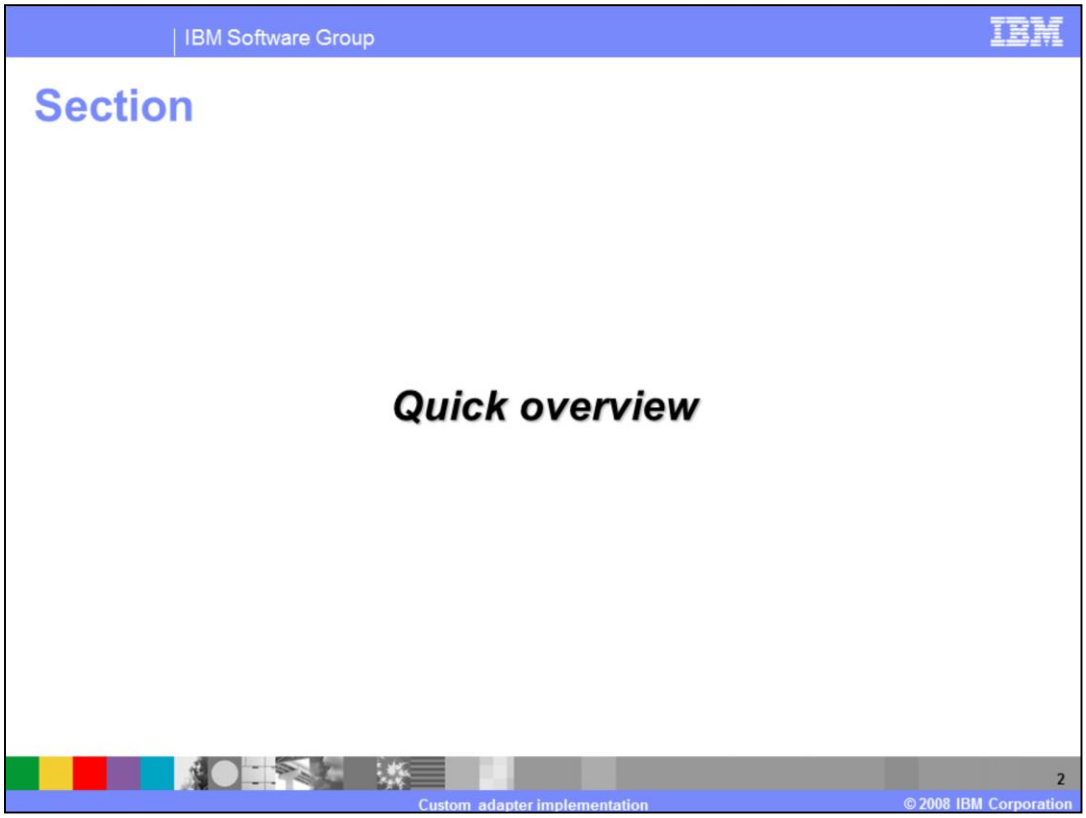
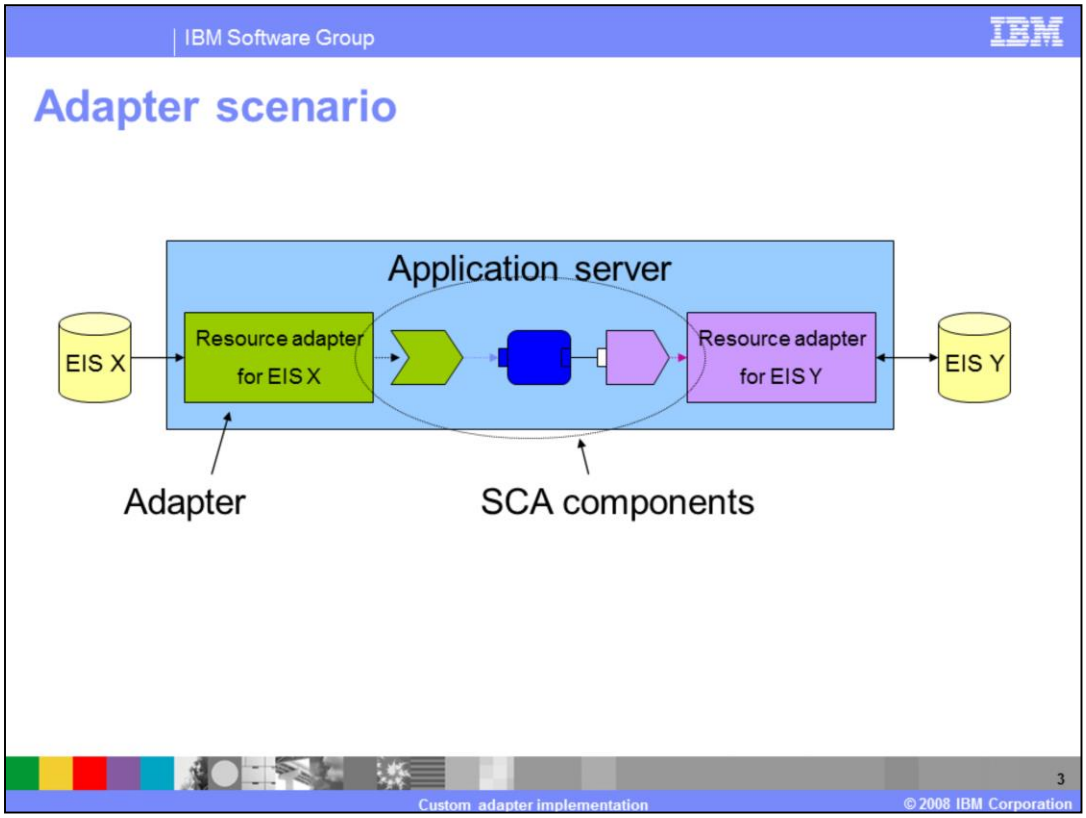
This presentation covers the high level details on how to use the WebSphere® adapter toolkit to develop custom WebSphere JCA adapters.



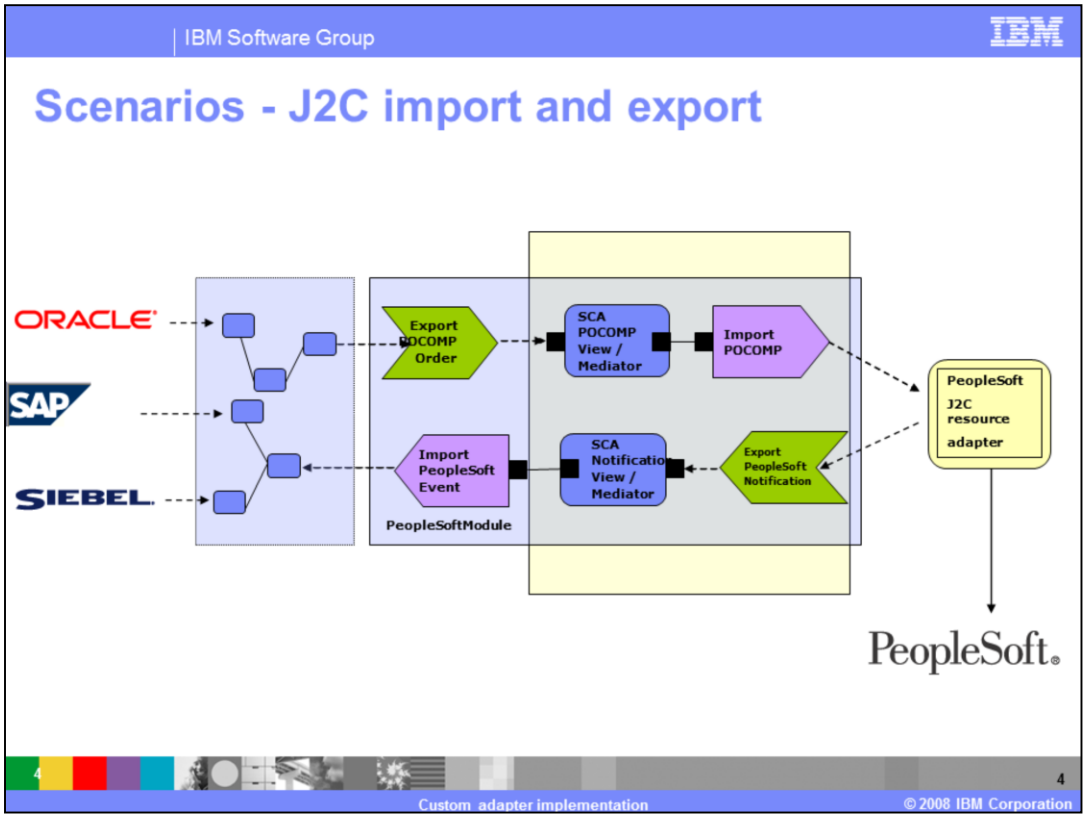
The next section provides a brief overview of a scenario where adapters can be used.



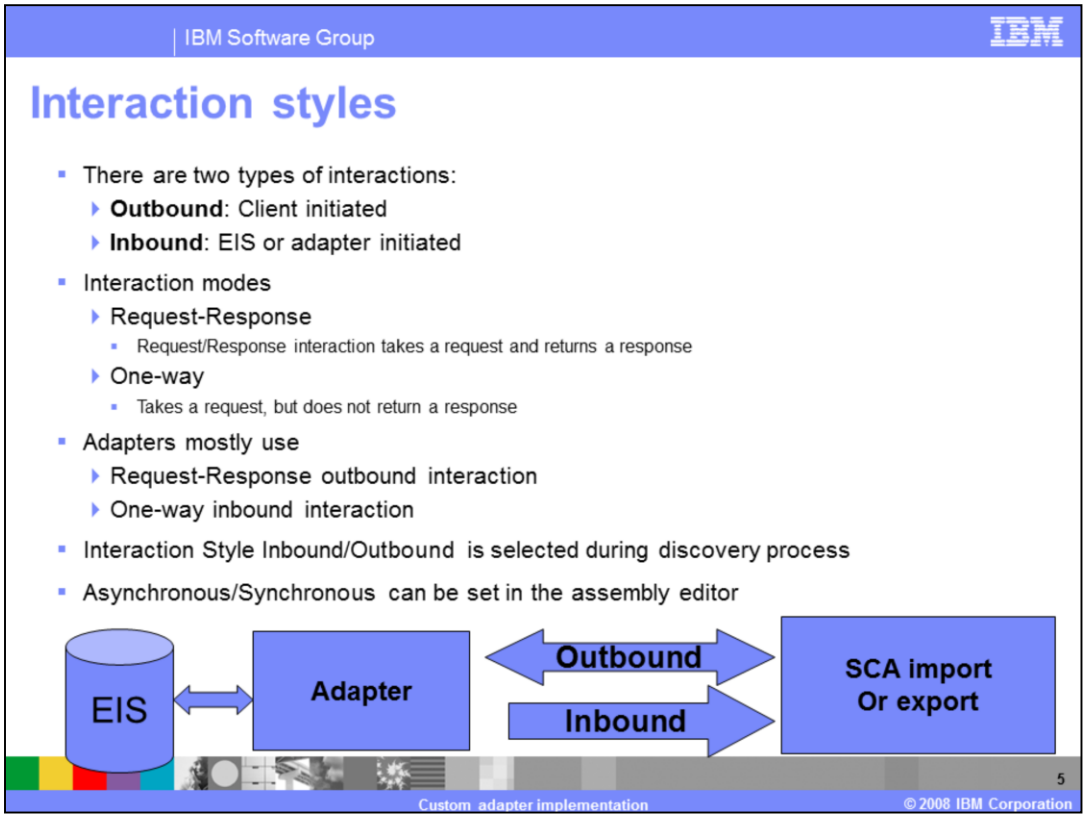
This slide shows an end-to-end integration scenario involving adapters. This is not the only scenario adapters are involved in, but it does place several requirements on adapters.

Walking through the scenario, data in EIS X is changed. The adapter for EIS X is configured to receive those changes. The adapter will convert the data into the runtime's format (SDO for WebSphere Process Server), then publish it to the listening application. The application is a business process of some sort that can involve mapping, relationships, or any number of runtime components. The business process will eventually call the service for the Adapter for EIS Y. This adapter will call the EIS to perform the necessary actions on the data such as creating or deleting the represented object.

If you are synchronizing data between EIS X and EIS Y, it is important to get an uninterrupted stream of events from EIS X. If any events are missed, then EIS Y might not have the complete copy. If any are duplicated, that might cause errors in the business process. If events arrive in the wrong order (an update before a create of the same object), that will also cause errors. Adapters provide assured once delivery, which allows the adapters to deliver the events, and keep the events from the source EIS in order.



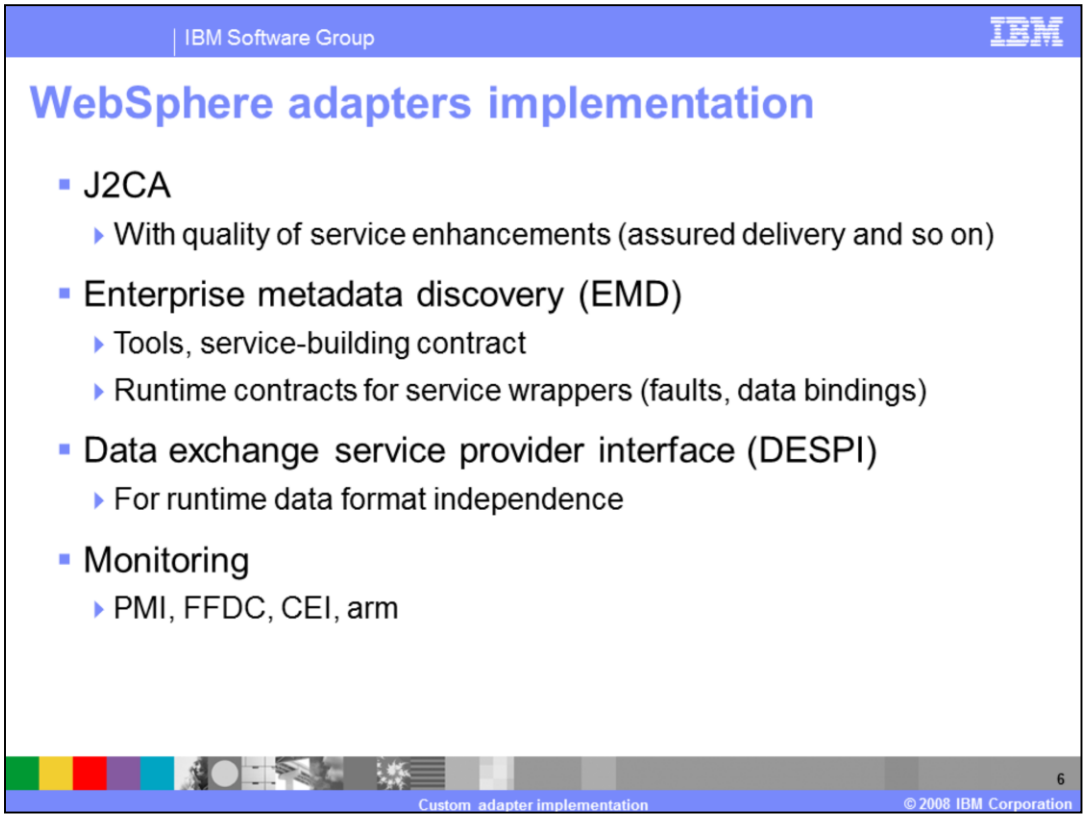
In WebSphere Process Server, WebSphere Enterprise Service Bus, adapters are represented as EIS exports and EIS imports. The imports and exports are generated during the enterprise service discovery process by the external service wizard in WebSphere Integration Developer.



It is important to understand the different interaction patterns. The common interaction patters used in WebSphere Adapters are outbound and inbound. Outbound is a two-way interaction with synchronous flow of information and inbound interaction style involves asynchronous one-way flow of information and data.

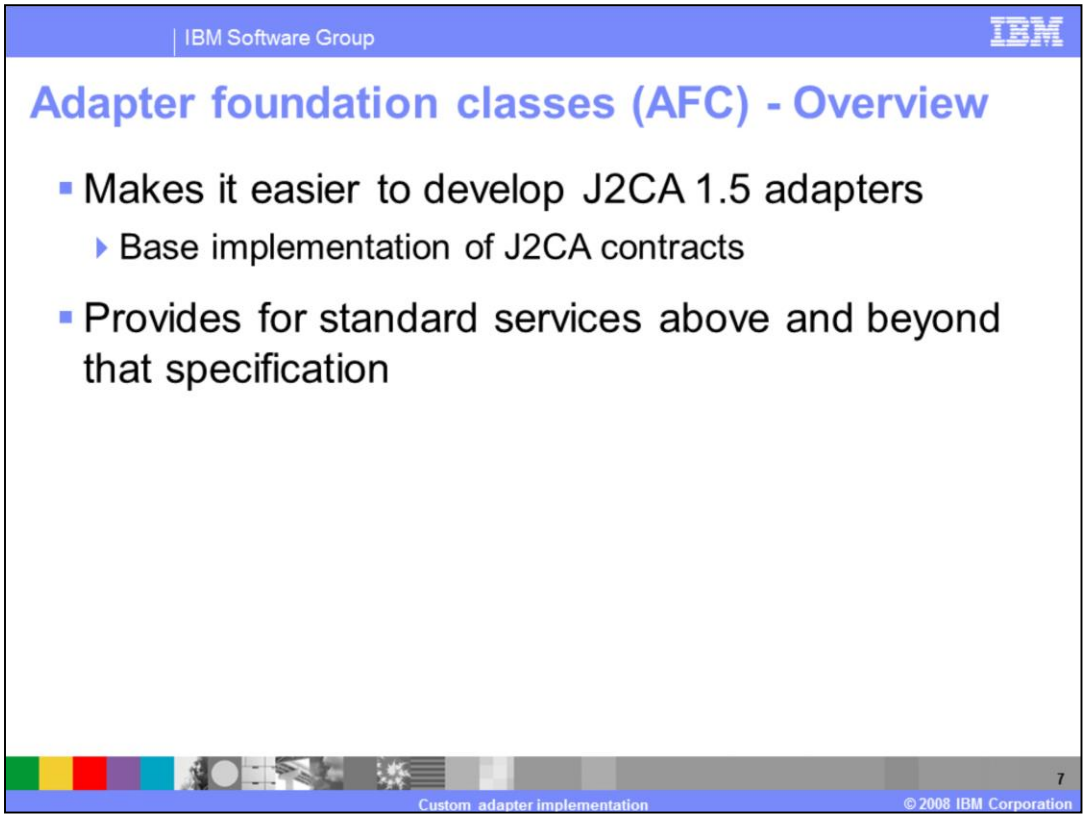
WebSphere adapter for SAP supports inbound two-way communication but it is not a common scenario.

In inbound, synchronous or asynchronous is set in the assembly editor and is a property of the EIS binding. Asynchronous in this context means that the EIS binding will place the data on a queue and return immediately. Synch means that the binding will call the next component directly, blocking the current thread until the entire process is finished. This has implications for your transaction boundaries. When the export is in asynchronous mode, the transaction is between the adapter and the queue only. When the export is in synchronous mode, the transaction can be propagated through the entire system.



WebSphere adapters implement J2CA which is the base runtime contract, Enterprise Metadata Discovery (EMD) specification for tool support which is a service building contract.

Data Exchange Service Provider Interface (DESPI) which is the contract that allows to abstract the data format and run in multiple runtime environments. Support is also provided for various monitoring and problem determination API.

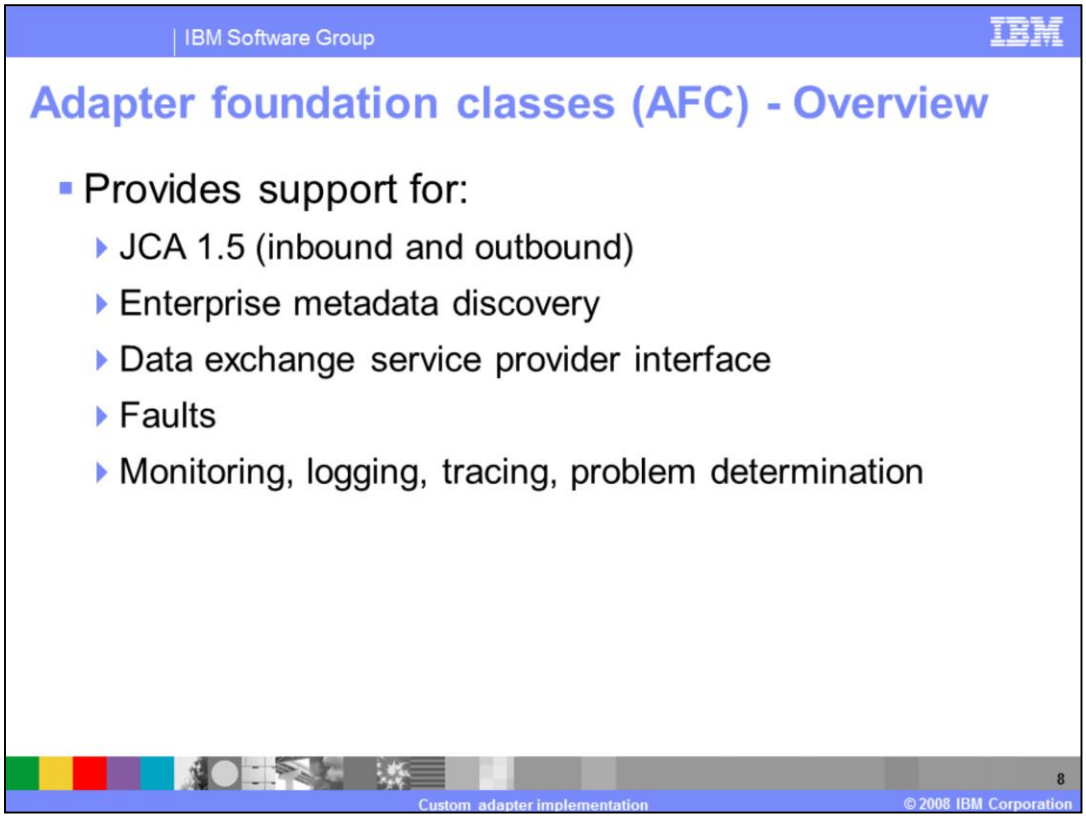


The goal of the base classes is to establish a standard for building resource adapters that conform to the Java™ 2 Connector Architecture 1.5 specification.

J2EE Connector Architecture defines a series of contracts that must be provided by a resource adapter. So anyone can develop a resource adapter without need of additional tools or support beyond what is provided by the J2EE SDK. However, repeatedly writing resource adapters from scratch is obviously not an efficient approach as implementation of many of the contracts defined by the connector architecture are similar regardless of the underlying enterprise information system (EIS). So developing resource adapters can be made more effective (and consistent) by identifying those areas of the JCA contracts that are generic and then providing a means for developers to take advantage of this common logic.

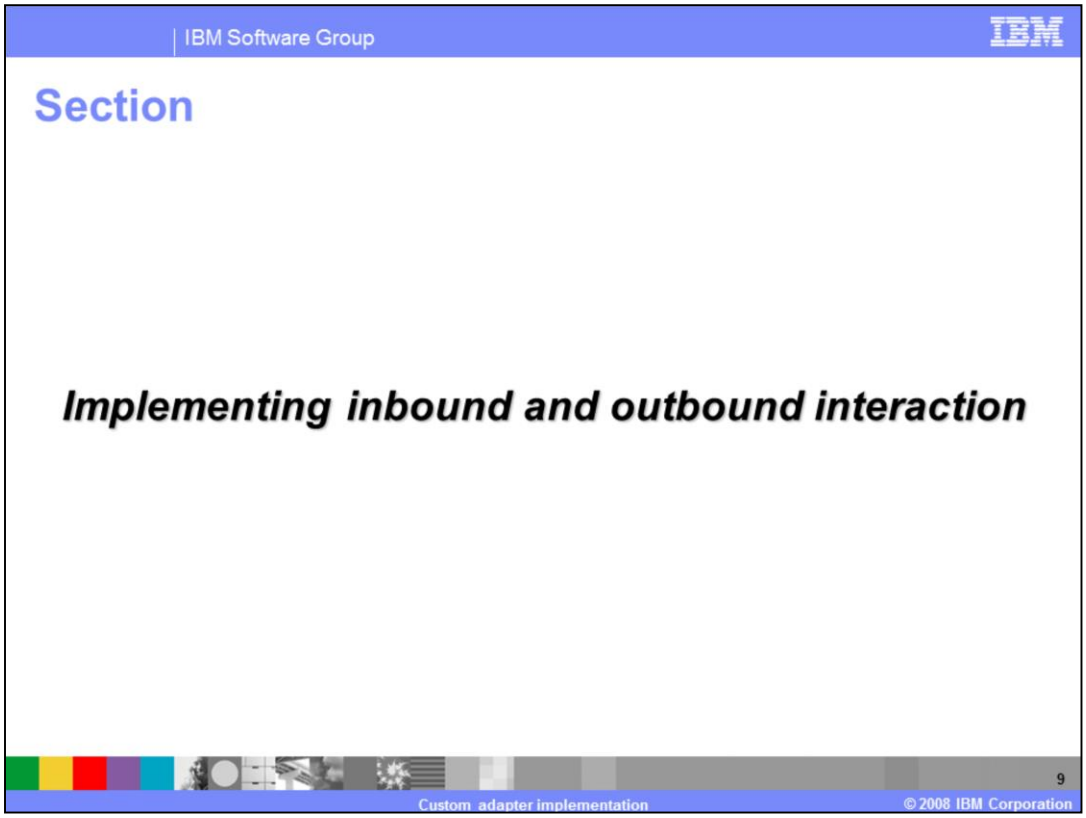
The chosen approach was to provide a set of abstract “base” classes that can be extended by developers. This set of base classes provides implementations of those contracts and methods which can be done generically and can be re-used by more than one adapter. Any methods for which EIS-specific logic is required are left abstract and the individual adapter developer needs to provide implementation for those methods.

For most of the resource adapters, developers can implement the abstract methods and have a working resource adapter. For those resource adapter designs requiring functionality that is different or beyond that which is provided by the base implementation, developers are left free to override the methods of interest in the connector specification.

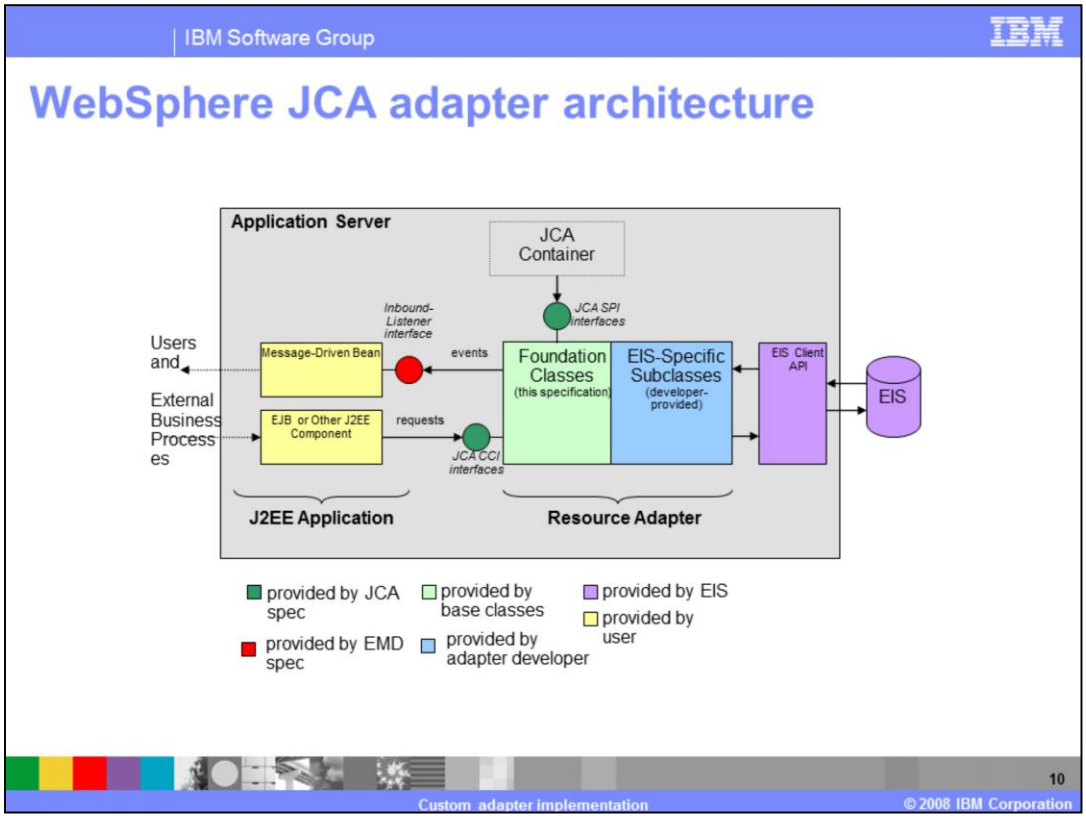


The base classes or the adapter foundation classes provide support for inbound, outbound and enterprise metadata discovery.

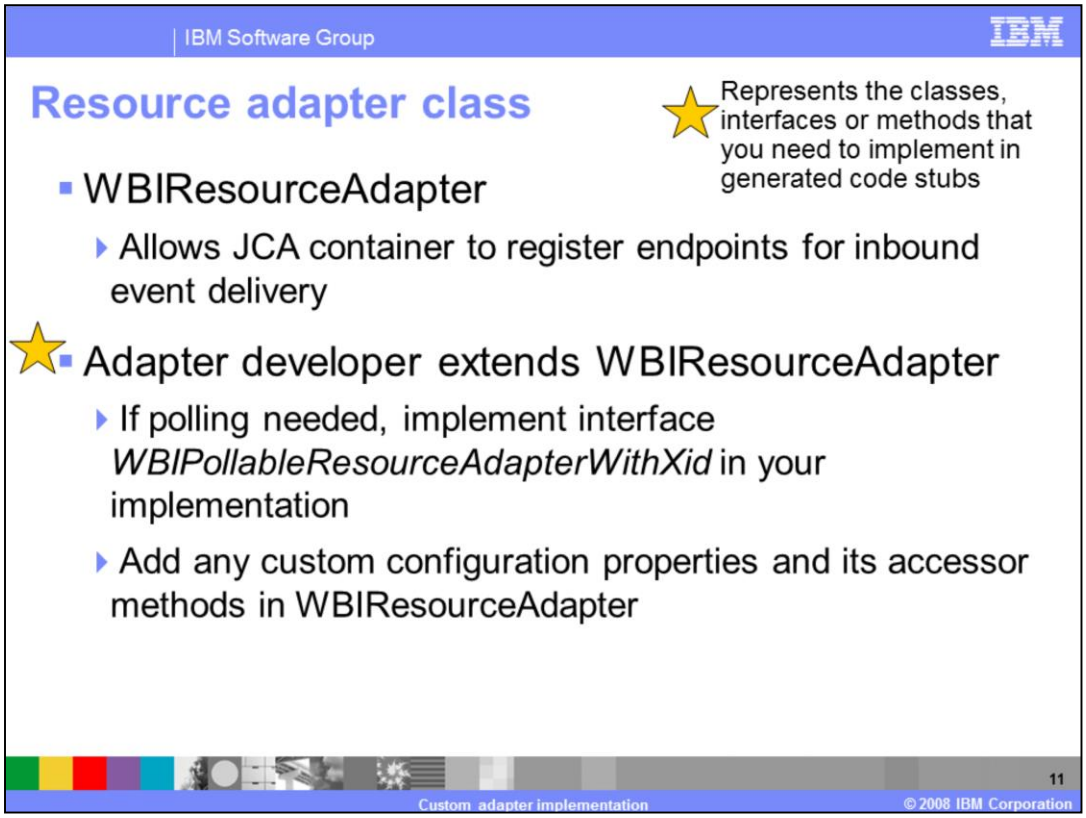
The JCA 1.5 specification is designed around EISs that provide 'push' models for event delivery where EIS calls the adapter when event is available and provides little support for polling models. To bridge this gap base classes provide pre-built event management logic that can be used by adapter developers. Outbound support involves sending service request to the EIS. Enterprise Metadata Discovery or a discovery service is a component within an adapter that enables the generation of business object definition and other artifacts required by SCA. Adapter foundation classes also provide a set of base faults, data exchange implementations for SDO, Java Bean, and much of the monitoring and problem determination capabilities that adapters need.



The next section goes into details of various classes and methods that you need to implement to support outbound and inbound interactions. The code stubs for these two interaction styles are created by using the J2C project wizard in WebSphere Integration Developer.



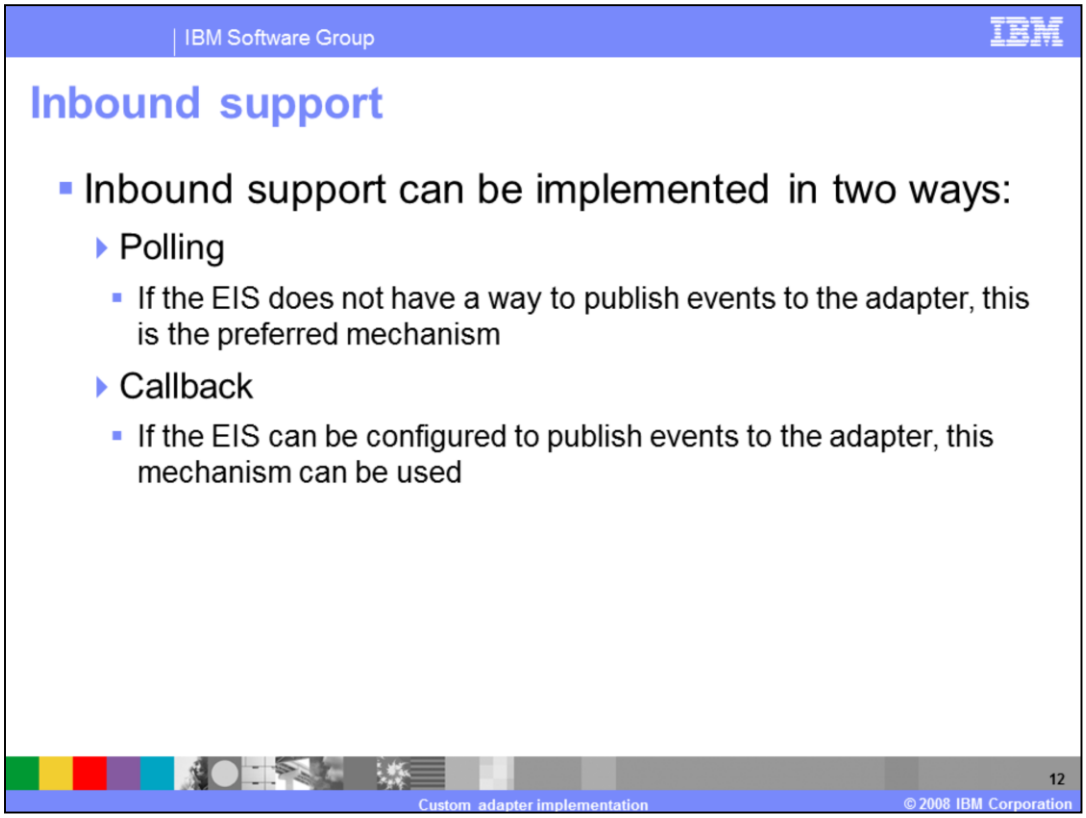
The adapter foundation classes in conjunction with the appropriate EIS-specific subclasses provide a JCA-compliant resource adapter implementation that can be managed by the application server to enable bi-directional connectivity to an EIS. Requests can be sent to the resource adapter by any J2EE component by making use of the Common Client Interface (CCI) defined by the JCA specification. For inbound, the adapter defines a “listener” interface and those message-driven beans that implement the interface are registered with the adapter enabling them to receive any appropriate inbound events from the EIS.



WBIResourceAdapter is a **javax.resource.spi.ResourceAdapter** instance and acts as a central authority for the state and instance-specific information about the adapter. It provides a means for the JCA container to register endpoints for inbound event delivery.

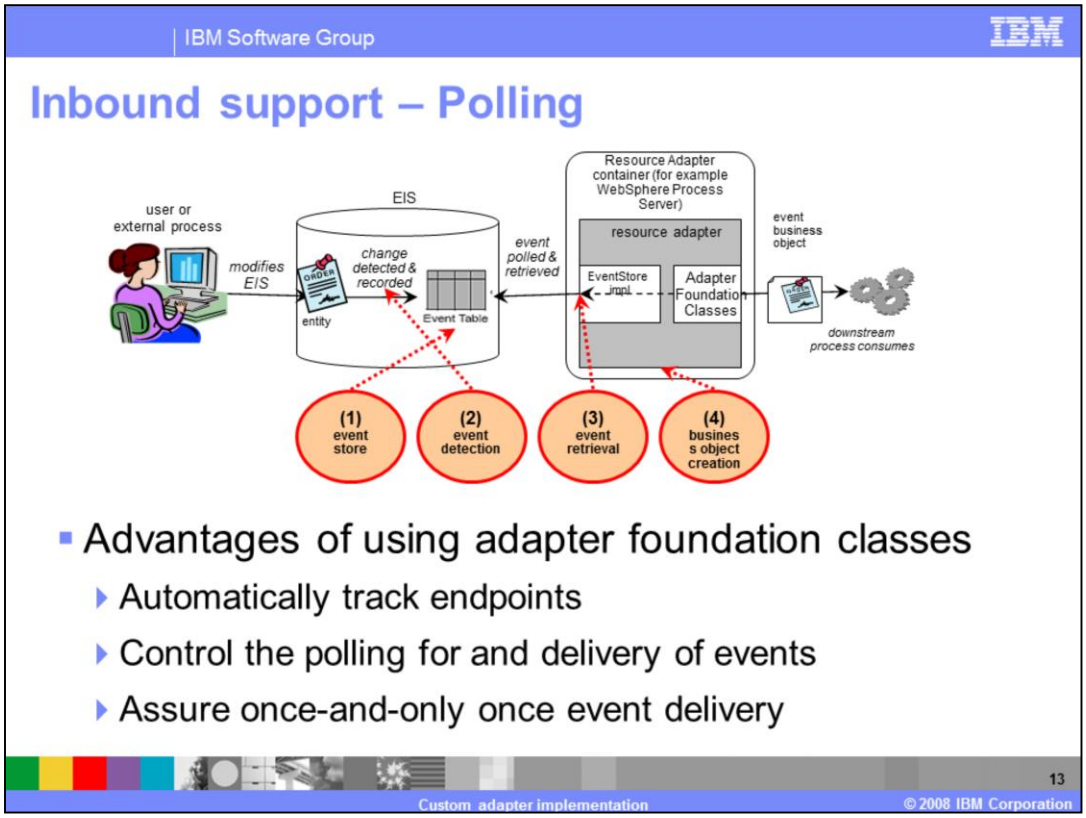
The **WBIResourceAdapter** implementation provides support for asynchronous event notification. This includes tracking of endpoints and activation and deactivation for inbound communication, creation and destruction of polling timers, and communication with the event manager to publish events from the EIS to endpoints. The implementation also includes generic Java bean logic for the tracking of property change listeners and firing of property change events when configuration properties are changed.

You need to extend the WBIReosurceAdpater class, implement the **start** method and the interface **WBIPollableResourceAdapter** if event polling quality of service (QOS) is needed. Implementing **WBIPollableResourceAdapter** interface requires you to implement the method **createEventStore.** You can add custom configuration properties specific to resource adapter. Any addition of properties to the ra.xml file using the wizard provided by the toolkit automatically generates the accessor methods in the appropriate code stubs.



Event polling or callback are the two ways to implement inbound support. Polling is the most common because most EISs don't have a way to publish events directly to the adapter. In this implementation, the adapter is in control. When polling, the adapter foundation class's event manager component provides the timer threads, the publishing mechanism, assured delivery algorithms, and so on. You need to provide implementation for Event Store interface.

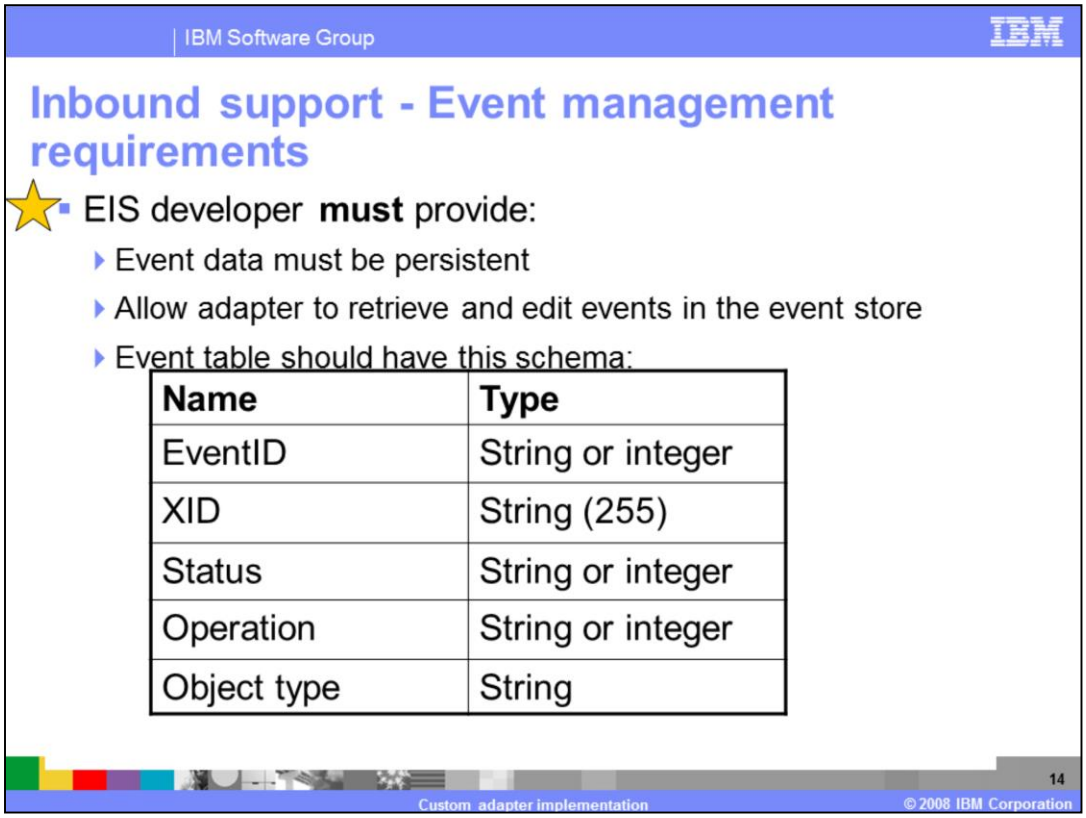
Certain EISs can publish events to the adapter directly. Callback option is generally preferred if the EIS provides this capability. Note, however, that the adapter will lose control over how many events will arrive and it might be necessary to add some throttling or provide some custom way to assure delivery.



The role of event-notification is to enable business processes to be notified of changes or new information about a given EIS.

As depicted in the diagram in the slide, user or external process modifies the EIS. Event detection mechanism implemented in the EIS detects any changes of interest in the EIS and records them in the event store. An event retrieval mechanism (event store interface) is implemented in the adapter that can detect and retrieve events from the event store. A data transformation mechanism is implemented in the adapter to convert the EIS events to WebSphere Business Integration business objects consumable by target business processes.

Use of the adapter foundation classes can dramatically simplify the often complicated implementation of event retrieval and publication. Adapter foundation classes will automatically track endpoints for the adapter, control the polling for and delivery of events, and handle recovery of any events if the adapter unexpectedly terminates and assures once-and-only once event delivery.

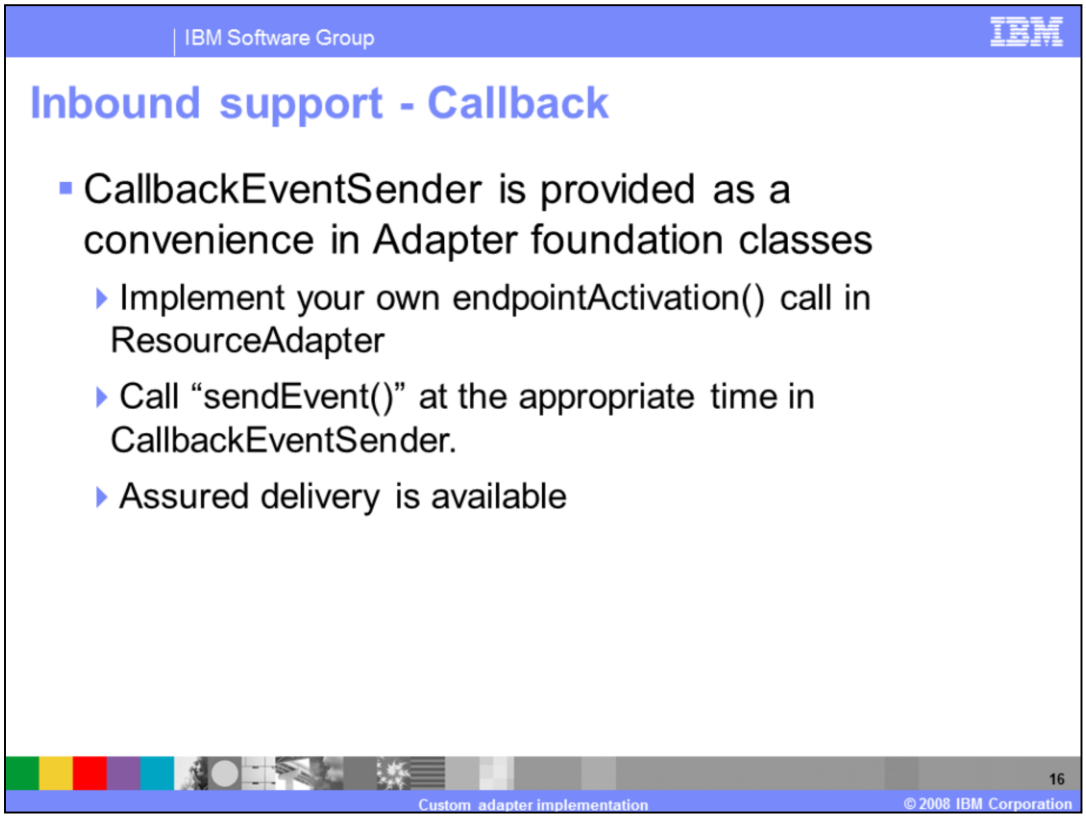


In order to use adapter foundation classes for inbound event management, there are some requirements on how the events should be maintained on the EIS side. Event information should be stored in an event store. The event store can be a database or any other mechanism that you want to use to store the information.

Event data must be persistent. An event detected in the event store should remain available in the event store until deleted by the adapter regardless of connection failure or time elapsed. The event store must allow adapter to both identify and change the state of event records in the event store. The event store must be able to store and modify a transaction ID (XID.)

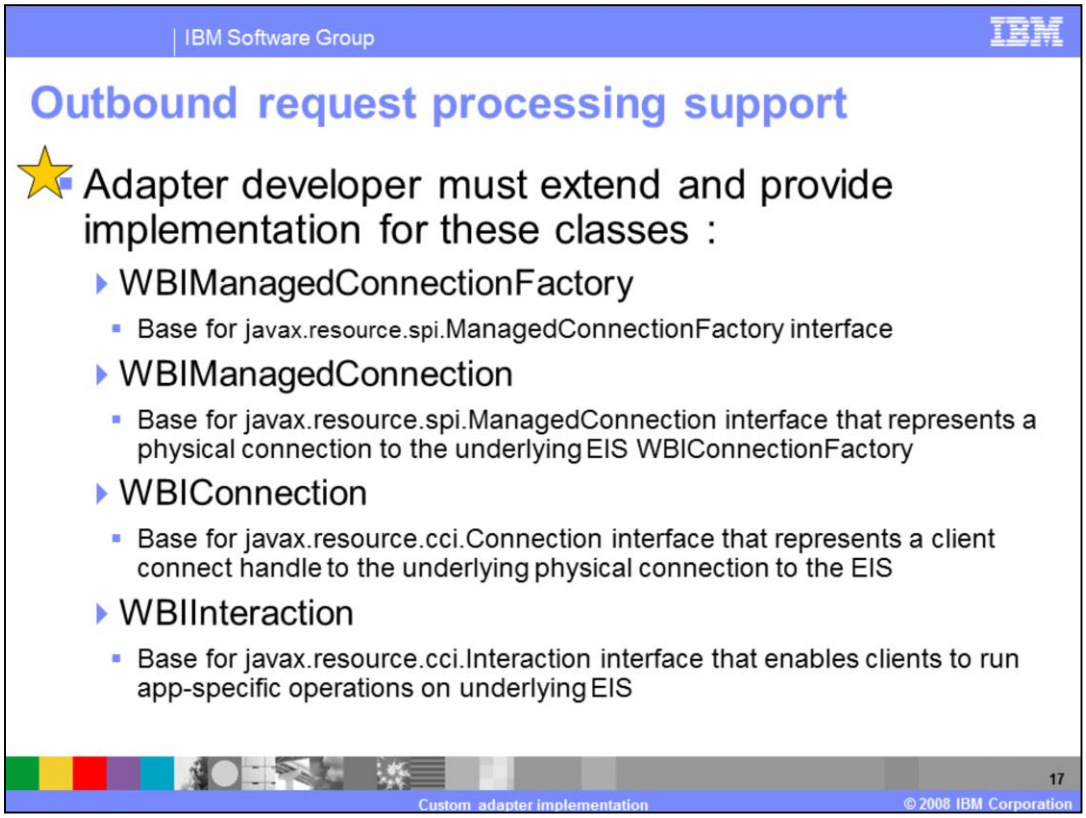
If you want to take advantage of the functionality provided by the adapter foundation classes (AFC) for inbound, you need to ensure that any subclass of ***com.ibm.j2ca.base.WBIResourceAdapter*** implements interface ***com.ibm.j2ca.WBIPollableResourceAdapter***. This interface allows the adapter foundation classes to acquire an event store implementation specific to the underlying EIS application. If this interface is implemented, the AFC will automatically begin checking for and publishing events as dictated by polling-related configuration properties like “PollPeriod” and “PollQuantity” as specified by active adapter endpoints.

You also need to implement the ***com.ibm.j2ca.extensions.eventmanagement.EventStore*** interface. This interface allows the AFC to manage events in the store without requiring specific knowledge of how and where the event store is implemented.



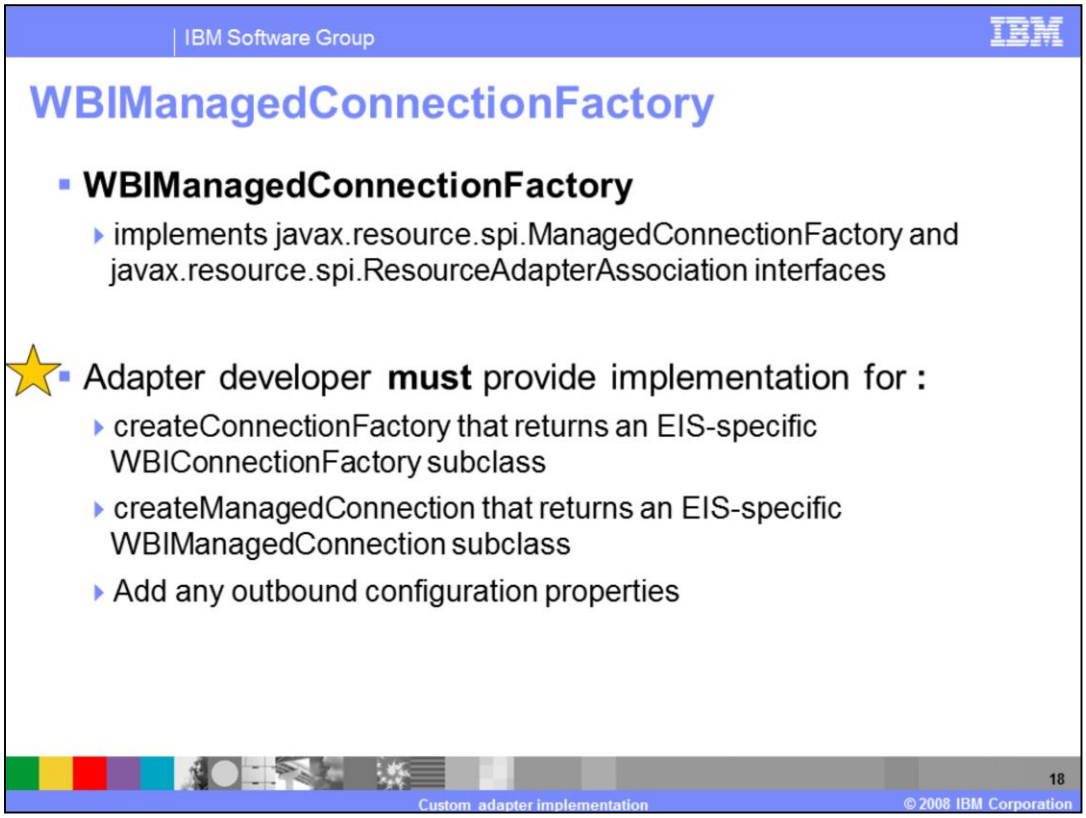
The CallbackEventSender is an API used to provide some help for the “callback” method of implementing inbound support. The adapter is responsible for setting up any necessary threads, connecting at the appropriate time, and managing “endpoints” (the message endpoint factory and activation spec pair passed in at endpointActivation.)

Assured delivery is supported by passing in an EventStore implementation to keep track of which events have been committed.



This slide shows the list of classes that you need to provide implementation for supporting outbound calls. You must extend the **WBIManagedConnectionFactory** and **WBIManagedConnection** base classes which implement the service provider interfaces **managedconnectionfactory** and **managedconnection**.

You must extend the **WBIConnection**, **WBIInteraction** base classes which implement the **cci** interfaces **connection** and **interaction**.



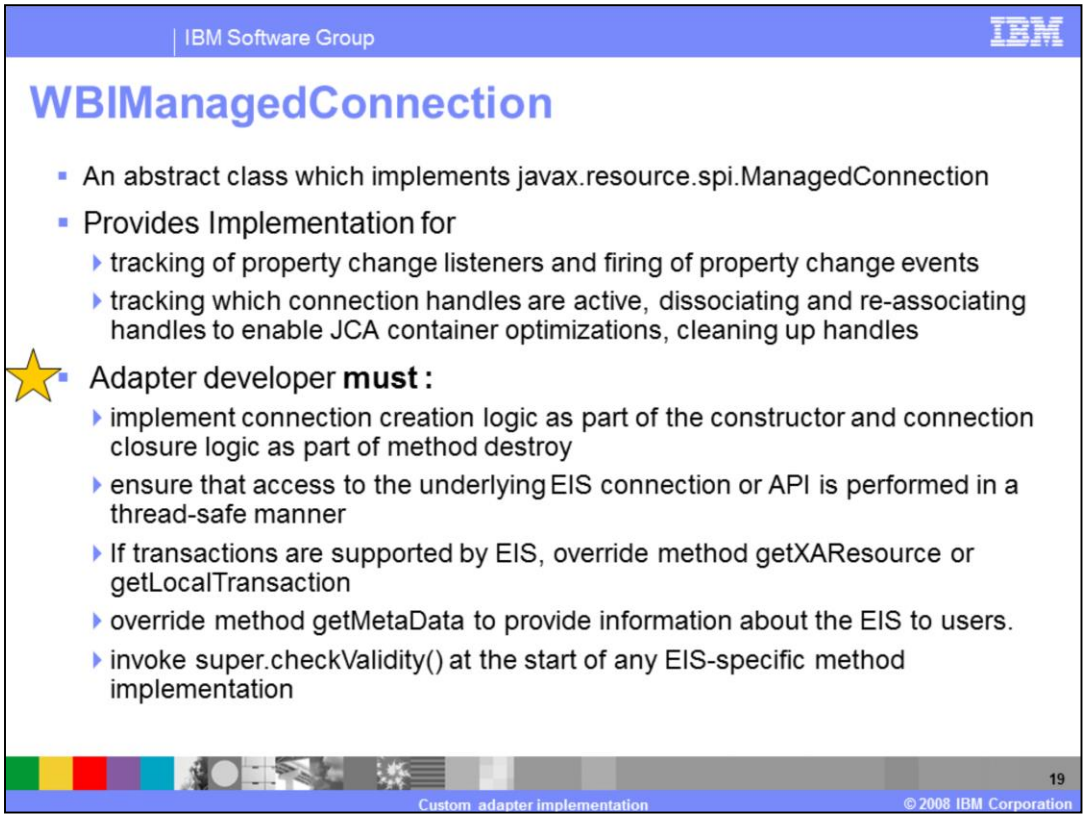
A managed connection factory is used for the container-managed configuration and creation of physical connections to the underlying EIS.

WBIManagedConnectionFactory Implements the service provider interfaces **javax.resource.spi.ManagedConnectionFactory** and **javax.resource.spi.ResourceAdapterAssiociation**.

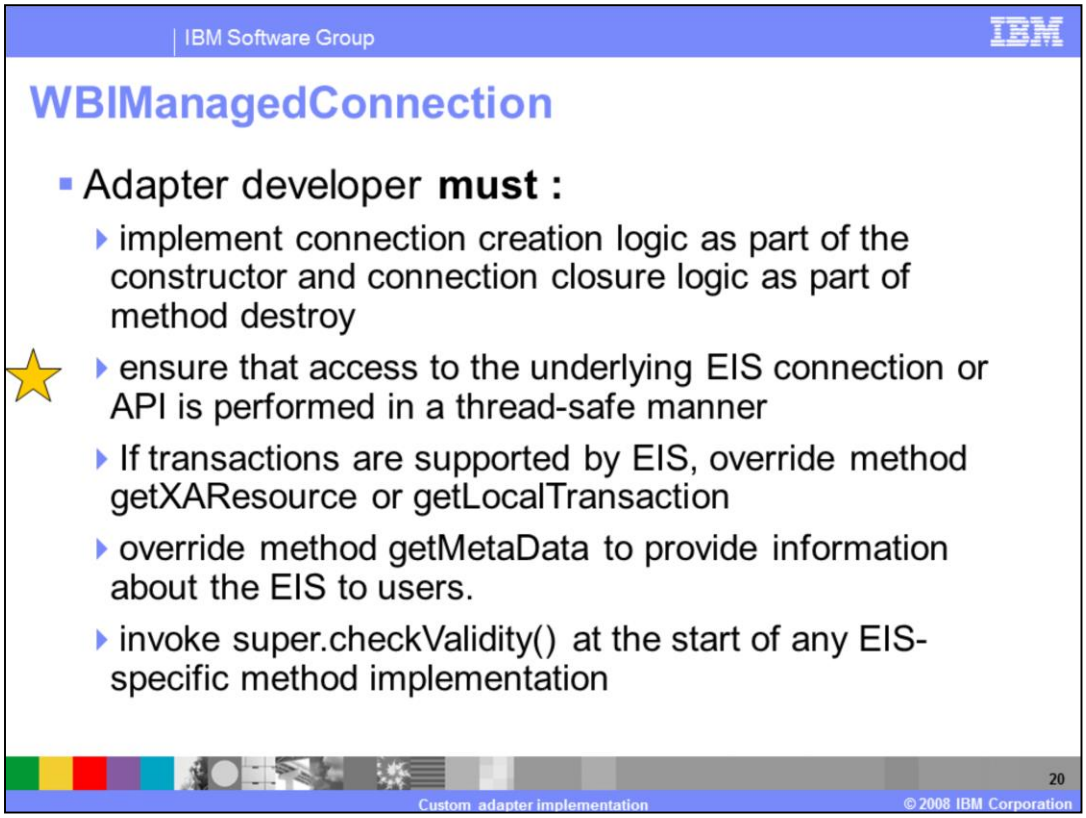
You must provide implementation for the methods **createConnectionFactory** and **createManagedConnection**.

For any custom EIS-specific outbound configuration properties defined for this class, you need to provide the appropriate get and set methods. If the resource adapter deployment descriptor editor in the toolkit is used to add these properties, the tool automatically generates the appropriate code stubs in the managed connection factory subclass.

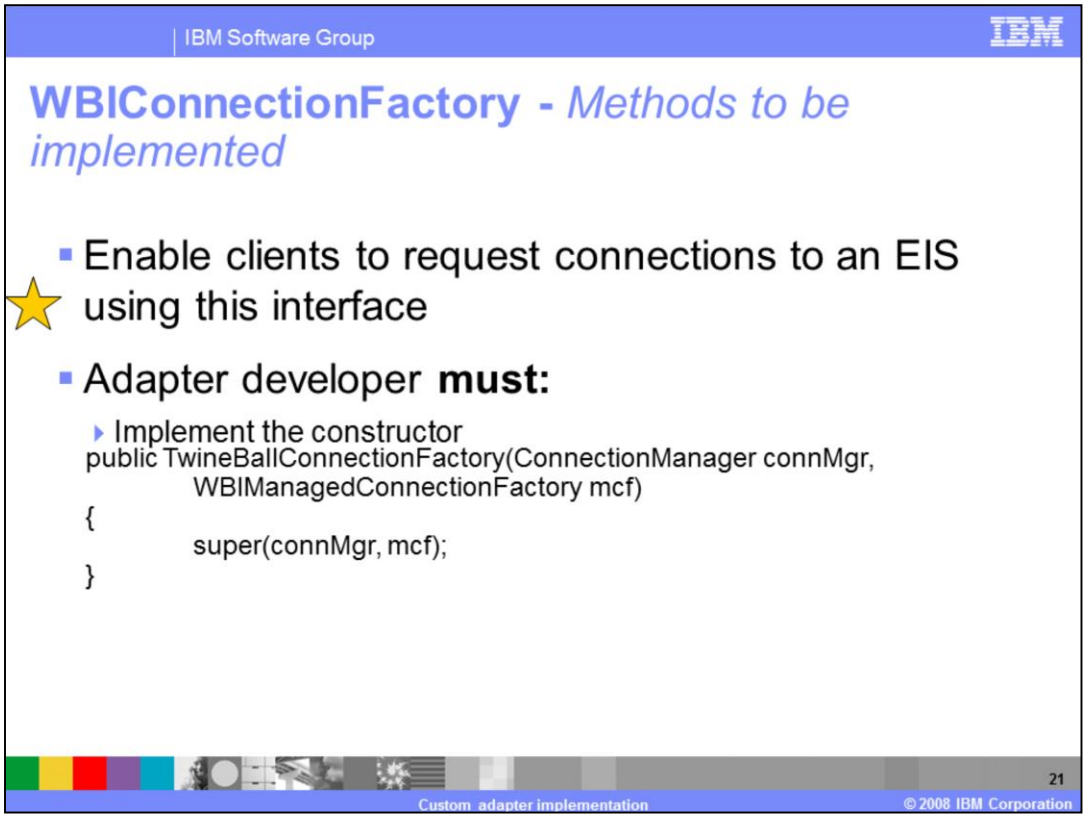
Properties defined in this class are generally intended for use by the **WBIManagedConnection** implementation when connecting to the EIS.



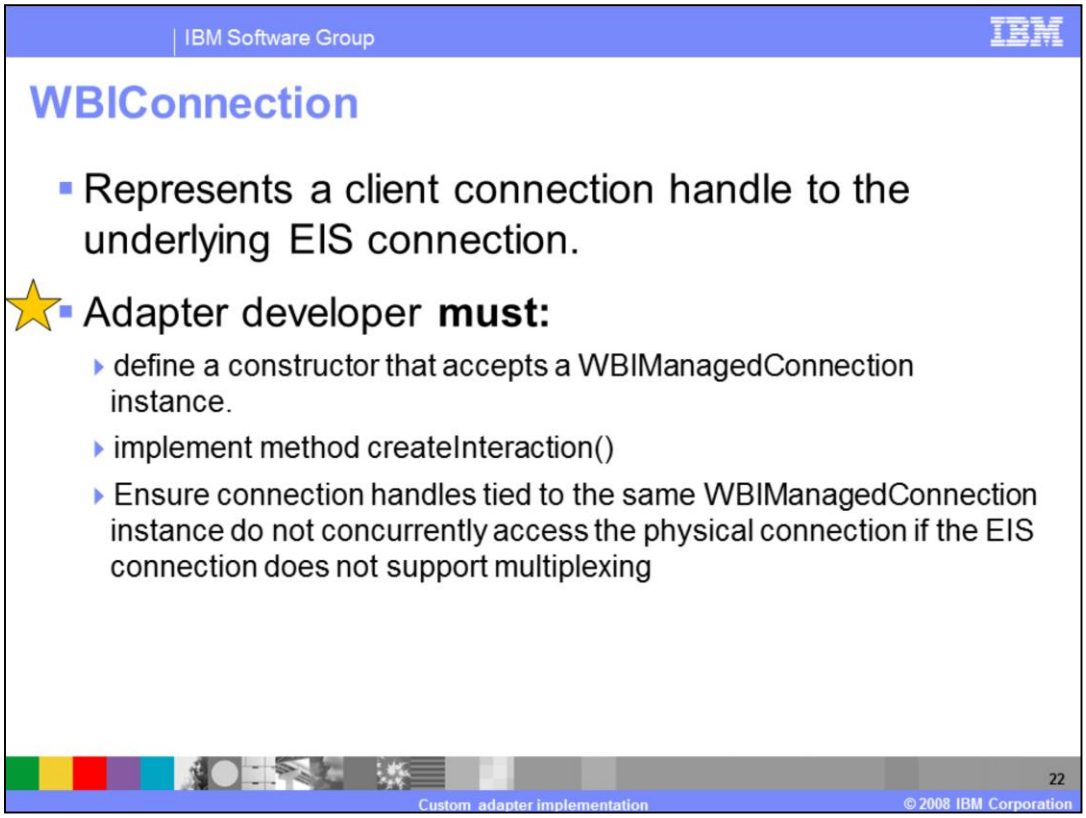
ManagedConnection instance represents a physical connection to the underlying EIS. You need to provide logic for connecting to the EIS. ManagedConnection instance provides generic logic for the tracking of property change listeners and firing of property change events when configuration properties are changed. It provides support for managed connection optimization as described in JCA specification. ManagedConnection instance provides private communication with WBIConnection instances providing support for tracking active connection handles, dissociating and re-associating handles, cleaning up handles and keeping the JCA container informed of connection handles no longer used by a client. It also provides support for tracking of connection event listeners and notifying them of connection failures.



You should implement connection creation logic as part of the constructor and connection closure logic as part of method destroy. Each instance should encapsulate at most one connection to the EIS. Since there can be more than one connection instance for each **ManagedConnection** instance, you should implement private contracts between their **WBIManagedConnection** subclass and their **WBIConnection** and **WBIInteraction** subclasses to ensure thread-safe access to the underlying EIS connection or API. If transactions-either XA or local are supported by the EIS, you should override method **getXAResource** or **getLocalTransaction** . Do not invoke the super implementations of these methods as the foundation classes throw exceptions. You should override method **getMetaData** to provide information about the EIS to users. At the start of any EIS-specific method implementation, you should always invoke **super.checkValidity();** this method checks the state of the **ManagedConnection** instance to ensure that it hasn't been closed, encountered an error, and so on..

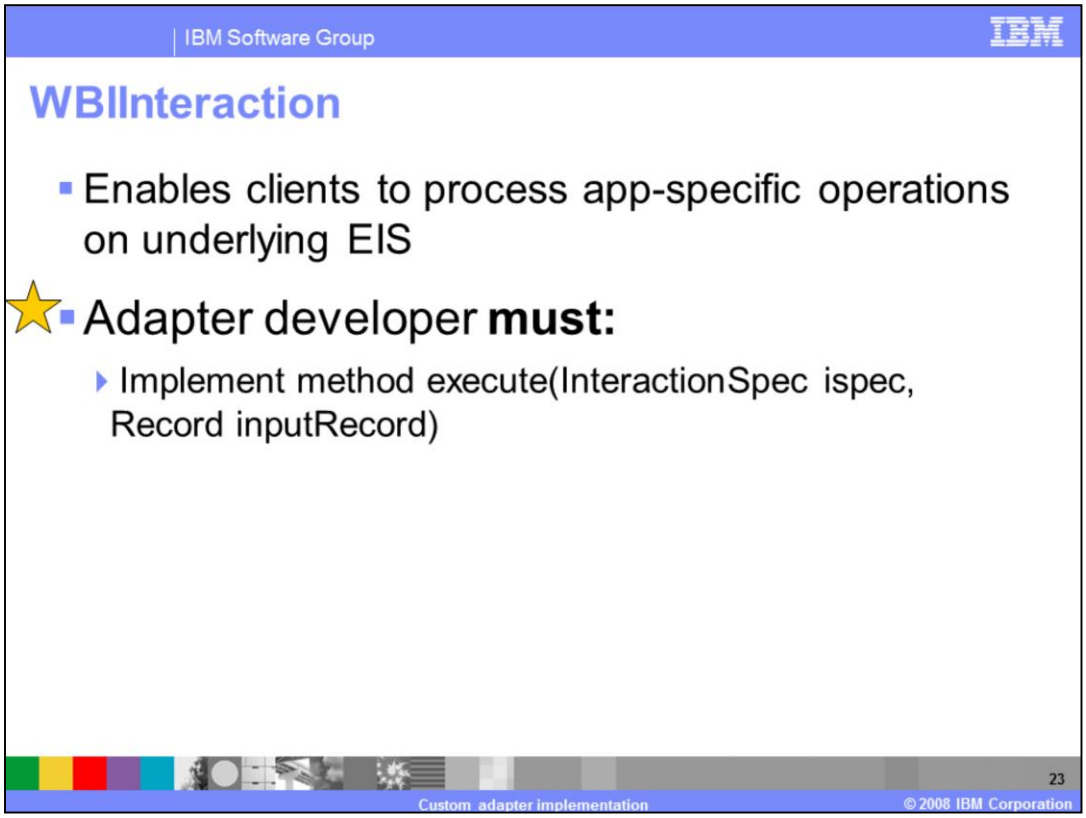


WBIConnectionFactory is a **ConnectionFactory** that enables a client to request handles to the underlying EIS connection. You must implement the constructor. **GetConnection** method can be used to return a new connection handle to the underlying EIS.



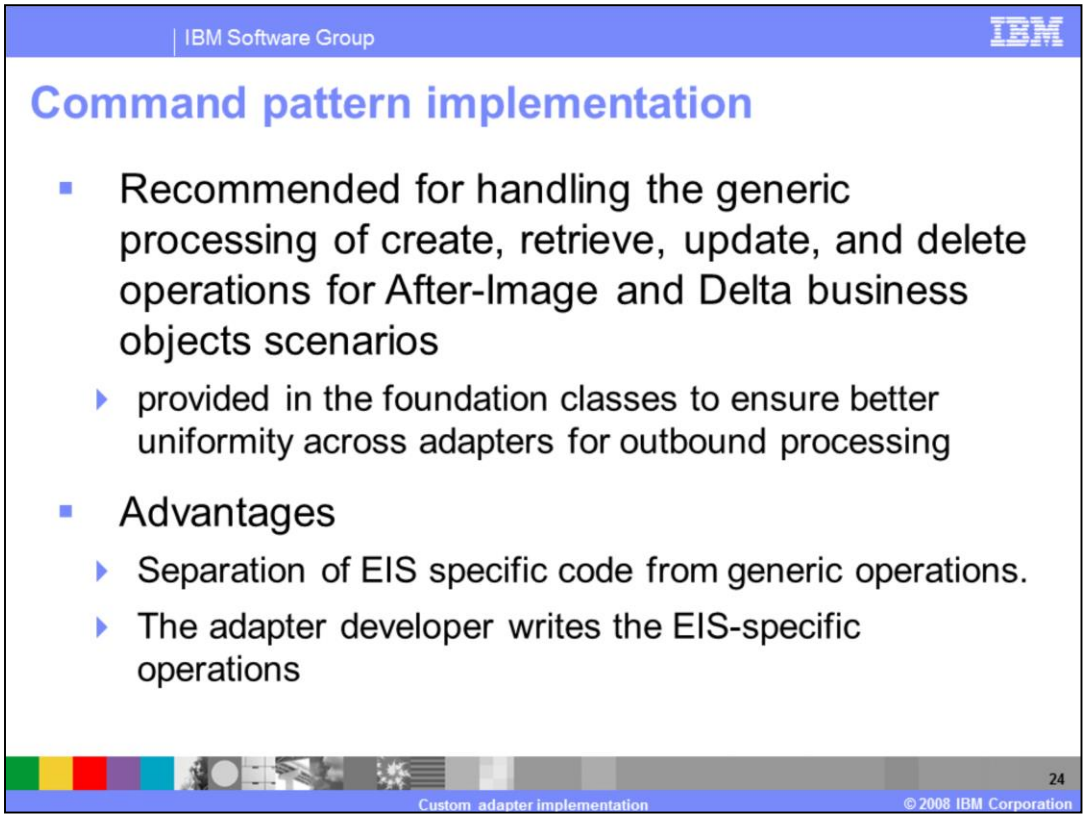
WBIConnection represents a client connection handle to the underlying EIS connection. Client gets this connection handle by calling the **getConnection** method of the **ConnectionFactory** instance.

You must define a constructor that accepts a WBIManagedConnection instance. Once constructed with a WBIManagedConnection instance, the WBIConnection will register itself with the managed connection and start communicating events. You must also implement method **createInteraction** and provide an EIS-specific Interaction instance so that clients can invoke functions on the underlying EIS. Since **WBIManagedConnection** instance can have multiple connection instances, you should make sure that that the connection handles do not concurrently access the physical connection if the EIS connection does not support multiplexing. Always invoke **getManagedConnection()** to locate the **WBIManagedConnection** instance with this connection handle before trying to access either the managed connection or the underlying EIS.



WBIInteraction instance enables clients to process app-specific operations on underlying EIS. Most of the implementation of this interface is EIS-specific. You must provide implementation for **execute** method and provide logic to call an EIS operation represented by the InteractionSpec and return an output Record.

The client and adapter exchange data using the record model. WebSphere Business Integration adapters use the WebSphere Business Integration business object model which is not compatible. So the base classes will provide a record implementation that wraps the WebSphere Business Integration business object to allow it to be passed between clients and the resource adapter implementation.



For systems that support a typical create, retrieve, update, and delete pattern of operations, adapters are responsible for carrying out these operations in the EIS based on the structure and contents of the incoming business object.

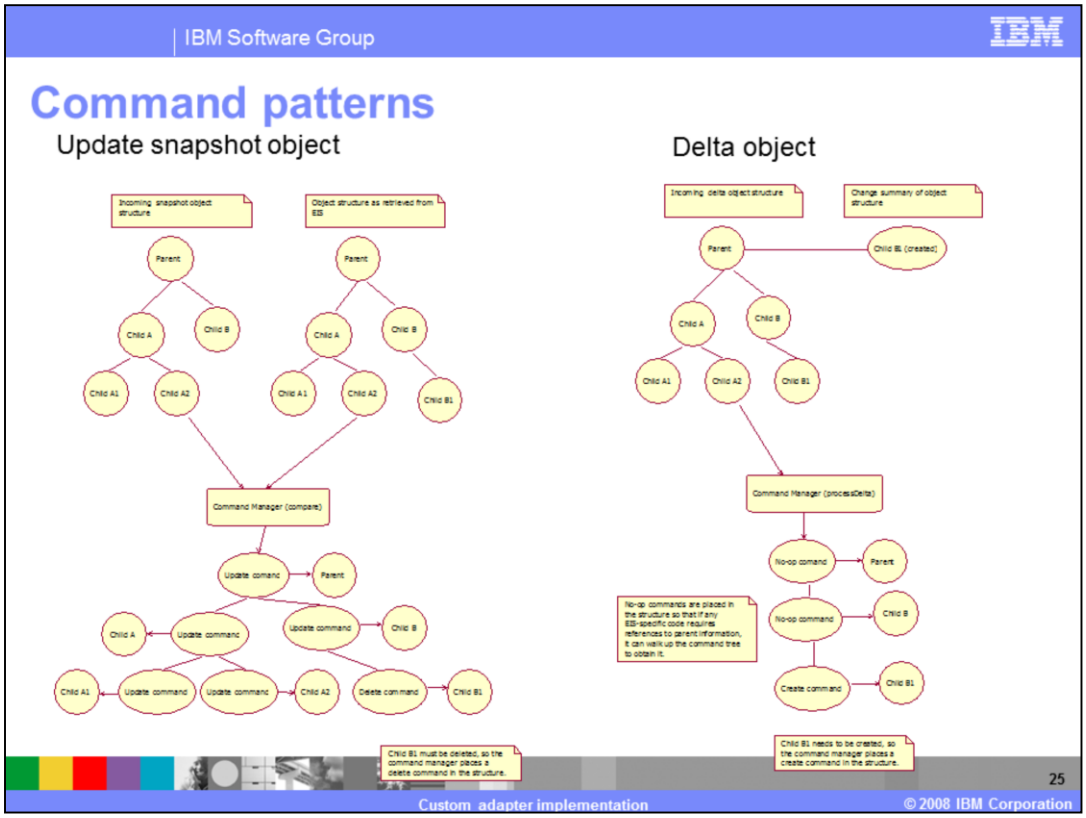
For example, if you have an incoming business object that represents an after-image update, the adapter must take steps to update the EIS such that the object in the EIS matches the structure and contents of the incoming business object.

To accomplish this, the adapter has to retrieve the structure currently in the EIS, compare it to the incoming SDO and perform the operations necessary to make the EIS match the input. These operations are to be done as the comparisons happen. This makes an adapter potentially quite complex. The command pattern abstracts this functionality into a generic logic

The command pattern breaks down a hierarchical update into a hierarchy of small sub-commands. These sub commands are fed to an interpreter, which retrieves and executes the code necessary to do the sub-command.

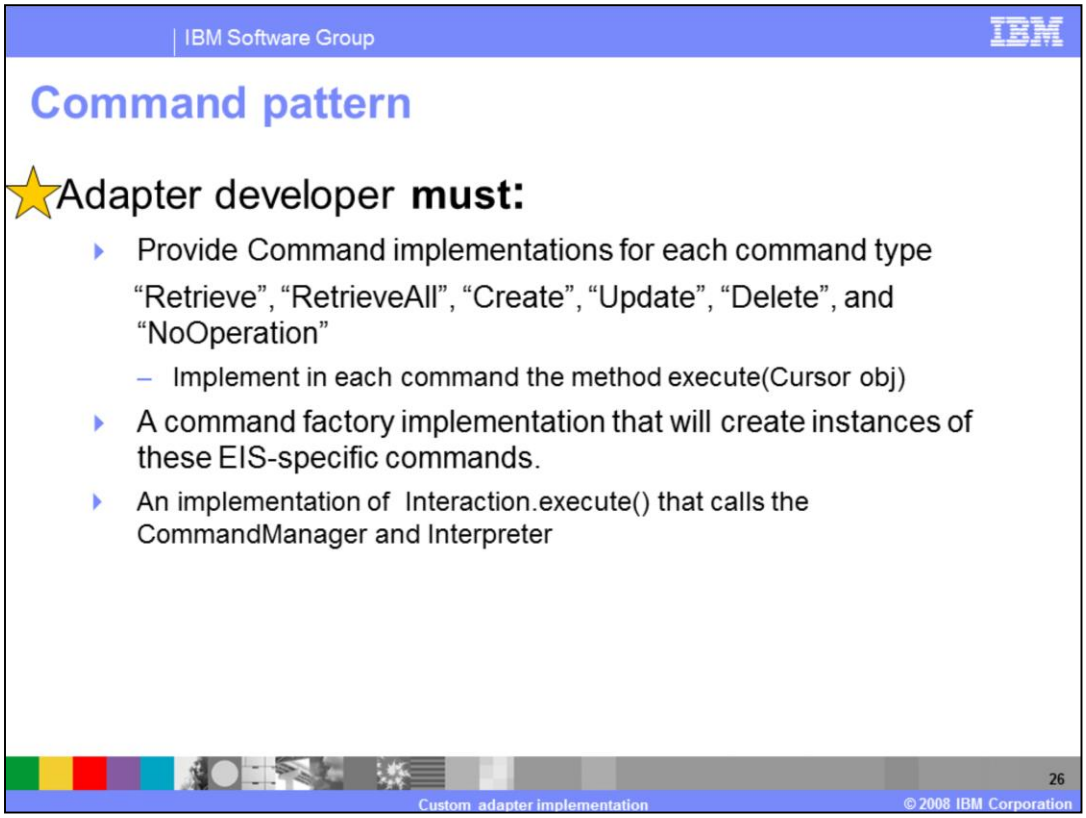
This makes it possible for you to deal with operations on node level entities, without having

to walk the structure and compare.



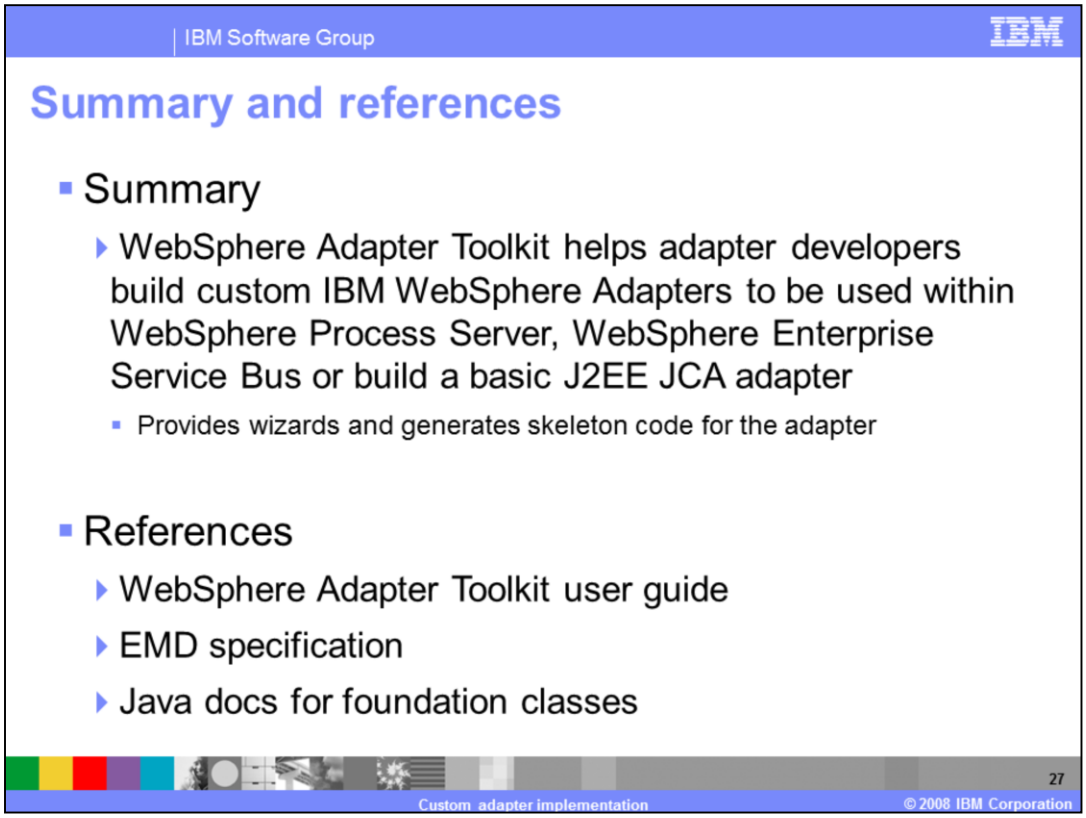
For snapshot object update, normally you have to retrieve the object from EIS and compare the input business object to EIS business object at each node and make changes as you compare. But with use of commandmanager and interpreter, you can just create sequence of commands and interpreter executes them. This separates EIS logic from generic operations.

The same logic works for delta objects. You can look at the change summary and make use of command manager to generate the commands. No-op commands are used where there are no entries in change summary.



You have to provide implementation for the create update, delete, retrieve, retireveall and no-op commands. You need to provide a command factory implementation that will create instances of these EIS-specific commands. You also need to provide implementation for Interaction.execute() method that calls the CommandManager and Interpreter in your WBIInteraction subclass.

You have an abstract base command for your EIS, and have the operation-specific commands extend that. This way, if all your commands need similar data, you can reduce your coding effort. WebSphere Adapter Toolkit creates stubs for only abstract methods. The result is that the BaseCommand stub has an execute() method and the subclasses do not have this method. Therefore you have to manually create the execute() method in the subclasses.



WebSphere Adapter Toolkit helps adapter developers build custom IBM WebSphere Adapters to be used within WebSphere Process Server, WebSphere Enterprise Service Bus or build a basic J2EE JCA adapter. The underlying adapter foundation classes used by the toolkit simplify the process of adapter development by providing implementation for most generic contracts so you only provide the implementation logic for your backend EIS.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

EJB, J2EE, Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.