	None	None

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValue Choices
`AvgExe`	Long	null	true	Average execution time of all completed advice instances (milliseconds).	None	None
`MinExe`	Long	null	true	Minimum execution time of all completed advice instances (milliseconds).	None	None
`MaxExe`	Long	null	true	Maximum execution time of all completed advice instances (milliseconds).	None	None
`LastExe`	Long	null	true	The execution time of the last completed advice instance (milliseconds).	None	None

Output Example

Name	Output
`AdviceInstanceID`	• 10000 • 10001
`ExecutionCount`	• 3 • 1
`ErrorCount`	• 3 • 0
`RunningCount`	• 2 • 4
`TotEla`	• 150090 • 104023

Name	Output
`TotExe`	- 150090 - 13311
`AvgEla`	- 50030 - 104023
`MinEla`	- 50007 - 104023
`MaxEla`	- 50075 - 104023
`LastEla`	- 50008 - 50008
`AvgExe`	- 50030 - 100011
`MinExe`	- 50007 - 100011
`MaxExe`	- 50075 - 100011
`LastExe`	- 50008 - 50008

getRunningAdviceInstancesCount

Microagent Method

Description This method provides the number of synchronous and asynchronous advice instances that are executing as part of the jobs that are currently running.

Note: This method requires no input.

Out Index

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValue Choices
`SyncCount`	Long	null	true	The total number of synchronous advice instances that are executing in the jobs that are currently running	None	None
`AsyncCount`	Long	null	true	The total number of asynchronous advice instances that are executing in the jobs that are currently running.	None	None

Example Output

Name	Output
`SyncCount`	• 2
`AsyncCount`	• 15

getRunningAdviceInstances

Microagent Method

Description This method provides metrics for the advice instances that are executing as part of the jobs that are currently running.

Index Name `AdviceInstanceID`

In Index

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValue Choices
`MinDuration`	Long	null	true	Minimum elapsed wall-clock time since the advice instance started (milliseconds).	None	None
`JobID`	Long	null	true	The Job identifier.	None	None
`MaxReturnCount`	Long	null	true	The maximum number of advice instances to be returned. **Note**: It is important when there are multiple advice instances running.	None	None

Out Index

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValue Choices
`AdviceInstanceID`	Long	null	true	The identifier of the advice instance.	None	None
`ProcDefName`	String	null	true	The name of the process definition.	None	None
`JobID`	Long	null	true	The Job identifier.	None	None
`ActivityName`	String	null	true	The name of the activity adjacent to the advice.	None	None

Name	Type	Default	IsOpen	Description	ValueChoices	LegalValue Choices
`AdviceNamespace`	String	null	true	The namespace of the advice.	None	None
`AdviceName`	String	null	true	The name of the advice.	None	None
`AdviceImpl`	String	null	true	The advice implementation name.	None	None
`Sync`	Boolean	null	true	Whether the advice implementation is synchronous.	None	None
`StartTime`	Long	null	true	Time at which the advice instance started executing in the current job (milliseconds).	None	None
`Duration`	Long	null	true	The elapsed wall-clock time since the advice instance started (milliseconds) executing in the current job.	None	None

Output Example

Name	Output
`AdviceInstanceID`	• 0 • 1
`ProcDefName`	• ProcessSynch.process • ProcessAsynch.process
`JobID`	• 34 • 36
`ActivityName`	• SynchLog • AsynchLog

Name	Output
`AdviceNs`	- http://examples.org/One - http://examples.org/One
`AdviceName`	- Logger1 - Logger2
`AdviceImpl`	- com.tibco.test.bwaa.SleepLogger - com.tibco.test.bwaa.AsynchTimeSleeper
`Sync`	- true - false
`StartTime`	- 1303661907012 - 1303661907011
`Duration`	- 43830 - 13831

Appendix A ActivityTypes

Topics

ActivityTypes

The tables contains the activity types of all TIBCO ActiveMatrix BusinessWorks activities. Refer to this table in order to generate expressions in Point cut query language.

Table 8 Active Enterprise Palette Activity Types

Activity	Activity Description	Activity Type
Active Enterprise Palette		
event-source	Adapter Request-Response Server	`bw.aeRRServer`
event-source	Adapter Subscriber	`bw.aeSubstriction`
signal-in	Wait for Adapter Message	`bw.aeSubSignalInActivity`
signal-in	Wait for Adapter Request	`bw.aeServerSignalInActivity`
activity	Invoke an Adapter Request-Response Service	`bw.aeOpClientReqActivity`
activity	Respond to Adapter Request	`bw.aeOpServerReplyActivity`
activity	Publish to Adapter	`bw.aePubActivity`
activity	Send Exception to Adapter Request	`bw.aeOpServerFaultActivity`

Table 9 Activity Types of Other ActiveMatrix BusinessWorks Palettes

Activity Kind	Activity Description	Activity Type
File Palette		

Table 9 Activity Types of Other ActiveMatrix BusinessWorks Palettes

Activity Kind	Activity Description	Activity Type
event-source	File Poller	`bw.FileEventSourceResource`
signal-in	Wait for File Change	`bw.FileSignalInUI`
activity	Copy File	`bw.FileCopyActivity`
activity	Create File	`bw.FileCreateActivity`
activity	List Files	`bw.ListFilesActivity`
activity	Read File	`bw.FileReadActivity`
activity	Remove File	`bw.FileRemoveActivity`
activity	Rename File	`bw.FileRenameActivity`
activity	Write File	`bw.FileWriteActivity`
FTP Palette		
activity	FTP Change Default Directory	`bw.FTPChangeDefaultDirActivityUI`
activity	FTP Delete File	`bw.FTPDeleteFileActivityUI`
activity	FTP Dir	`bw.FTPDirActivityUI`
activity	FTP Get Default Directory	`bw.FTPGetDefaultDirActivityUI`
activity	FTP Get	`bw.FTPGetActivityUI`
activity	FTP Make Remote Directory	`bw.FTPMakeRemoteDirActivityUI`
activity	FTP Put	`bw.FTPPutActivityUI`
activity	FTP Quote	`bw.FTPQuoteActivityUI`
activity	FTP Remove Remote Directory	`bw.FTPRemoveRemoteDirActivityUI`
activity	FTP Rename File	`bw.FTPRenameActivityUI`
activity	FTP SYS Type	`bw.FTPSysTypeActivityUI`
General Palette		

Table 9 Activity Types of Other ActiveMatrix BusinessWorks Palettes

Activity Kind	Activity Description	Activity Type
event-source	On Event Timeout	`bw.onEventTimeout`
event-source	On Notification Timeout	`bw.onNotificationTimeout`
event-source	On Shutdown	`bw.onShutdown`
event-source	On Startup	`bw.onStartup`
event-source	On Error	`bw.onErrorProcess`
event-source	Receive Notification	`bw.waitStarter`
event-source	Timer	`bw.timer`
signal-in	Catch	`bw.catch`
signal-in	Sleep	`bw.sleep`
signal-in	Wait	`bw.waitActivity`
activity	Assign	`bw.assignActivity`
activity	Call Process	`bw.subprocess`
activity	Checkpoint	`bw.checkpoint`
activity	Confirm	`bw.confirm`
activity	Engine Command	`bw.enginecommand`
activity	External Command	`bw.CmdExecActivity`
activity	Generate Error	`bw.throw`
activity	Get Shared Variable	`bw.getSharedVariable`
activity	Inspector	`bw.inspectorActivity`
activity	Label	`bw.label`
activity	Mapper	`bw.MapperActivity`
activity	Notify	`bw.notifyActivity`
activity	Null	`bw.null`

Table 9 Activity Types of Other ActiveMatrix BusinessWorks Palettes

Activity Kind	Activity Description	Activity Type
activity	Rethrow	`bw.rethrow`
activity	Set Shared Variable	`bw.setSharedVariable`
activity	Write To Log	`bw.log`
HTTP Palette		
event-source	HTTP Receiver	`bw.httpEventSource`
signal-in	Wait for HTTP Request	`bw.httpSignalIn`
activity	HTTP Send Request	`bw.httpRequest`
activity	HTTP Send Response	`bw.httpWebResponse`
Java Palette		
activity	Java Code	`bw.javaActivity`
event-source	Java Event Source	`bw.JavaEventSource`
activity	Java Method	`bw.JavaMethodActivity`
activity	Java To XML	`bw.JavaToXmlActivity`
activity	XML To Java	`bw.XmlToJavaActivity`
JDBC Palette		
activity	JDBC Call Procedure	`bw.JDBCCallActivity`
activity	JDBC Get Connection	`bw.JDBCGetConnectionActivity`
activity	JDBC Query	`bw.JDBCQueryActivity`
activity	JDBC Update	`bw.JDBCUpdateActivity`
activity	SQL Direct	`bw.JDBCGeneralActivity`
JMS Palette		
event-source	JMS Queue Receiver	`bw.JMSQueueEventSource`
event-source	JMS Topic Subscriber	`bw.JMSTopicEventSource`

Table 9 Activity Types of Other ActiveMatrix BusinessWorks Palettes

Activity Kind	Activity Description	Activity Type
signal-in	Wait for JMS Queue Message	`bw.JMSQueueSignalInActivity`
signal-in	Wait for JMS Topic Message	`bw.JMSTopicSignalInActivity`
activity	Get JMS Queue Message	`bw.JMSQueueGetMessageActivity`
activity	JMS Queue Requestor	`bw.JMSQueueRequestReplyActivity`
activity	JMS Queue Sender	`bw.JMSQueueSendActivity`
activity	JMS Topic Publisher	`bw.JMSTopicPublishActivity`
activity	JMS Topic Requestor	`bw.JMSTopicRequestReplyActivity`
activity	Reply to JMS Message	`bw.JMSReplyActivity`
Mail Palette		
event-source	Receive Mail	`bw.MailEventSourceResource`
activity	Send Mail	`bw.MailActivityResource`
Parse Palette		
activity	Parse Data	`bw.ParseActivity`
activity	Render Data	`bw.RenderActivity`
RMI Palette		
activity	RMI Lookup	`bw.lookup`
activity	RMI Server	`bw.starter`
RV Palette		
event-source	RV Subscriber	`bw.RVEventSource`
signal-in	Wait for RV Message	`bw.rvSignalInActivity`
activity	Publish RV Message	`bw.RVPubActivity`
activity	Reply to RV Request	`bw.RVReplyActivity`
activity	Send RV Request	`bw.RVRequestActivity`

Table 9 Activity Types of Other ActiveMatrix BusinessWorks Palettes

Activity Kind	Activity Description	Activity Type
Service Palette		
activity	Get Context	`bw.getContext`
activity	Invoke Partner	`bw.invokePartner`
activity	Set Context	`bw.setContext`
SOAP Palette		
event-source	SOAP Event Source	`bw.SOAPEventSourceUI`
activity	Retrieve Resources	`bw.RetrieveResource`
activity	SOAP Request Reply	`bw.SOAPSendReceiveUI`
activity	SOAP Send Fault	`bw.SOAPSendFaultUI`
activity	SOAP Send Reply	`bw.SOAPSendReplyUI`
activity	MIME Parse	`bw.MimeParserActivity`
TCP Palette		
event-source	TCP Receiver	`bw.TCPEventSource`
signal-in	Wait for TCP Request	`bw.TCPSignalIn`
activity	Read TCP Data	`bw.TCPRead`
activity	TCP Close Connection	`bw.TCPCloseConnection`
activity	TCP Open Connection	`bw.TCPOpenConnection`
activity	Write TCP Data	`bw.TCPWrite`
Transaction Palette		
activity	Transaction State	`bw.TransactionStateActivity`

Table 9 Activity Types of Other ActiveMatrix BusinessWorks Palettes

Activity Kind	Activity Description	Activity Type
XML Activities Palette		
activity	Parse XML	`bw.XMLParseActivity`
activity	Render XML	`bw.XMLRendererActivity`
activity	Transform XML	`bw.XMLTransformActivity`
Manual Work Palette		
activity	Assign Work	`bw.AssignWork`
activity	Download Document	`bw.DownloadDocument`
activity	Get Work Status	`bw.GetWorkStatus`
activity	Modify Work	`bw.ModifyWork`
signal-in	Wait For Completion	`bw.WaitForCompletion`

Appendix B Developing gXML Applications

Topics

Overview

A Generic Java API for XQuery Data Model (XDM) and eXtensible Markup Language (XML) Processing. gXML also provides, out-of-the-box, a cohesive suite of XML processing implementations such as XPath, XSLT, XQuery, Serialization, W3C XML Schema and Validation.

gXML is a new way of writing XML code in the Java language. The code that you write to the gXML API can be run against any Data Model that supports the gXML bridge.

This flexibility offers the following benefits:

- minimizes expensive conversion overhead
- greater opportunities for performance optimization
- greater code reuse
- minimize risks associated with locking into one Data Model

gXML currently supports Parsing, Serialization, XDM Data Model, XPath 2, XSLT 2 and XQuery, W3C XML Schema and Validation.

- A gXML bridge is provided for org.w3c.dom.Node.
- A gXML bridge for a high performance proprietary implementation is complete but not yet released.
- A gXML bridge for a reference implementation is complete but not yet released. A gXML bridge for AxiOM is in the works.

Developing gXML Applications

This section illustrates one way of using gXML. All gXML processors, including custom processing, run within a GxProcessingContext instance that provides necessary meta data. A GxProcessingContext instance in turn is created through a GxApplication instance. It is your responsibility to write a class that provides an instance of GxApplication. The best way to do this is to write an abstract class that implements all but the newProcessingContext method of GxApplication. This approach will allow you to write your application generically and then inject the choice of parameterization as late as possible for maximum code reuse and flexibility.

This, of course, is not the only way to use gXML. An existing architecture may force the choice of parameterization and create silos of XML processing. The degree of integration in this case may be less that is possible with a homogeneous solution.

Whatever the approach, the best way to use gXML is to write generic, parameterized, and XML processing code whenever possible.

Implementing GxApplication

```
package org.gxml.book.common;

import java.io.StringWriter;
import java.net.URI;
import java.net.URISyntaxException;

import junit.framework.TestCase;

import org.gxml.sa.GxApplication;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.sa.GxSequenceHandler;
import org.gxml.xdm.Resolver;

import com.tibco.gxml.sa.api.common.util.PreCondition;
import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;

public abstract class SampleApp<I, U, N extends I, A extends I, S, T, X> extends TestCase implements GxApplication<I, U, N, A, S, T, X>
{
    public Resolver getResolver()
    {
        try
        {
            return new SampleResolver(new URI("../../plugins/org.gxml.book/resources/foo.xml"));
        }
        catch (final URISyntaxException e)
        {
            throw new AssertionError(e);
        }
    }

    protected String serialize(final N node, final GxProcessingContext<I, U, N, A, S, T, X> pcx)
    {
        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);

        // Configure for "pretty" printing.
        sf.setIndent(Boolean.TRUE);

        final StringWriter w = new StringWriter();

        final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);

        final GxModel<N, A, S, T> model = pcx.getModel();

        handler.startDocument(null);
        try
        {
            model.stream(node, true, true, handler);
        }
        finally
        {
            handler.endDocument();
        }

        return w.toString();
    }

    /**
     * Some bridge implementations may use {@link String} directly for symbols. They must make them behave according to
     * symbol semantics (==,toString).
     */
    public void assertNodeSymbolSemantics(final N node, final GxProcessingContext<I, U, N, A, S, T, X> pcx)
    {
        final GxModel<N, A, S, T> model = pcx.getModel();
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        switch (model.getNodeKind(node))
        {
            case ELEMENT:
            {
                assertSymbolSemantics(model.getNamespaceURI(node),
nameBridge);
                assertSymbolSemantics(model.getLocalName(node),
nameBridge);
            }
            case TEXT:
            case DOCUMENT:
            {

            }
            break;
            default:
            {
                throw new AssertionError(model.getNodeKind(node));
            }
        }
    }

    public void assertSymbolSemantics(final S symbol, final
GxNameBridge<S> nameBridge)
    {
        PreCondition.assertArgumentNotNull(symbol, "symbol");
        PreCondition.assertArgumentNotNull(nameBridge,
"nameBridge");
       assertSame(symbol, nameBridge.symbolize(symbol.toString()));
        assertSame(symbol,
nameBridge.symbolize(copy(symbol.toString())));
    }

    /**
     * Do anything to manufacture a String that is equal, but not
identical (the same), as the original.
     * <p>
     * This method has the post-condition that the strings are equal
but not the same.
     * </p>
     *
     * @param original
     *            The original.
     * @return A copy of the original string.
     */
    private String copy(final String original)
    {
        final String copy = original.concat("junk").substring(0,
original.length());
```

```
       // Post-conditions verify that this actually works and isn't
"optimized" out.'
        assertEquals(original, copy);
        assertNotSame(original, copy);
        // Be Paranoid
        assertTrue(original.equals(copy));
        assertFalse(original == copy);
        // OK. That'll do.'
        return copy;
    }
}
```

Implementing GxCatalog

A catalog provides the means to isolate your application from the physical location of file resources. Writing a catalog simply means implementing the GxCatalog interface so that it maps form the logical locations specified in code or XML resources to the corresponding physical location.

```
package org.gxml.book.common;

public class SampleCatalog
{

}
```

Implementing GxResolver

A resolver takes a base-uri and an href and uses these two values to return a stream.

```
package org.gxml.book.common;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URL;

import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;

import com.tibco.gxml.sa.api.common.util.PreCondition;

public final class SampleResolver implements Resolver
{
    final URI baseURI;

    public SampleResolver(final URI baseURI)
    {
        this.baseURI = PreCondition.assertArgumentNotNull(baseURI, "baseURI");
    }

    /**
     * Convert a URI relative to a base URI into an input source.
     * <p/>
     * This default implementation requires that neither parameter be null, and performs the expected action to retrieve
     * the input source (which may involve network access).
     *
     * @param baseURI
     *            the base URI against which the target is to be resolved; must not be null
     * @param location
     *            the URI to resolve; must not be null
     * @return a pair of InputStream and resolved URI.
     */
    public Resolved<InputStream> resolveInputStream(final URI location) throws IOException
    {
        PreCondition.assertArgumentNotNull(location, "uri");
        if (location.isAbsolute())
        {
            return retrieve(location, location);
        }
        else
        {
           PreCondition.assertArgumentNotNull(baseURI, "baseURI");

            final URI base = baseURI.normalize();
            final URI resolved = base.resolve(location);

            return retrieve(location, resolved);
        }
    }

   private Resolved<InputStream> retrieve(final URI location, final URI uri) throws IOException
    {
        PreCondition.assertArgumentNotNull(uri, "uri");

        final URL toRetrieve;

        if (!uri.isAbsolute()) // assume local file
        {
            final File canonFile = new File(uri.toString()).getCanonicalFile();
            toRetrieve = canonFile.toURI().toURL();
        }
        else
        {
            toRetrieve = uri.toURL();
        }

        if (toRetrieve == null)
        {
            throw new FileNotFoundException(uri.toString());
        }

        final InputStream stream = toRetrieve.openStream();
        if (stream == null)
        {
           throw new FileNotFoundException(toRetrieve.toString());
        }
        try
        {
            return new Resolved<InputStream>(location, stream,
toRetrieve.toURI());
        }
        catch (final URISyntaxException e)
        {
            throw new AssertionError(e);
        }
    }
}
```

Injecting DOM

The final task in providing a concrete GxApplication class is to implement the newProcessingContext method on a derived class. This is where you get to choose the tree, atomic values, meta data and symbols that your application will use. In many cases you will use an off-the-shelf processing context class, but it is also possible to assemble your own variety or build one entirely from scratch.

If you are going to use gXML with org.w3c.dom.Node, you still have choices for the atomic values that your system will use as well as the meta data implementation. This example uses atomic values that are mostly Java Wrapper types and the reference sequence type implementation, SmSequenceType.

```
package org.gxml.book.parsing;

import org.gxml.sa.GxMetaBridge;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.mutable.GxApplicationMutable;
import org.gxml.sa.mutable.GxProcessingContextMutable;
import org.gxml.xs.SmMetaBridge;
import org.gxml.xs.SmSequenceType;
import org.w3c.dom.Node;

import com.tibco.gxml.sa.api.common.datatype.StringNameBridge;
import com.tibco.gxml.sa.common.atom.AtomBridge;
import com.tibco.gxml.sa.common.helpers.GxMetaBridgeOnSmMetaBridgeAdapter;
import com.tibco.gxml.sa.common.helpers.SmAtomBridgeOnGxAtomBridgeAdapter;
import com.tibco.gxml.sa.xdm.dom.DomProcessingContext;
import com.tibco.gxml.xs.SmMetaBridgeFactory;

/**
 * Demonstration of constructing a concrete GxApplication(Mutable) implementation using the DOM processing context.
 */
public final class DomValidatingParsingSample extends BookValidatingParsingSample<Object, Object, Node, Object, String, SmSequenceType<Object, String>, Object> implements GxApplicationMutable<Object, Object, Node, Object, String, SmSequenceType<Object, String>, Object>
{
    public final GxProcessingContextMutable<Object, Object, Node, Object, String, SmSequenceType<Object, String>, Object> newProcessingContext()
    {
        // The name bridge is created along with the processing context for maximum concurrency.
        final GxNameBridge<String> nameBridge = new StringNameBridge();
        final AtomBridge<String> atomBridge = new AtomBridge<String>(nameBridge);
        final SmMetaBridge<Object, String> cache = new SmMetaBridgeFactory<Object, String>(new SmAtomBridgeOnGxAtomBridgeAdapter<Object, String>(atomBridge)).newMetaBridge();
        final GxMetaBridge<Object, String, SmSequenceType<Object, String>> metaBridge = new GxMetaBridgeOnSmMetaBridgeAdapter<Object, String>(cache, atomBridge);

        final DomProcessingContext<Object, SmSequenceType<Object, String>> pcx = new DomProcessingContext<Object, SmSequenceType<Object, String>>(this, metaBridge, cache);

        // Set the "owning" processing context on the atom bridge.
        atomBridge.setProcessingContext(pcx);

        // Return the newly constructed processing context.
        return pcx;
    }
}
```

gXML Recipes

Parsing

Parsing a Character Stream and a Byte Stream

```
package org.gxml.book.parsing;

import java.io.InputStream;
import java.io.Reader;
import java.io.StringReader;
import java.net.URI;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.xdm.NodeKind;
import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;

import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;

public abstract class BookIntroParsingSample<I, U, N extends I, A extends I, S, T, X> extends SampleApp<I, U, N, A, S, T, X>
{
    public void testCharacterStreamParse() throws Exception
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final GxDocumentBuilderFactory<N, S> factory = new DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

        final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();

        final String xmlString = "<e>123</e>";
        final URI systemId = new URI("e.xml");
        final Reader characterStream = new StringReader(xmlString);
        final N doc = builder.parse(characterStream, systemId);

        final GxModel<N, A, S, T> model = pcx.getModel();

        assertEquals(NodeKind.DOCUMENT, model.getNodeKind(doc));

        final N e = model.getFirstChildElement(doc);
        assertEquals(NodeKind.ELEMENT, model.getNodeKind(e));
        assertEquals("e", model.getLocalNameAsString(e));
        assertEquals("123", model.getStringValue(e));
    }

    public void testByteStreamParse() throws Exception
    {
        final Resolver resolver = getResolver();

        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();

        final URI systemId = new URI("email.xml");
        final Resolved<InputStream> source =
resolver.resolveInputStream(systemId);

        final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

        final GxDocumentBuilder<N> builder =
factory.newDocumentBuilder();

        final N document = builder.parse(source.getResource(),
source.getSystemId());

        final GxModel<N, A, S, T> model = pcx.getModel();

        assertEquals(NodeKind.DOCUMENT,
model.getNodeKind(document));

        final N email = model.getFirstChildElement(document);
        assertEquals(NodeKind.ELEMENT, model.getNodeKind(email));
        assertEquals("email", model.getLocalNameAsString(email));
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
        final S namespaceURI =
nameBridge.symbolize("http://www.example.com");
        final S localName = nameBridge.symbolize("from");
        final N from = model.getFirstChildElementByName(email,
namespaceURI, localName);
        assertEquals("Julie", model.getStringValue(from));

       for (final N node : model.getDescendantOrSelfAxis(document))
        {
            assertNodeSymbolSemantics(node, pcx);
        }
    }
}
```

Constructing a Data Model Tree Programmatically

This example demonstrates constructing a tree directly using the fragment builder.

```
package org.gxml.book.snoopy;

import java.io.IOException;
import java.io.InputStream;
import java.io.StringReader;
import java.io.StringWriter;
import java.net.URI;
import java.net.URISyntaxException;

import javax.xml.namespace.QName;
import javax.xml.parsers.ParserConfigurationException;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxException;
import org.gxml.sa.GxFragmentBuilder;
import org.gxml.sa.GxMetaBridge;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.sa.GxSequenceHandler;
import org.gxml.sa.GxVariantBridge;
import org.gxml.xdm.NodeKind;
import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;
import org.gxml.xs.SmName;

import com.tibco.gxml.sa.api.common.lang.ExprException;
import com.tibco.gxml.sa.api.common.lang.ExprResult;
import com.tibco.gxml.sa.api.common.lang.GxExpr;
import com.tibco.gxml.sa.api.common.lang.GxExprContextDynamicArgs;
import com.tibco.gxml.sa.api.common.lang.GxExprContextStaticArgs;
import com.tibco.gxml.sa.api.common.lang.GxLanguageToolKit;
import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;
import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
import com.tibco.gxml.sa.processor.xquery.LanguageToolKit;
import com.tibco.gxml.sa.processor.xslt.GxTransform;
import com.tibco.gxml.sa.processor.xslt.GxTransformBuilder;
import com.tibco.gxml.sa.processor.xslt.GxTransformer;
import com.tibco.gxml.sa.processor.xslt.XSLTransformBuilder;
```

```
import
com.tibco.gxmlsa.processor.org.exslt.strings.ExsltStringsFunctionGroup;

public abstract class SnoopySample<I, U, N extends I, A extends I,
S, T, X> extends SampleApp<I, U, N, A, S, T, X>
{
    public void testDocumentFromString()
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();

        final N document = documentFromString(pcx);

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);

        sf.setIndent(true);

        final StringWriter sw = new StringWriter();

        final GxSequenceHandler<A, S, T> serializer =
sf.newSerializer(sw);

        final GxModel<N, A, S, T> model = pcx.getModel();

        model.stream(document, true, true, serializer);

        // System.out.println(sw.toString());
    }

    public void testFragmentBuilder()
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();

        final N document = documentFromEvents(pcx);

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);

        sf.setIndent(true);

        final StringWriter sw = new StringWriter();

        final GxSequenceHandler<A, S, T> serializer =
sf.newSerializer(sw);

        final GxModel<N, A, S, T> model = pcx.getModel();

        model.stream(document, true, true, serializer);

        // System.out.println(sw.toString());
    }

    private N documentFromString(final GxProcessingContext<I, U, N, A, S, T, X> pcx)
    {
        final String strval = "" + "<?xml version='1.0' encoding='UTF-8'?>" + "<book isbn='0836217462'>" + " <title>Being a Dog Is a Full-Time Job</title>" + " <author>Charles M. Schultz</author>" + " <character>" + " <name>Snoopy</name>" + " <since>1950-10-04</since>" + " </character>" + "</book>";

        final GxDocumentBuilderFactory<N, S> factory = new DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

        final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();

        try
        {
            return builder.parse(new StringReader(strval), null);
        }
        catch (final IOException e)
        {
            throw new AssertionError();
        }
    }

    private N documentFromEvents(final GxProcessingContext<I, U, N, A, S, T, X> pcx)
    {
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        final S NULL_NS_URI = nameBridge.empty();
        final S BOOK = nameBridge.symbolize("book");
        final S ISBN = nameBridge.symbolize("isbn");
        final S TITLE = nameBridge.symbolize("title");
        final S AUTHOR = nameBridge.symbolize("author");
        final S CHARACTER = nameBridge.symbolize("character");
        final S NAME = nameBridge.symbolize("name");
        final S SINCE = nameBridge.symbolize("since");

        final GxFragmentBuilder<N, A, S, T> builder = pcx.newFragmentBuilder();

        // Note: Using try...finally not only ensures that elements get closed when errors
        // occur, it also helps to remind you to end elements and makes the levels in
        // the XML more obvious.
        builder.startDocument(null);
        try
        {
            builder.startElement(NULL_NS_URI, BOOK, "", null);
            try
            {
                builder.attribute(NULL_NS_URI, ISBN, "", "0836217462");
                builder.startElement(NULL_NS_URI, TITLE, "", null);
                try
                {
                    builder.text("Being a Dog Is a Full-Time Job");
                }
                finally
                {
                    builder.endElement();
                }
                builder.startElement(NULL_NS_URI, AUTHOR, "", null);
                try
                {
                    builder.text("Charles M. Schultz");
                }
                finally
                {
                    builder.endElement();
                }
                builder.startElement(NULL_NS_URI, CHARACTER, "", null);
                try
                {
                    builder.startElement(NULL_NS_URI, NAME, "", null);
                    try
                    {
                        builder.text("Snoopy");
                    }
                    finally
                    {
                        builder.endElement();
                    }
                    builder.startElement(NULL_NS_URI, SINCE, "", null);
                    try
                    {
                        builder.text("1950-10-04");
                    }
                    finally
                    {
                        builder.endElement();
                    }
                }
                finally
                {
                    builder.endElement();
                }
            }
            finally
            {
                builder.endElement();
            }
        }
        finally
        {
            builder.endDocument();
        }

        return builder.getNodes().get(0);
    }

    public void testExample() throws ParserConfigurationException, IOException, GxException, ExprException, URISyntaxException
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final Resolver resolver = getResolver();

        final URI xmlSystemId = new URI("hotel.xml");
        final Resolved<InputStream> xmlInput = resolver.resolveInputStream(xmlSystemId);

        final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

       final GxDocumentBuilder<N> builder = f.newDocumentBuilder();

        final N document = builder.parse(xmlInput.getResource(), xmlInput.getSystemId());

        final URI xslSystemId = new URI("hotel.xsl");
        final Resolved<InputStream> xslInput = resolver.resolveInputStream(xslSystemId);

       final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

       // poem.xsl uses version="2.0", but we want to use XPath 1.0 compatibility mode
        // so that arguments to functions are converted etc.
        compiler.setCompatibleMode(true);

        final GxTransform<I, U, N, A, S, T, X> compiled = compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);

        // TODO: Extract the configuration?
        // compiled.configure(sf);

        sf.setIndent(true);

        final StringWriter w = new StringWriter();

        final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);

        final GxTransformer<I, U, N, A, S, T, X> transformer = compiled.newTransformer();

        transformer.transform(document, pcx, handler);
    }

    public void testVariableBinding() throws ParserConfigurationException, IOException, GxException, ExprException, URISyntaxException
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final Resolver resolver = getResolver();

        final URI xslSystemId = new URI("email.xsl");
        final Resolved<InputStream> xslInput = resolver.resolveInputStream(xslSystemId);

        final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

        final GxTransform<I, U, N, A, S, T, X> compiled = compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());

        final GxTransformer<I, U, N, A, S, T, X> transformer = compiled.newTransformer();

        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
        final SmName<S> varName = nameBridge.name(new QName("to"));
        final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
        final X value = valueBridge.stringValue("David");

        transformer.bindVariableValue(varName, value);
        transformer.bindVariableValue(nameBridge.name(new QName("http://www.example.com", "from")), valueBridge.stringValue("Julie"));

        final N documentNode = transformer.transform(null, pcx);

        final GxModel<N, A, S, T> model = pcx.getModel();

        assertEquals(NodeKind.DOCUMENT, 
model.getNodeKind(documentNode));
        final N email = model.getFirstChildElement(documentNode);
        final N to = model.getFirstChildElementByName(email, 
nameBridge.symbolize("http://www.example.com"), 
nameBridge.symbolize("to"));
        assertEquals("David", model.getStringValue(to));
       final N from = model.getFirstChildElementByName(email, null, 
nameBridge.symbolize("from"));
        assertEquals("Julie", model.getStringValue(from));
        final N again = model.getFirstChildElementByName(email, 
nameBridge.symbolize("http://www.example.com"), null);
        assertEquals("David", model.getStringValue(again));
    }

    public void testExternalFunctions() throws 
ParserConfigurationException, IOException, GxException, ExprException, 
URISyntaxException
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = 
newProcessingContext();

        final Resolver resolver = getResolver();

        final URI xmlSystemId = new URI("exslt.xml");
        final Resolved<InputStream> xmlInput = 
resolver.resolveInputStream(xmlSystemId);

        final GxDocumentBuilderFactory<N, S> f = new 
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

       final GxDocumentBuilder<N> builder = f.newDocumentBuilder();

        final N document = builder.parse(xmlInput.getResource(), 
xmlInput.getSystemId());

        final URI xslSystemId = new URI("exslt.xsl");
        final Resolved<InputStream> xslInput = 
resolver.resolveInputStream(xslSystemId);

       final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new 
XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

        final String namespaceURI = "http://exslt.org/strings";
        final ExsltStringsFunctionGroup<I, U, N, A, S, T, X> 
functions = new ExsltStringsFunctionGroup<I, U, N, A, S, T, 
X>(namespaceURI, pcx);
        compiler.setFunctionSigns(namespaceURI, functions);
        compiler.setFunctionImpls(namespaceURI, functions);

        final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);

        // TODO: Extract the configuration.
        // compiled.configure(sf);

        sf.setIndent(true);

        final StringWriter w = new StringWriter();

        final GxSequenceHandler<A, S, T> handler =
sf.newSerializer(w);

        final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();

        transformer.transform(document, pcx, handler);

        // System.out.println(w.toString());
    }

    public void testHotel() throws ParserConfigurationException,
IOException, GxException, ExprException, URISyntaxException
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();

        final Resolver resolver = getResolver();

        final URI xmlSystemId = new URI("hotel.xml");
        final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(xmlSystemId);

        final GxDocumentBuilderFactory<N, S> f = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

       final GxDocumentBuilder<N> builder = f.newDocumentBuilder();

        final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());

        final URI xslSystemId = new URI("hotel.xsl");
        final Resolved<InputStream> xslInput =
resolver.resolveInputStream(xslSystemId);

```

```
        final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

        final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);

        // TODO: Extract the configuration.
        // compiled.configure(sf);

        sf.setIndent(true);

        final StringWriter w = new StringWriter();

        final GxSequenceHandler<A, S, T> handler =
sf.newSerializer(w);

        final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
        final SmName<S> varName = nameBridge.name(new
QName("MessageData"));
        final GxVariantBridge<I, N, A, X> valueBridge =
pcx.getVariantBridge();
        final X value = valueBridge.node(document);

        transformer.bindVariableValue(varName, value);

        transformer.transform(null, pcx, handler);

        // System.out.println(w.toString());
    }

    public void testHelloWorld() throws Exception
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        final GxLanguageToolKit<I, U, N, A, S, T, X> xtk = new
LanguageToolKit<I, U, N, A, S, T, X>(pcx);

        final GxExprContextStaticArgs<I, U, N, A, S, T, X> senv =
xtk.newStaticContextArgs();
        final String NAMESPACE = "http://www.peanuts.com";

        senv.getInScopeNamespaces().declarePrefix("nuts",
nameBridge.symbolize(NAMESPACE));

```

```
        final SnoopyFunctionGroup<I, U, N, A, S, T, X> peanutsFunctionGroup = new SnoopyFunctionGroup<I, U, N, A, S, T, X>(NAMESPACE, pcx);
        senv.setFunctionSigns(NAMESPACE, peanutsFunctionGroup);
        senv.setFunctionImpls(NAMESPACE, peanutsFunctionGroup);

        final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();

        final ExprResult<I, U, N, A, S, T, X> prepared = xtk.prepare("nuts:GetVariableProperty('foo','bar')", metaBridge.emptyType(), senv);

       final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();

        final GxExprContextDynamicArgs<I, U, N, A, S, T, X> darg = xtk.newDynamicContextArgs();

       final String strval = expr.stringFunction(xtk.emptyFocus(), darg, pcx);

        assertEquals("Bingo!", strval);
    }
}
```

Validating

```
package org.gxml.book.parsing;

import java.io.InputStream;
import java.net.URI;

import javax.xml.namespace.QName;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxApplication;
import org.gxml.sa.GxAtomBridge;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;
import org.gxml.xs.SmComponentBag;
import org.gxml.xs.SmExceptionCatcher;
import org.gxml.xs.SmMetaLoadArgs;
import org.gxml.xs.SmName;

import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;
import
com.tibco.gxml.sa.common.helpers.SmAtomBridgeOnGxAtomBridgeAdapter;
import com.tibco.gxml.xs.W3cXmlSchemaParser;

public abstract class BookValidatingParsingSample<I, U, N extends I,
A extends I, S, T, X> extends SampleApp<I, U, N, A, S, T, X>
{
    public void testValidatingParse() throws Exception
    {
        final GxApplication<I, U, N, A, S, T, X> app = this;

        final Resolver resolver = app.getResolver();

        final SmMetaLoadArgs args = new SmMetaLoadArgs();

        final SmExceptionCatcher errors = new SmExceptionCatcher();

        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
app.newProcessingContext();

        final Resolved<InputStream> resource =
getResolver().resolveInputStream(new URI("email.xsd"));

        final W3cXmlSchemaParser<A, S> parser = new
W3cXmlSchemaParser<A, S>(new SmAtomBridgeOnGxAtomBridgeAdapter<A,
S>(pcx.getAtomBridge()));

        final SmComponentBag<A, S> components =
parser.parse(resource.getLocation(), resource.getResource(),
resource.getSystemId(), errors, args, pcx);

        pcx.register(components);

        pcx.lock();

        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        assertEquals(0, errors.size());

        final URI xmlURI = new URI("email.xml");
        final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(xmlURI);

        final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

        // Enable validation of the XML input.
        factory.setValidating(true, nameBridge.name(new
QName("http://www.example.com", "email")));

```

```
        final GxDocumentBuilder<N> builder =
factory.newDocumentBuilder();

        // TODO: Need to catch errors...
        // builder.setExceptionHandler(errors);

        final N doc = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());

        assertEquals(0, errors.size());

        // System.out.println(serialize(doc, pcx));

        final GxModel<N, A, S, T> model = pcx.getModel();
        final GxAtomBridge<A, S> atomBridge = pcx.getAtomBridge();

        final N email = model.getFirstChildElement(doc);
        final S namespaceURI =
nameBridge.symbolize("http://www.example.com");
        final N sent = model.getFirstChildElementByName(email,
namespaceURI, nameBridge.symbolize("sent"));
        assertNotNull("model.getFirstChildElementByName", sent);
        final SmName<S> typeName = model.getTypeName(sent);
        assertNotNull("model.getTypeName", typeName);
       assertEquals("dateTime", typeName.toQName().getLocalPart());
        final A dateTime = model.getTypedValue(sent).get(0);

        //
assertTrue(metaBridge.sameAs(metaBridge.handle(pcx.getTypeDefinition(ty
pe)),
        // metaBridge.getType(SmNativeType.DATETIME)));

        assertEquals("2008-03-23T14:49:30-05:00",
atomBridge.getC14NForm(dateTime));
    }
}
```

Navigation

```
package org.gxml.book.parsing;

import java.io.InputStream;
import java.net.URI;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;

import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;

public abstract class BookNavigationParsingSample<I, U, N extends I, A extends I, S, T, X> extends SampleApp<I, U, N, A, S, T, X>
{
    public void testBooksByNealStephenson() throws Exception
    {
        final Resolver resolver = getResolver();

        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final URI systemId = new URI("books.xml");
        final Resolved<InputStream> source = resolver.resolveInputStream(systemId);

        final GxDocumentBuilderFactory<N, S> factory = new DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

        final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();

        final N doc = builder.parse(source.getResource(), source.getSystemId());

        final GxModel<N, A, S, T> model = pcx.getModel();

        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        final S namespaceURI = nameBridge.symbolize("http://www.example.com/books");

        final N inventory = model.getFirstChildElementByName(doc, namespaceURI, nameBridge.symbolize("inventory"));

        for (final N book : model.getChildElementsByName(inventory, namespaceURI, nameBridge.symbolize("book")))
        {
            boolean found = false;

           for (final N author : model.getChildElementsByName(book, namespaceURI, nameBridge.symbolize("author")))
            {
                if (model.getStringValue(author).equals("Neal Stephenson"))
                {
                    found = true;
                    break;
                }
            }

            if (found)
            {
                final N title =
model.getFirstChildElementByName(book, namespaceURI,
nameBridge.symbolize("title"));

                System.out.println(model.getStringValue(title));
            }
        }
    }

    public void testPurchaseOrder() throws Exception
    {
        final Resolver resolver = getResolver();

        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
        final GxModel<N, A, S, T> model = pcx.getModel();
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        final URI systemId = new URI("PurchaseOrder.xml");
        final Resolved<InputStream> source =
resolver.resolveInputStream(systemId);

        final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

        final GxDocumentBuilder<N> builder =
factory.newDocumentBuilder();

        final N po = builder.parse(source.getResource(),
source.getSystemId());

        final N root = model.getFirstChildElement(po);

       final N items = model.getFirstChildElementByName(root, null,
nameBridge.symbolize("items"));

        double total = 0;
        for (final N item : model.getChildElementsByName(items,
null, nameBridge.symbolize("item")))
        {
            System.out.println("partNum:" +
model.getAttributeStringValue(item, nameBridge.empty(),
nameBridge.symbolize("partNum")));

            final N price = model.getFirstChildElementByName(item,
null, nameBridge.symbolize("USPrice"));
            total +=
Double.valueOf(model.getStringValue(price)).doubleValue();
        }
        System.out.println("Grand total = " + total);
    }
}
```

Mutation

```
package org.gxml.book.mutable;

import java.math.BigDecimal;

import javax.xml.XMLConstants;

import org.gxml.book.common.MutableApp;
import org.gxml.sa.GxAtomBridge;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.mutable.GxModelMutable;
import org.gxml.sa.mutable.GxProcessingContextMutable;
import org.gxml.xdm.NodeKind;

/**
 * This sample illustrates the use of the optional mutability API.
 *
 * @author dholmes
 *
 * @param <I>
 * @param <U>
 * @param <N>
 * @param <A>
 * @param <S>
 * @param <T>
 * @param <X>
 */
public abstract class MutableSample<I, U, N extends I, A extends I, S, T, X> extends MutableApp<I, U, N, A, S, T, X>
{
    /**
     * This is a test of basic mutability through the optional mutability API.
     * Line 2
     * Line 3
     * Line 4 // OK
     */
    public void testIntroduction() throws Exception
    {
        final GxProcessingContextMutable<I, U, N, A, S, T, X> pcx = newProcessingContext();
        final GxAtomBridge<A, S> atomBridge = pcx.getAtomBridge();
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        /* // Create a new document. */
        final N documentNode = pcx.newDocument();

        final GxModelMutable<N, A, S, T> model = pcx.getModel();

        assertEquals(NodeKind.DOCUMENT, model.getNodeKind(documentNode));

        // Every node in the tree has an owner which is a document node. /* OK */
        final N owner = model.getOwner(documentNode);

        assertTrue(model.isSameNode(documentNode, owner));

        final S namespaceURI = nameBridge.symbolize("http://www.example.com");
        final S localName = nameBridge.symbolize("foo");
        final String prefix = "x";
        final N documentElement = model.createElement(owner, namespaceURI, localName, prefix);

        // Append the document element to the documentNode.
        model.appendChild(documentNode, documentElement);

        model.setNamespace(documentElement, prefix, namespaceURI);

        model.setAttribute(documentElement, nameBridge.empty(), nameBridge.symbolize("version"), XMLConstants.DEFAULT_NS_PREFIX, atomBridge.wrapAtom(atomBridge.createDecimal(BigDecimal.valueOf(2.7))));

        // Append four text nodes to the document element.
        model.appendChild(documentElement, model.createText(owner, "Hello"));
       model.appendChild(documentElement, model.createText(owner, " "));
        model.appendChild(documentElement, model.createText(owner, "World"));
        model.appendChild(documentElement, model.createText(owner, "!"));

       // Compress the four contiguous text nodes into a single text node.
        model.normalize(documentNode);

        @SuppressWarnings("unused")
        final String strval = serialize(documentNode, pcx);
        //System.out.println(strval);
    }
}
```

Serialization

```
package org.gxml.book.serialization;

import java.io.StringWriter;

import javax.xml.namespace.QName;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxProcessingContext;
import org.gxml.sa.GxSequenceHandler;
import org.gxml.xdm.Emulation;

import com.tibco.gxml.sa.processor.serialization.api.GxDocumentSerializer;
import com.tibco.gxml.sa.processor.serialization.api.GxDocumentSerializerFactory;
import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
import com.tibco.gxml.sa.processor.serialization.impl.DocumentSerializerFactory;
import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;

public abstract class IntroSerializationSample<I, U, N extends I, A extends I, S, T, X> extends SampleApp<I, U, N, A, S, T, X>
{
    public void exampleUsingDocumentSerializer(final N node, final GxProcessingContext<I, U, N, A, S, T, X> pcx)
    {
        final GxDocumentSerializerFactory<N, S> sf = new DocumentSerializerFactory<I, U, N, A, S, T, X>(pcx);

        // Configure for "pretty" printing.
        sf.setIndent(Boolean.TRUE);
        sf.setMethod(new QName("xml"));
        sf.setOmitXmlDeclaration(false);

        final StringWriter sw = new StringWriter();

        final GxDocumentSerializer<N> serializer = sf.newDocumentSerializer(sw);

        serializer.serialize(node);

        System.out.print(sw.toString());
    }

    public void exampleUsingSequenceHandler(final N node, final GxProcessingContext<I, U, N, A, S, T, X> pcx)
    {
        final GxSerializerFactory<I,U,N,A, S, T,X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);

        // Configure for "pretty" printing.
        sf.setIndent(Boolean.TRUE);
        sf.setMethod(new QName("xml"));
        sf.setOmitXmlDeclaration(false);
        sf.setEmulation(Emulation.C14N);

        final StringWriter sw = new StringWriter();

        final GxSequenceHandler<A, S, T> serializer = sf.newSerializer(sw);

        final GxModel<N, A, S, T> model = pcx.getModel();

        model.stream(node, true, true, serializer);

        System.out.print(sw.toString());
    }
}
```

XPath

```
package org.gxml.book.xpath;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxMetaBridge;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.sa.GxVariantBridge;
import org.gxml.xdm.Emulation;
import org.gxml.xs.SmName;
import org.gxml.xs.SmNativeType;

import com.tibco.gxml.sa.api.common.lang.ExprResult;
import com.tibco.gxml.sa.api.common.lang.GxExpr;
import com.tibco.gxml.sa.api.common.lang.GxExprContextDynamicArgs;
import com.tibco.gxml.sa.api.common.lang.GxExprContextStaticArgs;
import com.tibco.gxml.sa.api.common.lang.GxFocus;
import com.tibco.gxml.sa.api.common.lang.GxLanguageToolKit;
import com.tibco.gxml.sa.processor.xquery.LanguageToolKit;
import com.tibco.gxmlsa.processor.org.exslt.math.ExsltMathFunctionGroup;

```

```
public abstract class XPathSample<I, U, N extends I, A extends I, S, T, X> extends SampleApp<I, U, N, A, S, T, X>
{
    public void testGettingStarted() throws Exception
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

       // For demonstration purposes, register the language toolkit with the processing context.
        pcx.register("xyz", new LanguageToolKit<I, U, N, A, S, T, X>(pcx));

        @SuppressWarnings("unchecked")
        // Immediately get back the registered processor.
        GxLanguageToolKit<I, U, N, A, S, T, X> xtk = pcx.getProcessor("xyz", GxLanguageToolKit.class);

        final GxExprContextStaticArgs<I, U, N, A, S, T, X> sarg = xtk.newStaticContextArgs();

        final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();

        final ExprResult<I, U, N, A, S, T, X> prepared = xtk.prepare("concat('Hello', ', ', 'World', '!')", metaBridge.emptyType(), sarg);
       final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();

        final GxExprContextDynamicArgs<I, U, N, A, S, T, X> darg = xtk.newDynamicContextArgs();

        final String strval = expr.stringFunction(xtk.emptyFocus(), darg, pcx);

        assertEquals("Hello, World!", strval);
    }

    public void testBindingVariables() throws Exception
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final GxLanguageToolKit<I, U, N, A, S, T, X> xtk = new LanguageToolKit<I, U, N, A, S, T, X>(pcx);

        final GxExprContextStaticArgs<I, U, N, A, S, T, X> statArgs = xtk.newStaticContextArgs();
        statArgs.setEmulation(Emulation.MODERN);

        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
        final SmName<S> varName = new SmName<S>(nameBridge.symbolize("x"), nameBridge);

        final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();
        statArgs.bindVariableType(varName, metaBridge.getType(SmNativeType.STRING));

        final String es = "concat('Hello', ', ', $x, '!')";
        final T sfocus = metaBridge.emptyType();

        final ExprResult<I, U, N, A, S, T, X> prepared = xtk.prepare(es, sfocus, statArgs);

        final GxExprContextDynamicArgs<I, U, N, A, S, T, X> dynArgs = xtk.newDynamicContextArgs();
        dynArgs.setEmulation(Emulation.MODERN);

        final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
        final X value = valueBridge.stringValue("World");
        dynArgs.bindVariableValue(varName, value);

        final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();
        final GxFocus<I> dfocus = xtk.emptyFocus();
        final String strval = expr.stringFunction(dfocus, dynArgs, pcx);

        assertEquals("Hello, World!", strval);
    }

    public void testEXSLT() throws Exception
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        final GxLanguageToolKit<I, U, N, A, S, T, X> xtk = new LanguageToolKit<I, U, N, A, S, T, X>(pcx);

        final GxExprContextStaticArgs<I, U, N, A, S, T, X> sarg = xtk.newStaticContextArgs();
        sarg.getInScopeNamespaces().declarePrefix("math", nameBridge.symbolize("http://exslt.org/math"));
        final ExsltMathFunctionGroup<I, U, N, A, S, T, X> exsltMathFunctionGroup = new ExsltMathFunctionGroup<I, U, N, A, S, T, X>("http://exslt.org/math", pcx);
        sarg.setFunctionSigns("http://exslt.org/math", exsltMathFunctionGroup);
        // The function implementations can be provided now or just prior to execution.
        sarg.setFunctionImpls("http://exslt.org/math", exsltMathFunctionGroup);

        final GxMetaBridge<A, S, T> metaBridge =
pcx.getMetaBridge();

        final ExprResult<I, U, N, A, S, T, X> prepared =
xtk.prepare("math:exp(1)", metaBridge.emptyType(), sarg);

        final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();

        final GxExprContextDynamicArgs<I, U, N, A, S, T, X> darg =
xtk.newDynamicContextArgs();
        // Here we also (redundantly) provide the function
implementations just prior to execution.
        darg.setFunctionImpls("http://exslt.org/math",
exsltMathFunctionGroup);

        final String strval = expr.stringFunction(xtk.emptyFocus(),
darg, pcx);

        assertEquals("2.718281828459045", strval);
    }

    public void testExpressionType() throws Exception
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();

        final GxLanguageToolKit<I, U, N, A, S, T, X> xtk = new
LanguageToolKit<I, U, N, A, S, T, X>(pcx);

        final GxExprContextStaticArgs<I, U, N, A, S, T, X> sarg =
xtk.newStaticContextArgs();

        final GxMetaBridge<A, S, T> metaBridge =
pcx.getMetaBridge();

        final ExprResult<I, U, N, A, S, T, X> prepared =
xtk.prepare("'Hello'", metaBridge.emptyType(), sarg);
        /* final GxExpr<I, U, N, A, S, T, X> expr =
*/prepared.getExpr();
        /* final GxExprInfo<T> info = */prepared.getInfo();
    }
}
```

XSLT

```
package org.gxml.book.xslt;

import java.io.IOException;
import java.io.InputStream;
import java.io.StringReader;
import java.io.StringWriter;
import java.net.URI;
import java.net.URISyntaxException;

import javax.xml.namespace.QName;
import javax.xml.parsers.ParserConfigurationException;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxException;
import org.gxml.sa.GxMetaBridge;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.sa.GxSequenceHandler;
import org.gxml.sa.GxVariantBridge;
import org.gxml.xdm.NodeKind;
import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;
import org.gxml.xs.SmName;
import org.gxml.xs.SmNativeType;

import com.tibco.gxml.sa.api.common.lang.ExprException;
import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;
import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
import com.tibco.gxml.sa.processor.xslt.GxTransform;
import com.tibco.gxml.sa.processor.xslt.GxTransformBuilder;
import com.tibco.gxml.sa.processor.xslt.GxTransformer;
import com.tibco.gxml.sa.processor.xslt.XSLTransformBuilder;
import com.tibco.gxmlsa.processor.org.exslt.strings.ExsltStringsFunctionGroup;

public abstract class XSLTSample<I, U, N extends I, A extends I, S, T, X> extends SampleApp<I, U, N, A, S, T, X>
{
    public void testExample() throws ParserConfigurationException, IOException, GxException, ExprException, URISyntaxException
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();
        final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        final Resolver resolver = getResolver();

        final URI xmlSystemId = new URI("hotel.xml");
        final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(xmlSystemId);

        final GxDocumentBuilderFactory<N, S> f = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
        f.setIgnoreComments(false);

        final GxDocumentBuilder<N> builder = f.newDocumentBuilder();

        final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());

        final URI xslSystemId = new URI("hotel.xsl");
        final Resolved<InputStream> xslInput =
resolver.resolveInputStream(xslSystemId);

        final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

        compiler.setCompatibleMode(true);
        // compiler.setRestrictedMode(true); // XSLT 2.0 subset for
mapper.

        // Specify the static type for the context item:
        // document-node(element(*,xs:untyped))
        final T documentType =
metaBridge.documentType(metaBridge.elementType(new SmName<S>(null,
null, nameBridge), metaBridge.getType(SmNativeType.UNTYPED), false));
        compiler.setFocus(documentType);

        final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);

        // TODO: Extract output configuration.
        // compiled.configure(sf);

        sf.setIndent(true);

        final StringWriter w = new StringWriter();

        final GxSequenceHandler<A, S, T> handler =
sf.newSerializer(w);

        final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();

        transformer.transform(document, pcx, handler);

        @SuppressWarnings("unused")
        final String s = w.toString();
        // System.out.println(s);
    }

    @SuppressWarnings("unused")
    private void bar(final GxProcessingContext<I, U, N, A, S, T, X> pcx)
    {
        try
        {
            final GxTransformBuilder<I, U, N, A, S, T, X> builder = new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

            final GxTransform<I, U, N, A, S, T, X> transform = builder.prepareTransform(new StringReader("<x xsl:version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'></x>"), new URI(""));

            final GxTransformer<I, U, N, A, S, T, X> transformer = transform.newTransformer();

            final N document = transformer.transform(null, pcx);

            final GxModel<N, A, S, T> model = pcx.getModel();

            final N element = model.getFirstChild(document);

           final String name = model.getLocalNameAsString(element);

            // System.out.println("XSLT: " + name);
        }
        catch (final Throwable e)
        {
            e.printStackTrace();
        }
    }

    public void skipVariableBinding() throws ParserConfigurationException, IOException, GxException, ExprException, URISyntaxException
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final Resolver resolver = getResolver();

        final URI xslSystemId = new URI("email.xsl");
        final Resolved<InputStream> xslInput = resolver.resolveInputStream(xslSystemId);

       final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

```

```
        final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());

        final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();

        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
        final SmName<S> varName = nameBridge.name(new QName("to"));
        final GxVariantBridge<I, N, A, X> valueBridge =
pcx.getVariantBridge();
        final X value = valueBridge.stringValue("David");

        transformer.bindVariableValue(varName, value);
        transformer.bindVariableValue(nameBridge.name(new
QName("http://www.example.com", "from")),
valueBridge.stringValue("Julie"));

        final N documentNode = transformer.transform(null, pcx);

        final GxModel<N, A, S, T> model = pcx.getModel();

        assertEquals(NodeKind.DOCUMENT,
model.getNodeKind(documentNode));
        final N email = model.getFirstChildElement(documentNode);
        final N to = model.getFirstChildElementByName(email,
nameBridge.symbolize("http://www.example.com"),
nameBridge.symbolize("to"));
        assertEquals("David", model.getStringValue(to));
        final N from = model.getFirstChildElementByName(email, null,
nameBridge.symbolize("from"));
        assertEquals("Julie", model.getStringValue(from));
        final N again = model.getFirstChildElementByName(email,
nameBridge.symbolize("http://www.example.com"), null);
        assertEquals("David", model.getStringValue(again));
    }

    public void skipExternalFunctions() throws
ParserConfigurationException, IOException, GxException, ExprException,
URISyntaxException
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();

        final Resolver resolver = getResolver();

        final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(new URI("exslt.xml"));

        final GxDocumentBuilderFactory<N, S> f = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

        final GxDocumentBuilder<N> builder = f.newDocumentBuilder();

        final N document = builder.parse(xmlInput.getResource(), xmlInput.getSystemId());

        final Resolved<InputStream> xslInput = resolver.resolveInputStream(new URI("exslt.xsl"));

       final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

        final String namespaceURI = "http://exslt.org/strings";
        final ExsltStringsFunctionGroup<I, U, N, A, S, T, X> functions = new ExsltStringsFunctionGroup<I, U, N, A, S, T, X>(namespaceURI, pcx);
        compiler.setFunctionSigns(namespaceURI, functions);
        compiler.setFunctionImpls(namespaceURI, functions);

        final GxTransform<I, U, N, A, S, T, X> compiled = compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);

        // TODO: Extract configuration.
        // compiled.configure(sf);

        sf.setIndent(true);

        final StringWriter w = new StringWriter();

        final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);

        final GxTransformer<I, U, N, A, S, T, X> transformer = compiled.newTransformer();

        transformer.transform(document, pcx, handler);

        // System.out.println(w.toString());
    }

    public void skipHotel() throws ParserConfigurationException, IOException, GxException, ExprException, URISyntaxException
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final Resolver resolver = getResolver();

        final Resolved<InputStream> xmlInput = resolver.resolveInputStream(new URI("hotel.xml"));

        final GxDocumentBuilderFactory<N, S> f = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

       final GxDocumentBuilder<N> builder = f.newDocumentBuilder();

        final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());

        final Resolved<InputStream> xslInput =
resolver.resolveInputStream(new URI("hotel.xsl"));

       final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

        final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());

        final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
        final SmName<S> varName = nameBridge.name(new
QName("MessageData"));
        final GxVariantBridge<I, N, A, X> valueBridge =
pcx.getVariantBridge();
        final X value = valueBridge.node(document);

        transformer.bindVariableValue(varName, value);

        final N documentNode = transformer.transform(null, pcx);

        final GxModel<N, A, S, T> model = pcx.getModel();

        assertEquals(NodeKind.DOCUMENT,
model.getNodeKind(documentNode));
        final N searchHotelRequest =
model.getFirstChildElement(documentNode);
        final N parameters =
model.getFirstChildElementByName(searchHotelRequest,
nameBridge.symbolize("http://xmlns.example.com/1189038295781"),
nameBridge.symbolize("parameters"));
        final N searchHotel =
model.getFirstChildElementByName(parameters,
nameBridge.symbolize("http://www.xyzcorp/procureservice/QueryGDS_Europe
/"), nameBridge.symbolize("searchHotel"));
        final N country =
model.getFirstChildElementByName(searchHotel,
nameBridge.symbolize("http://www.xyzcorp/procureservice/QueryGDS_Europe
/"), nameBridge.symbolize("country"));
        assertEquals("USA", model.getStringValue(country));
    }
}
```

XQuery

```
package org.gxml.book.xquery;

import java.io.StringWriter;
import java.math.BigInteger;
import java.net.URI;

import javax.xml.namespace.QName;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxAtomBridge;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.sa.GxSequenceHandler;
import org.gxml.sa.GxVariantBridge;
import org.gxml.xs.SmName;

import com.tibco.gxml.sa.api.common.lang.GxXQConnection;
import com.tibco.gxml.sa.api.common.lang.GxXQDataSource;
import com.tibco.gxml.sa.api.common.lang.GxXQExpression;
import com.tibco.gxml.sa.api.common.lang.GxXQPreparedExpression;
import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
import com.tibco.gxml.sa.processor.xquery.XQEngine;
import com.tibco.gxml.sa.processor.xquery.XQErrorCatcher;

/**
 * Introduction to XQuery.
 */
public abstract class XQuerySample<I, U, N extends I, A extends I, S, T, X> extends SampleApp<I, U, N, A, S, T, X>
{
    public void testExample() throws Exception
    {
        // Obtain a new processing context from the application.
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A, S, T, X>(pcx);

        final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();

        final String expression = "<x>{text{for $i in (1,2,3,4) return $i * 2}}</x>";

        final GxXQPreparedExpression<I, U, N, A, S, T, X> expr = conn.prepareExpression(expression);

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);
        sf.setMethod(new QName("xml"));
        sf.setOmitXmlDeclaration(true);
        final StringWriter sw = new StringWriter();
        final GxSequenceHandler<A, S, T> handler = sf.newSerializer(sw);

        expr.executeQuery(handler);

        final String actual = sw.toString();
        assertEquals(expression, "<x>2 4 6 8</x>", actual);
    }

    public void testGettingStarted() throws Exception
    {
        // Obtain a new processing context from the application.
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A, S, T, X>(pcx);

        final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();

        final GxXQExpression<I, U, N, A, S, T, X> expr = conn.createExpression();

        final String es = "for $n in fn:doc('catalog.xml')//item return fn:data($n/name)";

        final URI systemId = new URI("catalog.xml");

        expr.setBaseURI(systemId);

        @SuppressWarnings("unused")
        final X value = expr.executeQuery(es);
    }

    public void testHelloWorld() throws Exception
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = this.newProcessingContext();

        final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A, S, T, X>(pcx);

        final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();

        conn.setScriptingMode(true);

        final String expression = "declare variable $x external; concat('Hello, ',$x, '!')";

        final GxXQPreparedExpression<I, U, N, A, S, T, X> expr = conn.prepareExpression(expression);

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);
        sf.setOmitXmlDeclaration(true);
        sf.setIndent(false);
        sf.setMethod(new QName("xml"));
        final StringWriter sw = new StringWriter();
        final GxSequenceHandler<A, S, T> handler = sf.newSerializer(sw);

        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
        final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();

        final SmName<S> varName = new SmName<S>(nameBridge.symbolize("x"), nameBridge);
        final X value = valueBridge.stringValue("World");

        expr.bindVariableValue(varName, value);

        expr.executeQuery(handler);

        String actual = sw.toString();
        assertEquals(expression, "Hello, World!", actual);
    }

    public void testMergeTextNodes() throws Exception
    {
        // Obtain a new processing context from the application.
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A, S, T, X>(pcx);

        final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();

        // final String expression = "";
       final String expression = "count((element elem {1, 'string', 1,2e3})/text())";

```

```
        final GxXQPreparedExpression<I, U, N, A, S, T, X> expr = conn.prepareExpression(expression);

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);
        sf.setMethod(new QName("xml"));
        sf.setOmitXmlDeclaration(true);
        final StringWriter sw = new StringWriter();
        final GxSequenceHandler<A, S, T> handler = sf.newSerializer(sw);

        expr.executeQuery(handler);

        final String actual = sw.toString();
        assertEquals(expression, "1", actual);
    }

    public void testProblem() throws Exception
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = this.newProcessingContext();

        final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A, S, T, X>(pcx);

        final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();

        final XQErrorCatcher messages = new XQErrorCatcher();

        conn.setErrorHandler(messages);
        conn.setCompatibleMode(false);
        conn.setScriptingMode(true);

        final String expression = "(xs:untypedAtomic('1'),xs:untypedAtomic('2')) = (xs:untypedAtomic('2.0'),2.0)";

        final GxXQPreparedExpression<I, U, N, A, S, T, X> expr = conn.prepareExpression(expression);

        final X value = expr.executeQuery();

        final GxVariantBridge<I, N, A, X> variantBridge = pcx.getVariantBridge();
        switch (variantBridge.getNature(value))
        {
            case ITEMS:
            {
                @SuppressWarnings("unused")
                final Iterable<I> items = variantBridge.getItemSet(value);
                // System.out.println(items);
            }
            break;
            case ATOM:
            {
                @SuppressWarnings("unused")
                final A atom = variantBridge.getAtom(value);
                @SuppressWarnings("unused")
                final GxAtomBridge<A, S> atomBridge = 
pcx.getAtomBridge();
              // System.out.println(atomBridge.getC14NForm(atom));
            }
            break;
            case STRING:
            {
                @SuppressWarnings("unused")
              final String strval = variantBridge.getString(value);
                // System.out.println(strval);
            }
            break;
            case INTEGER:
            {
                @SuppressWarnings("unused")
                final BigInteger integer = 
variantBridge.getInteger(value);
                // System.out.println(integer);
            }
            break;
            default:
            {
                throw new 
AssertionError(variantBridge.getNature(value));
            }
        }
    }

    public void testTyping() throws Exception
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = 
this.newProcessingContext();

        final GxXQDataSource<I, U, N, A, S, T, X> ds = new 
XQEngine<I, U, N, A, S, T, X>(pcx);

        final GxXQConnection<I, U, N, A, S, T, X> conn = 
ds.getConnection();

        conn.setScriptingMode(true);

        final XQErrorCatcher messages = new XQErrorCatcher();

        conn.setErrorHandler(messages);

        final String expression = "declare variable $x external; contains(string(number($x)),'NaN')";

        final GxXQPreparedExpression<I, U, N, A, S, T, X> expr = conn.prepareExpression(expression);

        final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);
        sf.setOmitXmlDeclaration(true);
        sf.setIndent(false);
        sf.setMethod(new QName("xml"));
        final StringWriter sw = new StringWriter();
        final GxSequenceHandler<A, S, T> handler = sf.newSerializer(sw);

        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
        final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();

        final SmName<S> varName = new SmName<S>(nameBridge.symbolize("x"), nameBridge);
        final X value = valueBridge.doubleValue(5.0);

        expr.bindVariableValue(varName, value);

        expr.executeQuery(handler);

        String actual = sw.toString();
        assertEquals(expression, "false", actual);
    }
}
```

Validation

```
package org.gxml.book.validation;

import java.io.InputStream;
import java.net.URI;
import java.util.LinkedList;
import java.util.List;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxFragmentBuilder;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxProcessingContext;
import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;
import org.gxml.xs.SmException;
import org.gxml.xs.SmExceptionCatcher;
import org.gxml.xs.SmExceptionHandler;
import org.gxml.xs.SmMetaLoadArgs;

import com.tibco.gxml.sa.common.helpers.DocumentBuilderFactory;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilderFactory;
import com.tibco.gxml.sa.common.helpers.SmAtomBridgeOnGxAtomBridgeAdapter;
import com.tibco.gxml.sa.processor.validation.GxContentValidator;
import com.tibco.gxml.sa.processor.validation.GxValidatorCache;
import com.tibco.gxml.sa.processor.validation.GxValidatorCacheFactory;
import com.tibco.gxml.sa.processor.validation.ValidatorCacheFactory;
import com.tibco.gxml.xs.W3cXmlSchemaParser;

public abstract class ValidationSample<I, U, N extends I, A extends I, S, T, X> extends SampleApp<I, U, N, A, S, T, X>
{
    public void testByteStreamValidation() throws Exception
    {
        // Load a top-level schema into the processing context.
        final List<Resolved<InputStream>> resources = new LinkedList<Resolved<InputStream>>();
        resources.add(getResolver().resolveInputStream(new URI("PurchaseOrder.xsd")));

        final SmExceptionCatcher errors = new SmExceptionCatcher();
        final SmMetaLoadArgs args = new SmMetaLoadArgs();

        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final W3cXmlSchemaParser<A, S> parser = new W3cXmlSchemaParser<A, S>(new SmAtomBridgeOnGxAtomBridgeAdapter<A, S>(pcx.getAtomBridge()));

        for (final Resolved<InputStream> resource : resources)
        {
            pcx.register(parser.parse(resource.getLocation(), resource.getResource(), resource.getSystemId(), errors, args, pcx));
        }

        pcx.lock();

        // Create a validator...
        final GxValidatorCacheFactory<A, S, T> vcf = new ValidatorCacheFactory<I, U, N, A, S, T, X>(pcx);
        final GxValidatorCache<A, S, T> vc = vcf.newValidatorCache();
        final GxContentValidator<A, S, T> validator =
vc.newContentValidator();

        // Set the downstream event handler which contains
annotations and typed content.
        // validator.setGxContentHandler(/* ...*/null);
        validator.setExceptionHandler(errors);

        // The document node that we wish to validate.
        final Resolved<InputStream> xmlInput =
getResolver().resolveInputStream(new URI("PurchaseOrder.xml"));

        final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

        final GxDocumentBuilder<N> builder =
factory.newDocumentBuilder();

        final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());

        // Stream the document into the validator.
        final GxModel<N, A, S, T> model = pcx.getModel();

        model.stream(document, true, true, validator);

        if (errors.size() > 0)
        {
            // You've got errors.'
        }
    }

    public void testTreeValidation() throws Exception
    {
        final Resolver resolver = getResolver();

        // Load a top-level schema into the processing context.
        final List<Resolved<InputStream>> resources = new
LinkedList<Resolved<InputStream>>();
        resources.add(getResolver().resolveInputStream(new
URI("PurchaseOrder.xsd")));

        final SmExceptionCatcher errors = new SmExceptionCatcher();
        final SmMetaLoadArgs args = new SmMetaLoadArgs();

        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
        final W3cXmlSchemaParser<A, S> parser = new
W3cXmlSchemaParser<A, S>(new SmAtomBridgeOnGxAtomBridgeAdapter<A,
S>(pcx.getAtomBridge()));
        for (final Resolved<InputStream> resource : resources)
        {
            pcx.register(parser.parse(resource.getLocation(),
resource.getResource(), resource.getSystemId(), errors, args, pcx));
        }
        pcx.lock();
        // The document node that we wish to validate.
        @SuppressWarnings("unused")
        final URI xmlLocation = new URI("PurchaseOrder.xml");
        final URI xmlSystemId = new URI("PurchaseOrder.xml");
        final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(xmlSystemId);

        final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

        final GxDocumentBuilder<N> builder =
factory.newDocumentBuilder();

        final N documentIn = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());

        @SuppressWarnings("unused")
        final N documentOut = validate(documentIn, errors, pcx);

        if (errors.size() > 0)
        {
            // You've got errors.'
            for (@SuppressWarnings("unused")
            final SmException error : errors)
            {
               // System.out.println(error.getLocalizedMessage());
            }
        }
    }

    /**
     * This static function illustrates a helper function for
validating a document tree. <br/>
     * Note that we assume that the processing context is already
loaded with meta-data.
     *
     * @param node
     *            The input document.
     * @param errors
     *            The error handler.
     * @param pcx
     *            The processing context.
     */
    public static <I, U, N extends I, A extends I, S, T, X> N
validate(final N node, final SmExceptionHandler errors, final
GxProcessingContext<I, U, N, A, S, T, X> pcx)
    {
        final GxValidatorCacheFactory<A, S, T> vcf = new ValidatorCacheFactory<I, U, N, A, S, T, X>(pcx);

        // We already have a tree as input so we'll use the content validator'
        // and stream the document in as a bunch of events (a bit like SAX, but not lexical).
        final GxValidatorCache<A, S, T> vc = vcf.newValidatorCache();

        final GxContentValidator<A, S, T> validator = vc.newContentValidator();

        validator.setExceptionHandler(errors);

        final GxModel<N, A, S, T> model = pcx.getModel();

        // We want to produce a node so we'll need a fragment builder at the output.'
        final GxFragmentBuilder<N, A, S, T> builder = pcx.newFragmentBuilder();

        // Connect the pieces together so that the validation output builds a tree.
        validator.setGxContentHandler(builder);

        // Make it so!
        model.stream(node, true, true, validator);

        // Practice safe coding: We don't know what might happen if there are errors.'
        final List<? extends N> nodes = builder.getNodes();
        if (nodes.size() > 0)
        {
            return nodes.get(0);
        }
        else
        {
            return null;
        }
    }

```

Index