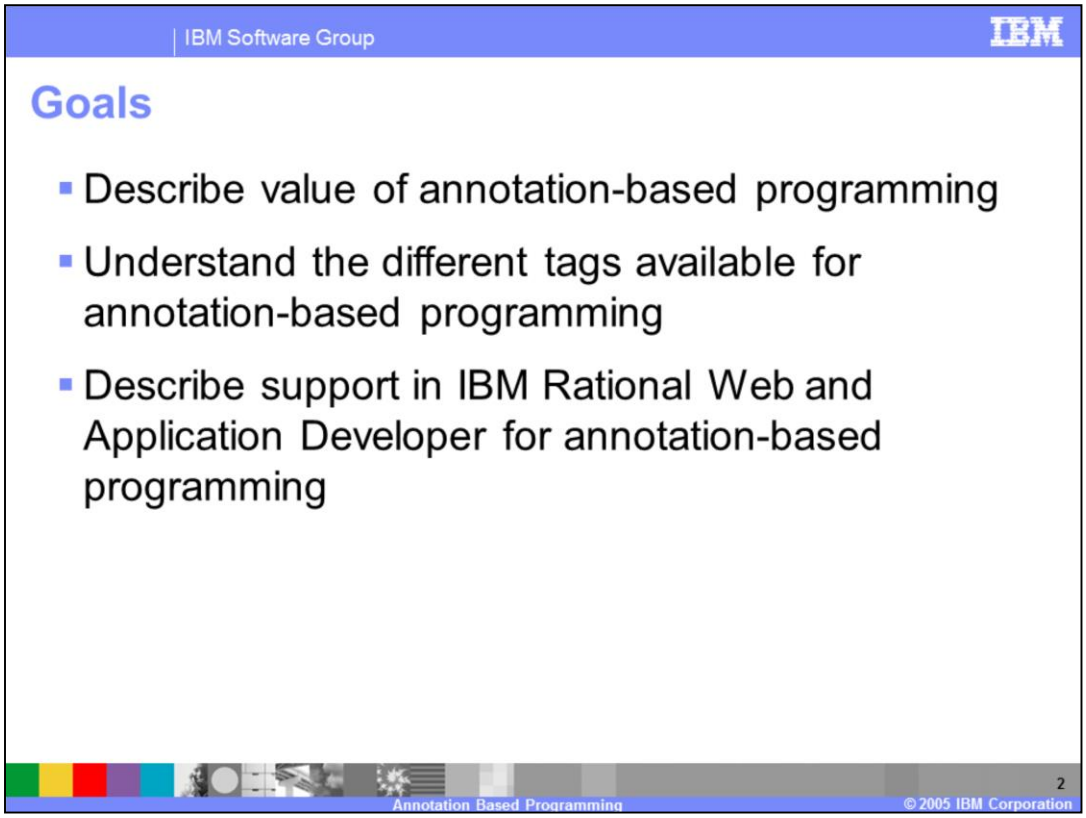
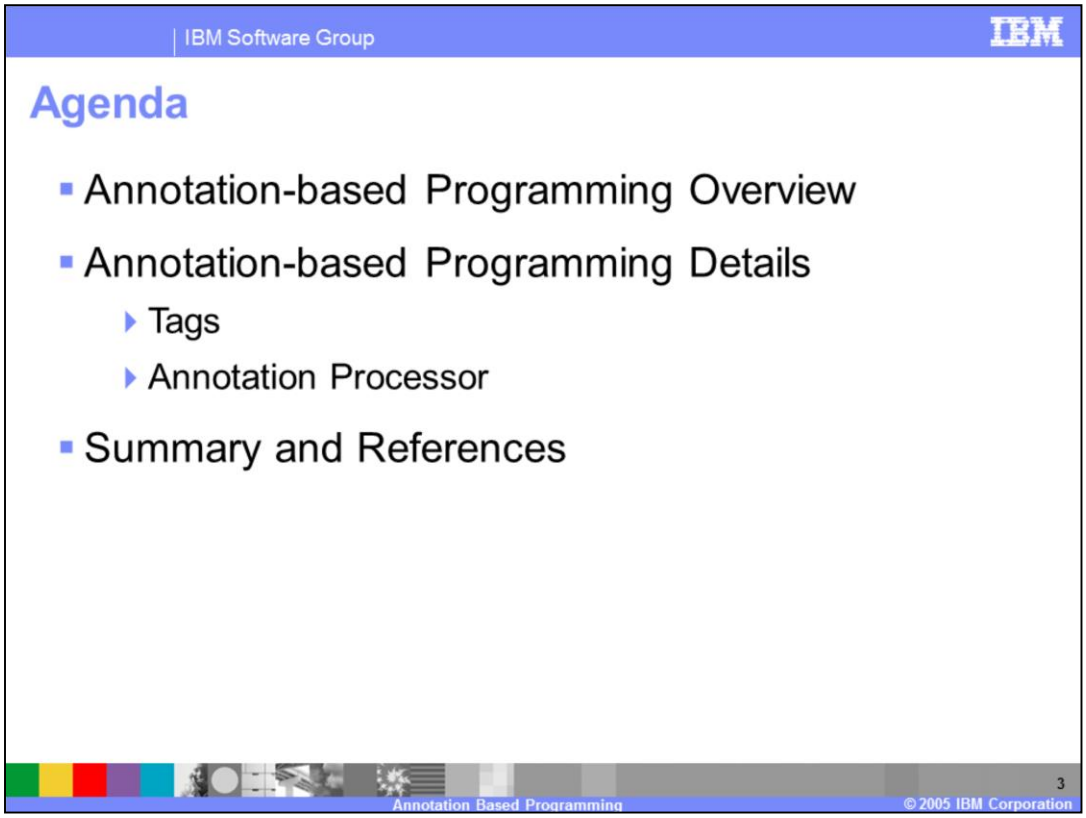
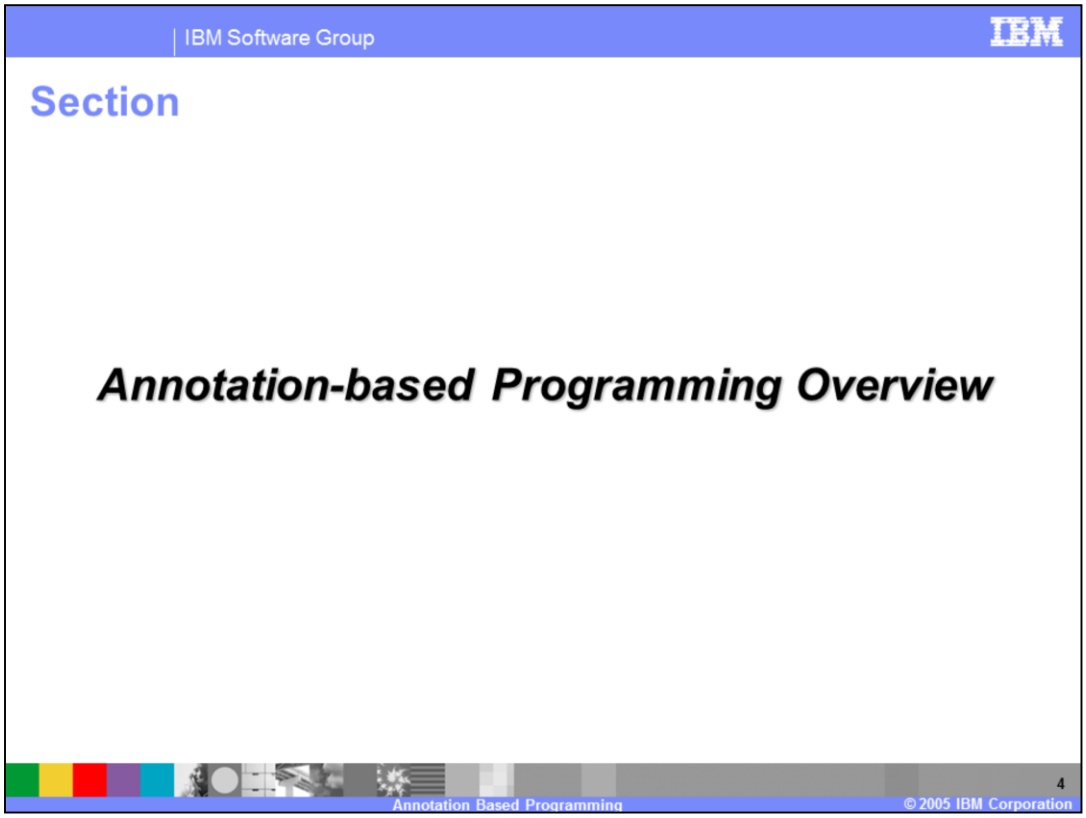

This presentation will focus on an overview of the annotation-based programming support included with IBM Rational® Web Developer and IBM Rational Application Developer. This new feature allows developers an easier and faster way to develop Java™ 2 Enterprise Edition (J2EE) applications.



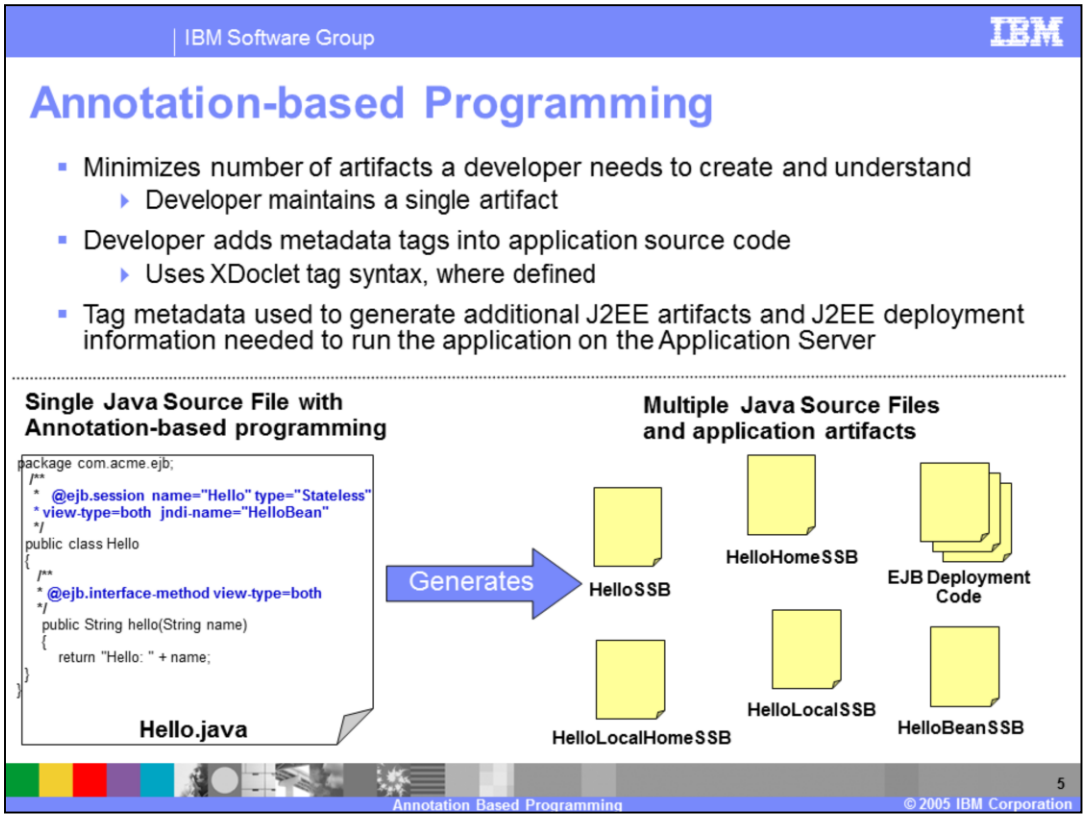

The main goals of this presentation start by describing the value of annotation-based programming for developers building J2EE applications. Describing the different tags which are used in annotation-based programming is also a goal. The final goal is to understand the level of support for annotation-based programming within IBM Rational Web Developer and IBM Rational Application Developer.



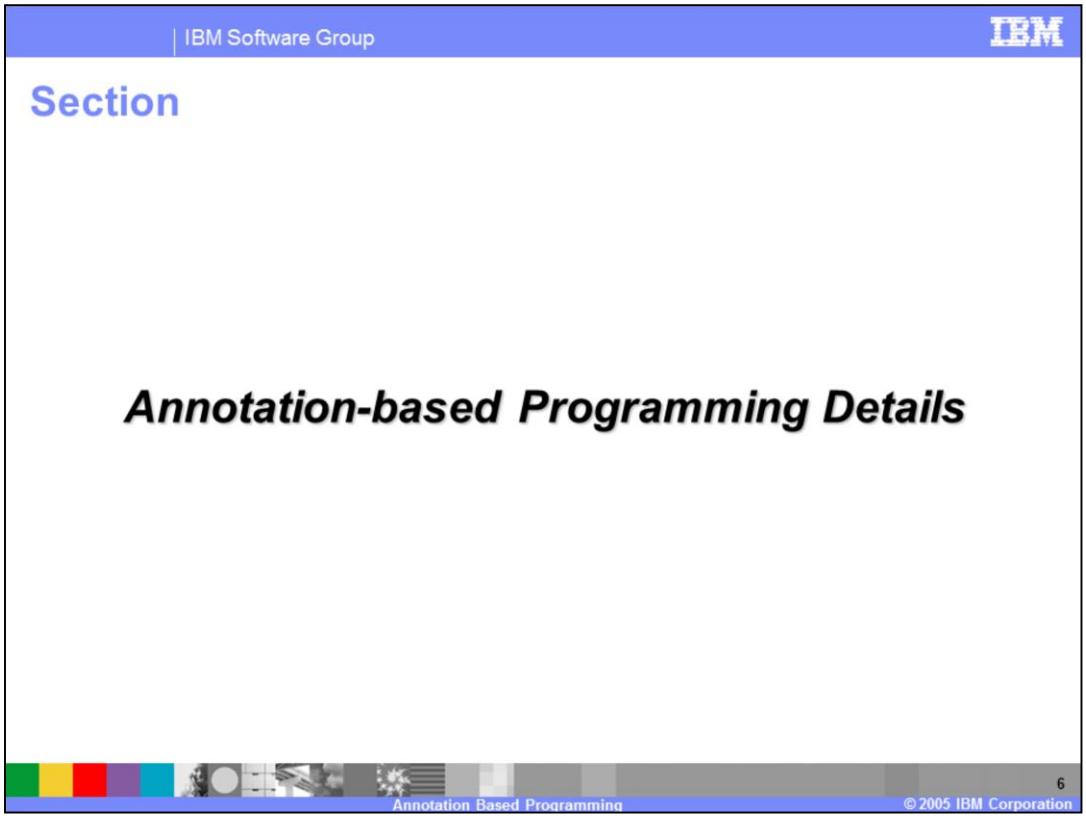

This presentation will cover the features of annotation-based programming. In this presentation, an overview of annotation-based programming and the details behind it will be covered, specifically looking at the tags which are supported and the annotation processor which generates the appropriate code and deployment information based on the annotations.



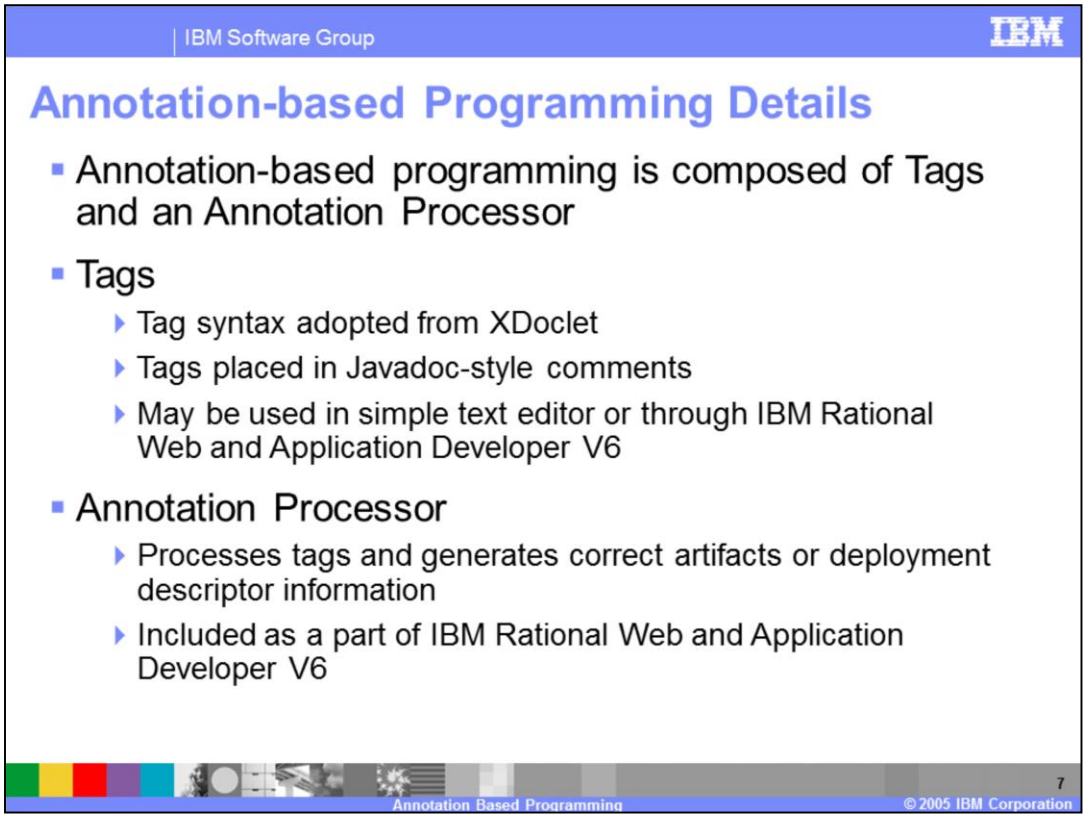

This section will provide an overview of annotation-based programming.




Annotation-based programming (ABP) is the technology of allowing the developer to add additional metadata into the source code of their application and then using that additional metadata to derive the artifacts necessary to run the application in a J2EE environment. The goal of annotation-based programming is to minimize the number of artifacts that the developer has to create, maintain, and understand, thereby simplifying the development experience. For example, consider a stateless session Enterprise JavaBean. With annotation-based programming, the developer would simply create a single Java source file containing the bean implementation logic and indications as to which methods should be made public on the interfaces of the Enterprise JavaBean. A few additional tags would be needed to indicate the desire to deploy the class as an EJB . With this single artifact containing tags, the home and remote interface classes can be created along with a stateless session implementation wrapper class. Deployment information can also be created in the ejb-jar.xml deployment descriptor as well as WebSphere-specific binding data and all of the remaining artifacts necessary to produce a compliant J2EE application. All the developer has to deal with is the single Java artifact.

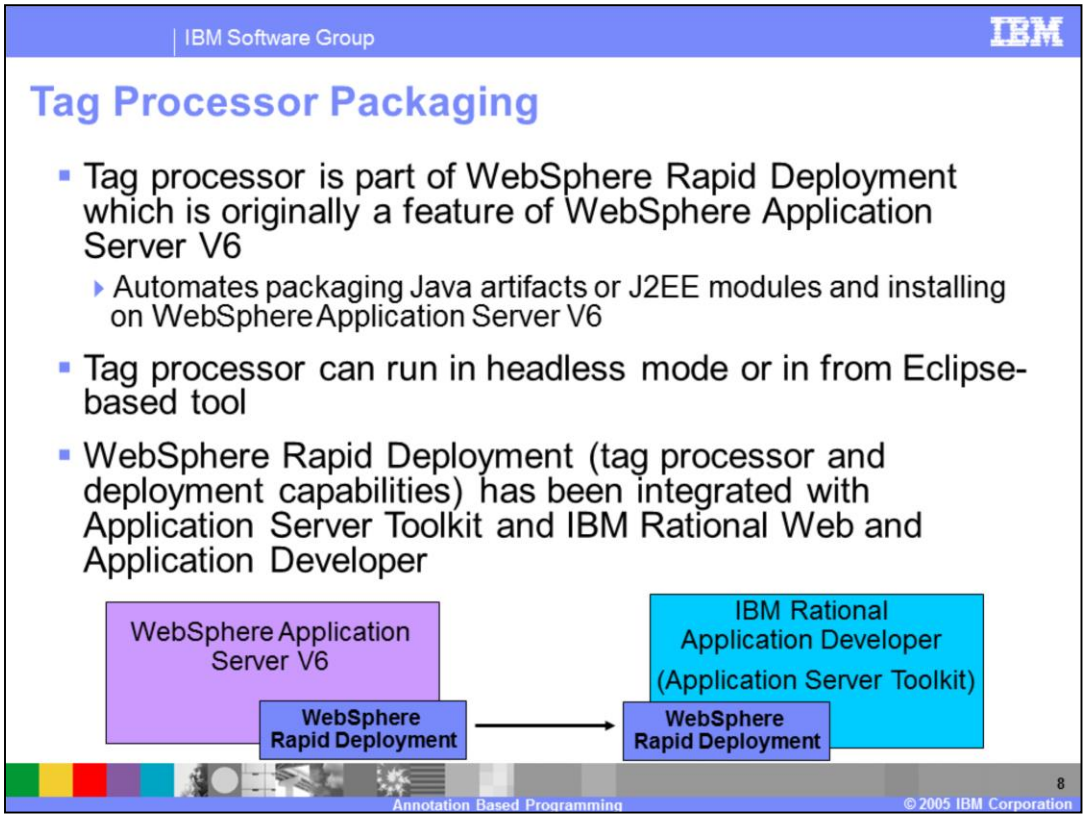



This section will discuss the details of annotation-based programming.




Annotation-based programming is comprised of two components. The first component is the actual tags which a developer may use. These tags follow the syntax as defined by the XDoclet open source project. The tags are placed in Javadoc-style comments and are easily recognizable as related to annotation-based programming. The artifacts which contain the tags may be created with a simple text editor, however working with the tags in a development environment such as IBM Rational Web Developer or IBM Rational Application Developer is much more productive. The different artifacts and deployment information can be quickly generate as part of the build process without any additional steps by the developer. The second component of annotation-based programming is the annotation processor that provides the mechanics of processing the tags and generating the correct artifacts and deployment information. IBM Rational Web Developer or IBM Rational Application Developer contain a annotation processor for processing the tags.

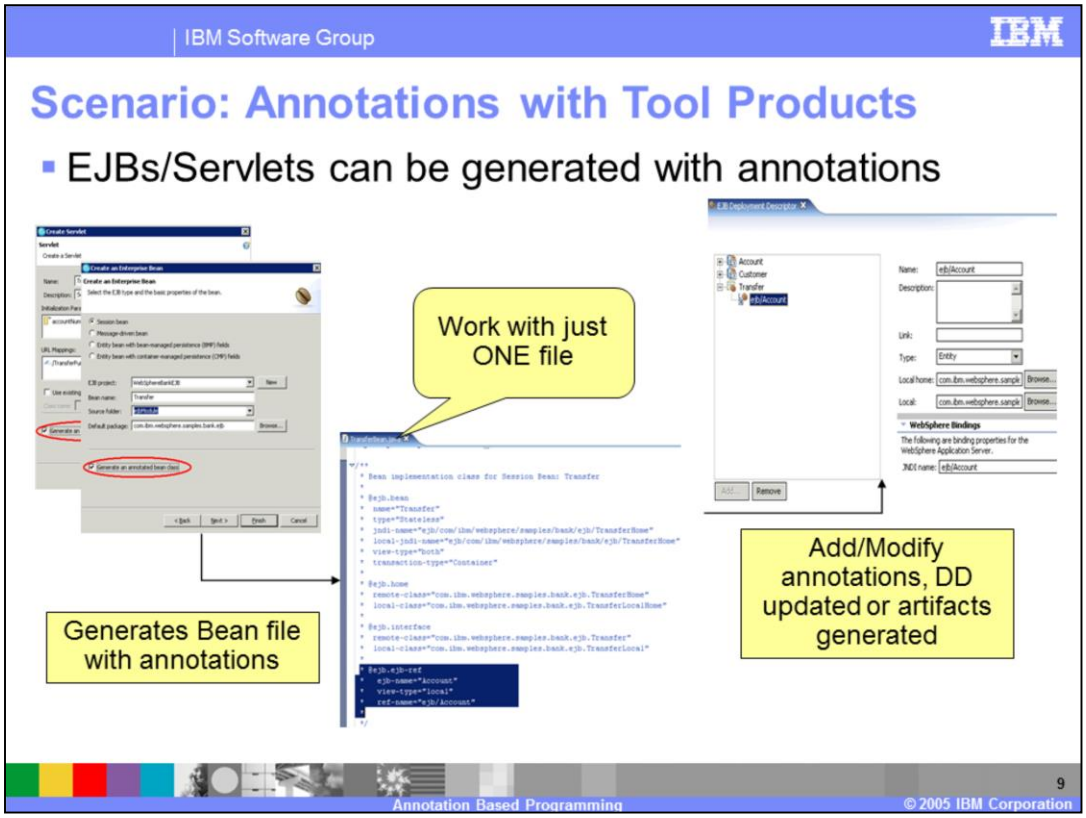
Annotations take the form of Javadoc-style comments in the Java source file. The annotation is entered using an @-tag in the comment block of the source code. Annotations can be placed on class, field, or method declarations.



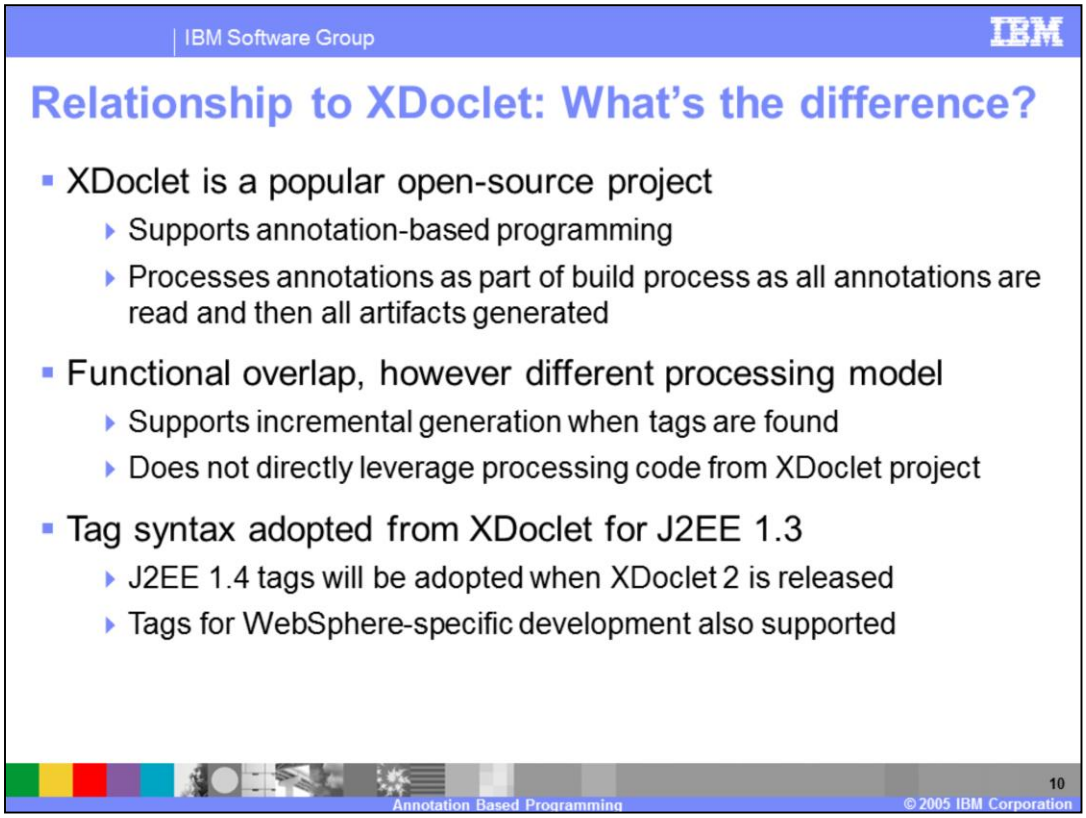

The annotation tag processor included with IBM Rational Web Developer and IBM Rational Application Developer is actually a feature of the WebSphere Rapid Deployment which is part of WebSphere Application Server V6.0. The WebSphere Rapid Deployment feature is designed to ease the deployment and installation of applications into WebSphere Application Server. With WebSphere Rapid Deployment, a directory can be setup where J2EE artifacts ,such as enterprise applications, Web modules , EJB JAR files, or individual Java files may be placed, and from there automatically packaged, deployed, and installed into WebSphere Application Server V6.0. As part of the packaging and deploying steps, if any artifact has annotation tags, the appropriate artifacts and deployment information must be generated first. WebSphere Rapid Deployment includes an annotation tag processor to support this scenario. The WebSphere Rapid Deployment feature is actually based on Eclipse and is executed in a headless manner (non-GUI) as applications are installed packaged, deployed, and installed into WebSphere Application Server V6.0. Because it is already based on Eclipse, it is easily included with the Application Server Toolkit and also IBM Rational Web Developer and IBM Rational Application Developer.

WebSphere Rapid Deployment does not include any development support for using the tags, although the tags can be specified with a simple text editor. IBM Rational Web Developer and IBM Rational Application Developer provide tight integration in the form of content assist to help developers specify the correct tags as well as wizards which will

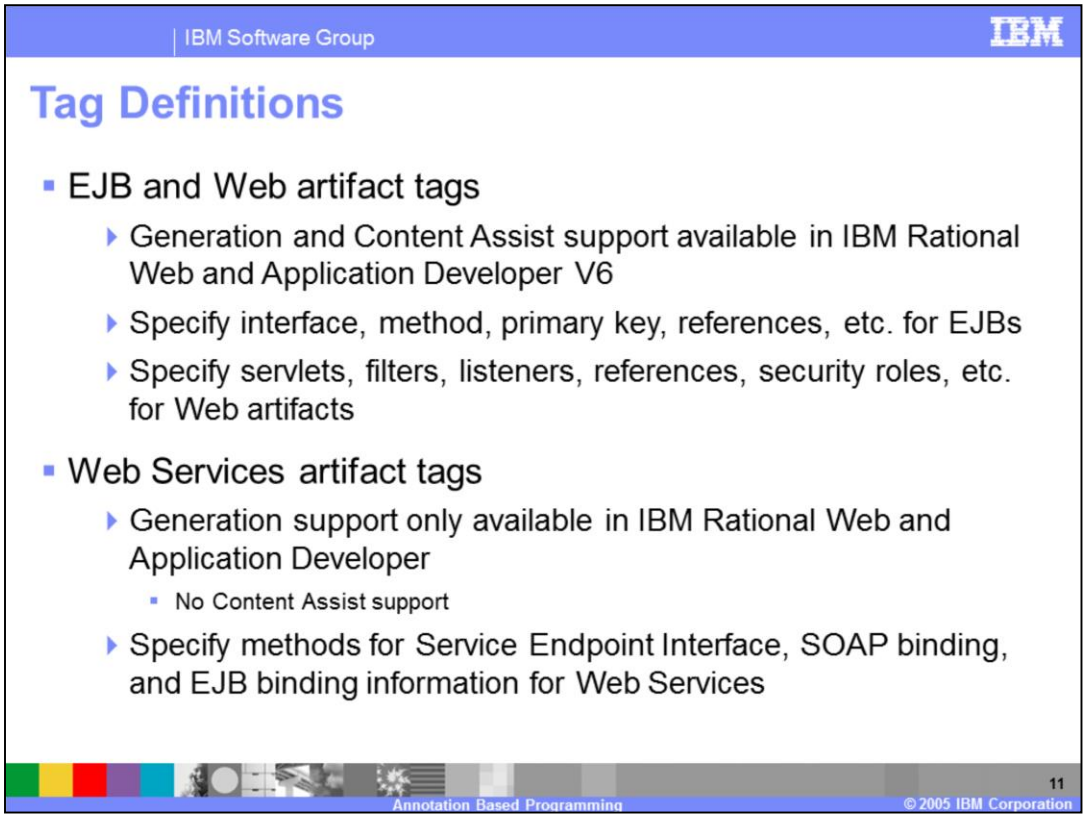
generate artifacts with tags.



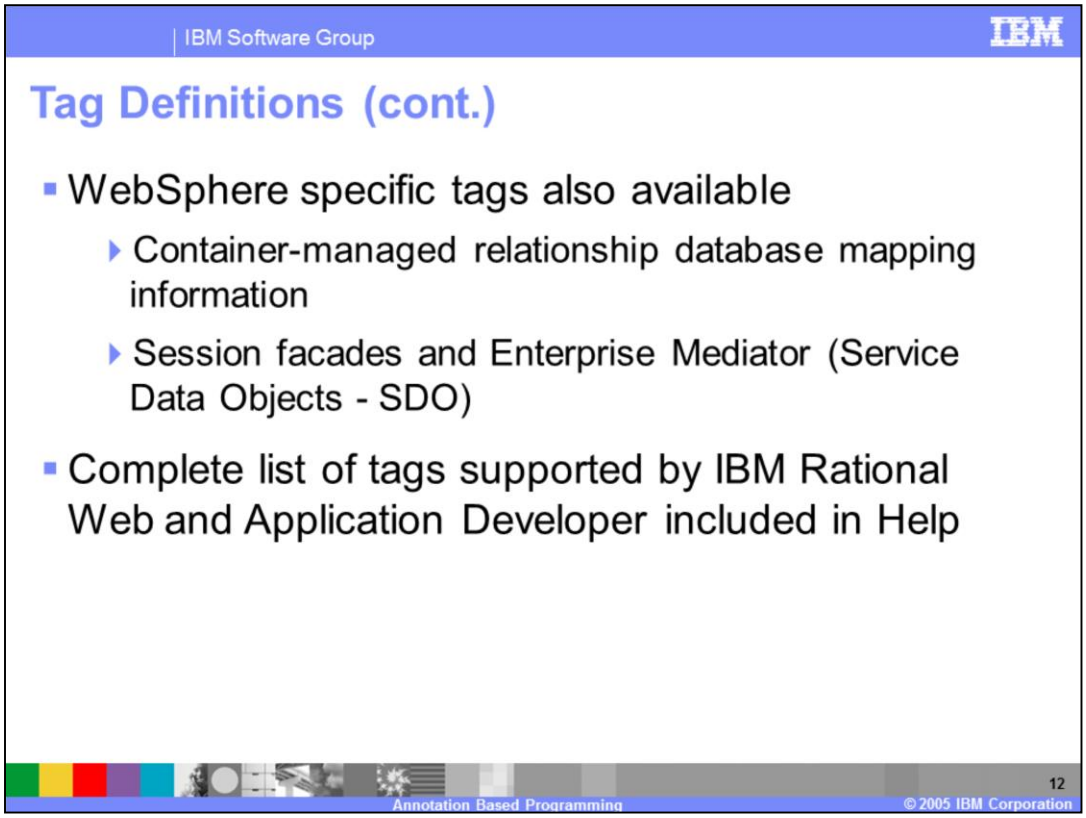

When you generate either a servlet or Enterprise JavaBean, you now have the option to generate with annotation tags. This allows a developer to work with one source file for a specific artifact. In this scenario, using the IBM Rational Application Developer EJB creation wizard, you create a Session Enterprise JavaBean and choose to generate annotations for the Enterprise JavaBean. As a developer, you work with this single resource and as changes are made and the file is built, the appropriate artifacts are generated and the deployment descriptor information is updated. The generated artifacts, like the interface files, will be stored in a new directory called gen/src (generated source). The bean file, which contains all of your business logic, will still be under the Java Source directory just like previous releases of the WebSphere Studio products. By generating annotations for the specific bean, you now have to work with only a single file. Making updates to the annotations of that file will generate the necessary artifacts and deployment descriptor as needed.



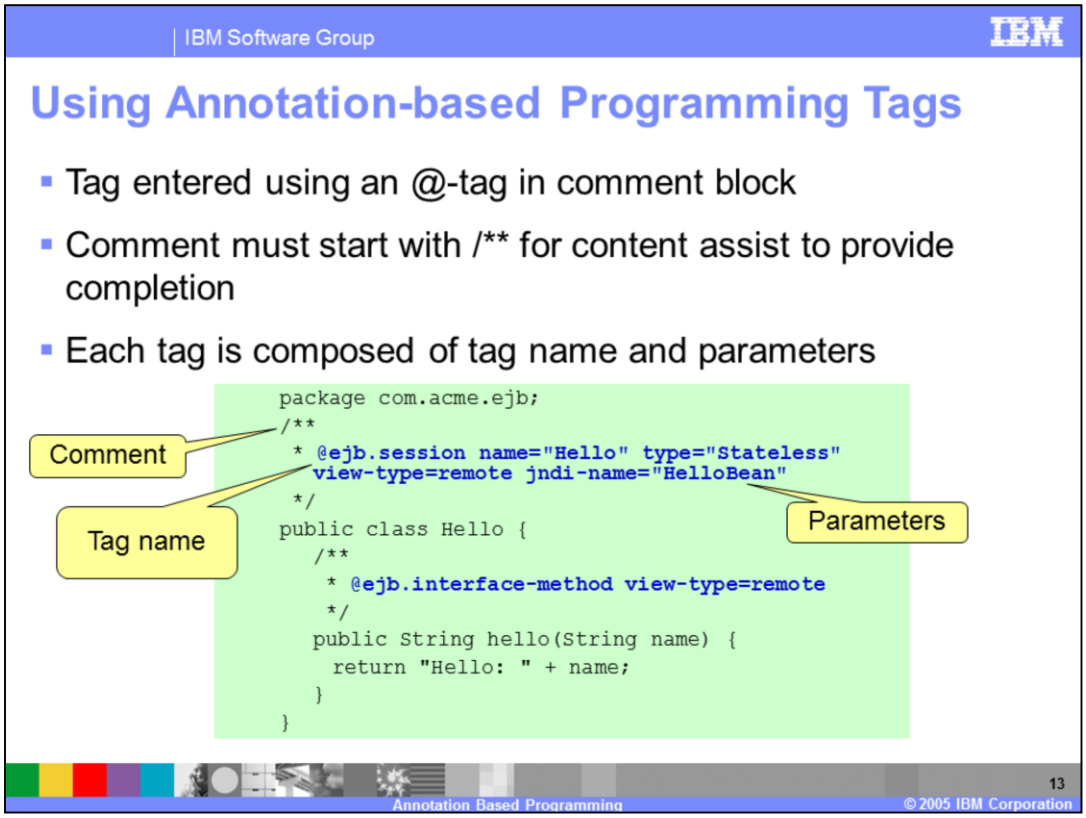

As stated earlier, XDoclet is a popular open-source project that supports annotation-based programming. The XDoclet model processes annotations as part of the build process, when all annotations are read and all artifacts are regenerated. There is much functional overlap between annotations in XDoclet and in IBM Rational Web Developer and IBM Rational Application Developer. The processing model however, is very different as WebSphere Rapid Deployment and the Rational products support an on-demand processing of the annotations. Once all of the tags are identified which have changed, the appropriate artifact or deployment information will be generated rather than generating artifacts from all of the tags which may have not changed. Due to this reason, WebSphere Rapid Deployment does not directly leverage code from the XDoclet project. WebSphere Rapid Deployment and the IBM Rational products have adopted the tag syntax used by XDoclet in places where XDoclet already defines a set of tags, such as for J2EE applications. This will allow people who understand XDoclet to be immediately familiar with annotation-based programming, and it will allow source code compatibility between WebSphere Rapid Deployment, the IBM Rational products and XDoclet tools. When XDoclet 2.0 is released with tags for J2EE 1.4, support will be updated to support these tags and generate the appropriate code and deployment descriptors.




In terms of creating or generating resources from annotation tags, there are three types of tag definitions that are supported for generation; EJB, Web, and Web Services. The first two types of tags have content assist built within IBM Rational Web Developer and IBM Rational Application Developer. For EJB resources, the type of EJB can be specified along with the interfaces, methods promoted to those interfaces, primary key information for entity beans, references, and other things such as EJB QL queries. For Web resources, servlets, filters, listeners, and references can be specified along with things such as security roles. For the Web Service tags, there is currently no content assist support within IBM Rational Web Developer or IBM Rational Application Developer, however Web Services may be selected to be created with tags. With the Web Service tags, the service endpoint interface, SOAP bindings and EJB bindings can be specified. The tags syntax itself is derived from XDoclet and the complete list of supported tags is included in the Information Center and Help of the IBM Rational products.



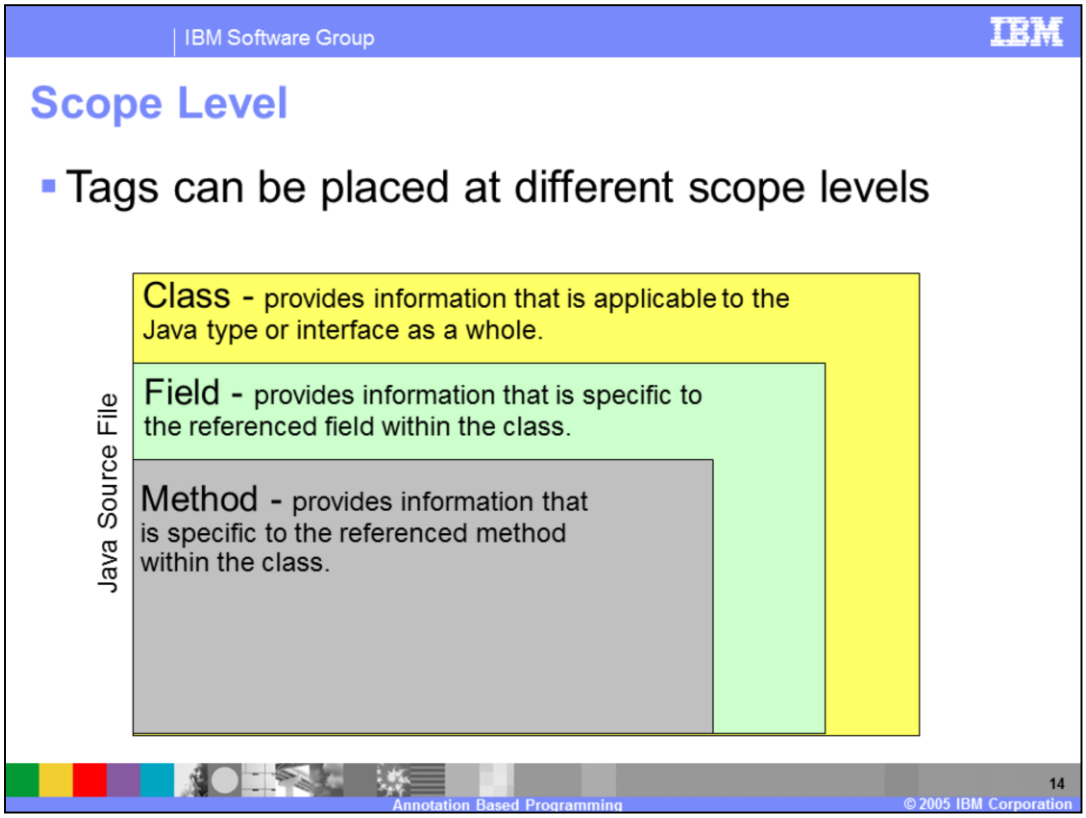

Besides those XDoclet tags which are specific for J2EE, there are WebSphere specific tags which are also supported within IBM Rational Web Developer and IBM Rational Application Developer. These tags which begin with the @websphere string are for specifying mapping information for database columns and foreign key information when establishing container-managed relationships. There are other tags (@ws.*) for support of Service Data Objects (SDO). These tags are available for specifying a session bean façade wrapper for a container-managed persisted entity bean. There are additional tags for specifying a value-object to hold the attributes of a entity bean accessed through the session bean façade wrapper and to indicate which attributes may be contributed to the value-object. Finally, you can define an EJB QL query with tags which will return a set of value-objects. Again, the complete list of supported tags is included in the Information Center and Help.




Annotations can be added in two ways. The first way is at the time an artifact, such as an Enterprise JavaBean or Servlet, is created by selecting the checkbox in the wizard to create the artifact using annotations. A set of basic tags will be used to define the artifacts and deployment information. The second way is to add tags manually to an artifact which has been created with annotations. When adding annotation tags, it must begin with an "@" and in a comment block which starts with /**. Other comment indicators (/* or //) will prevent the annotation processor from recognizing the tags.

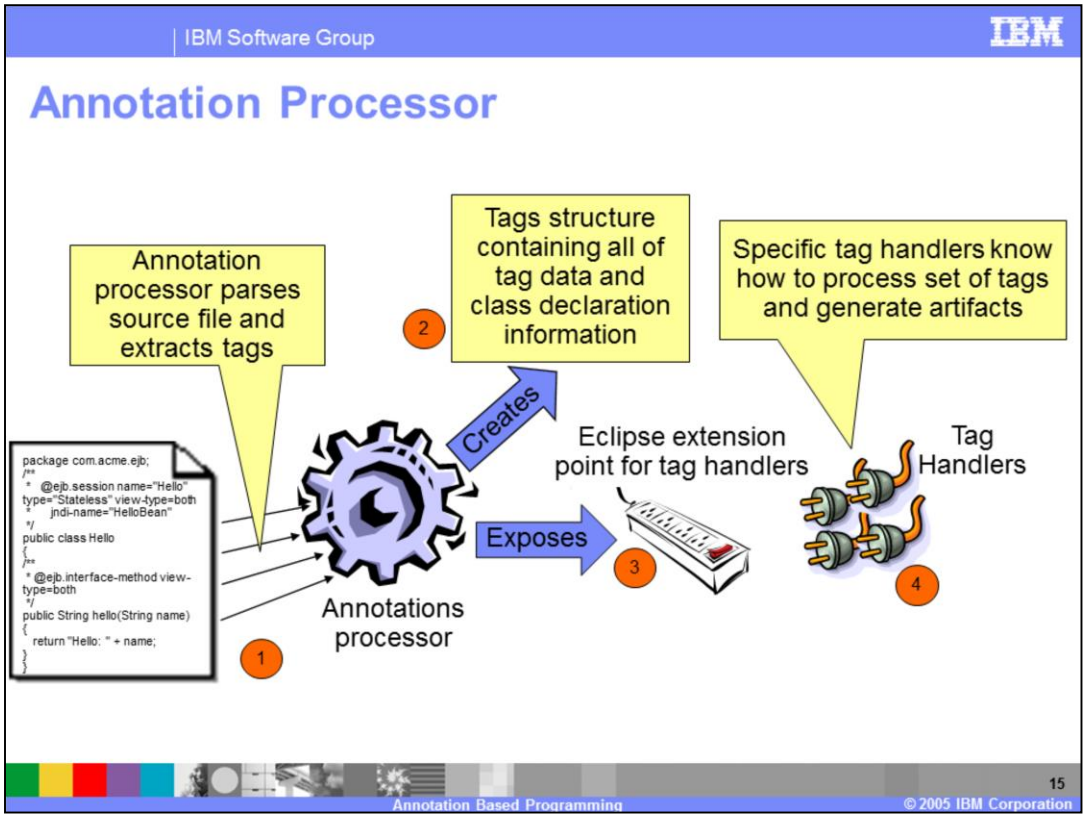
Remembering all of the different tags might be challenging. Built into the Application Server Toolkit (AST) and IBM Rational Web Developer and IBM Rational Application Developer is content assist support for the tags. With a hint or the first part of the tag, contest assist can provide the available tags and parameters that are required for each of the tags at the current scope level. Additional parameters which are not required can be specified with content assist as well by providing a hint and then pressing Ctrl+Space together. For a complete list of parameters for the supported tags, check the Information Center or Help.

Notice the tags are specified in comments in different places in the Java file. These different locations are known as scopes for the tags. Certain tags are only available in certain scopes.

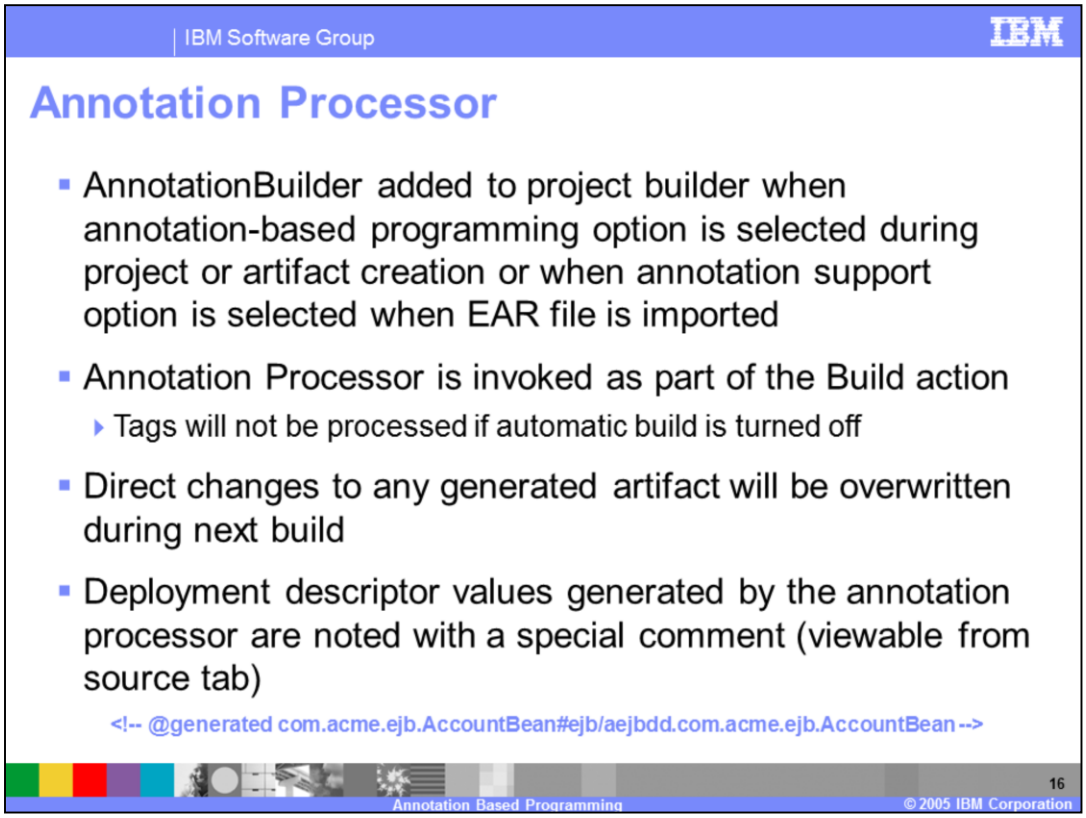



For tags in general, they can be placed in one of three locations within the Java source file. The location is called the **scope** of the tag. The first scope level is class. The Class scope tags are added to the class comment. This scope provides information that is applicable to the Java type or interface as a whole. The second scope is field. The Field scope tags are added to the comments of a particular field within the class. This scope provides information that is specific to the referenced field within the class. The last scope is Method. The Method scope tags are added to the comments of a particular method within the class. This scope provides information that is specific to the referenced method within the class.

There is a fourth scope called package. This scope is for any tags which provide information applicable to the entire Java package, to the module, or to the application as a whole. Currently there are no tags which are supported with content assist for this level.




Besides specific tags placed in certain scopes, there is also the processing of the tags. Annotations will be processed using a special builder called the AnnotationsProcessor. The function of the AnnotationsProcessor is to extract the tag data from the Java source file and enable artifacts to be generated from that data. The extraction phase will utilize a Java syntax parse tree function that is included with Eclipse. This allows for a fast mechanism in extracting the tag data. The AnnotationsProcessor will create a structure containing all of the tag data and class declaration information and then expose an Eclipse extension point for plugging in a tag handler. A tag handler is what understands how to process a particular set of tags and generate the appropriate artifacts. After extracting all of the tag data from the Java source file, the AnnotationsProcessor calls all of the registered tag handlers for each tag that is encountered. The tag handler then generate the appropriate artifacts or deployment information for the specified tag.




The annotation processor is responsible for processing the tags. It is added to a project's list of builders when an artifact is created with annotation tags through the Enterprise JavaBean (EJB) or Servlet creation wizard. The annotation processor will also be added to the list of builders if the annotation support option is selected when a enterprise application (EAR) is imported into the workspace. The annotation processor is placed at the top of the list of builders by default and should remain at the top although the builders may be re-sequenced. With the annotation processor at the top, the different artifacts will be generated before other build operations which may require those resources for things such as resolving references. The annotation processor is invoked when a build operation is performed on a project and will generate the different artifacts and deployment information. "Automatic build when a resource is modified" is enabled by default for a workspace and when an artifact is modified in a project, the build process will begin by calling the annotation processor. If automatic build is turned off, when an artifact is modified there will be no processing of the tags. There is no other way in IBM Rational Web Developer or IBM Rational Application Developer to invoke the annotation processor besides through the build process.

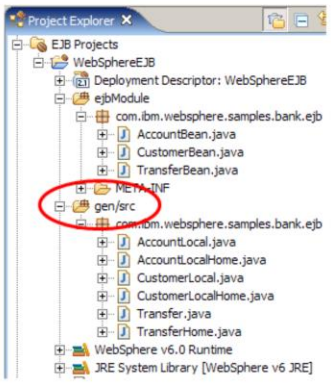
The generated artifacts can be modified at any time, however, when the annotation processor runs again on the source object which contains the tags which caused the artifact to be generated, any changes will be overwritten. Modifying generated artifacts should be avoided.

For deployment information which is generated from tags, these values can not be

modified without disabling the annotation support for the source artifact which caused the values to be created. When an Enterprise JavaBean (EJB) or Servlet is created with annotations through the creation wizard, a special comment, which is only recognized by the Application Server Toolkit (AST), IBM Rational Web Developer, and IBM Rational Application Developer, will be placed in the deployment descriptor near the definition of the object. This comment is used by the editor to prevent accidental modification of the generated values. If there is an attempt to change a generated deployment value, the editor will warn the user that changes to the value will disable the annotations support for the particular artifact. The user can accept the warning and change the value, however the annotation processor will no longer generate artifacts or deployment information for the artifact. Normal means of updating the generated artifacts and deployment information can be used.

## Generated Artifacts

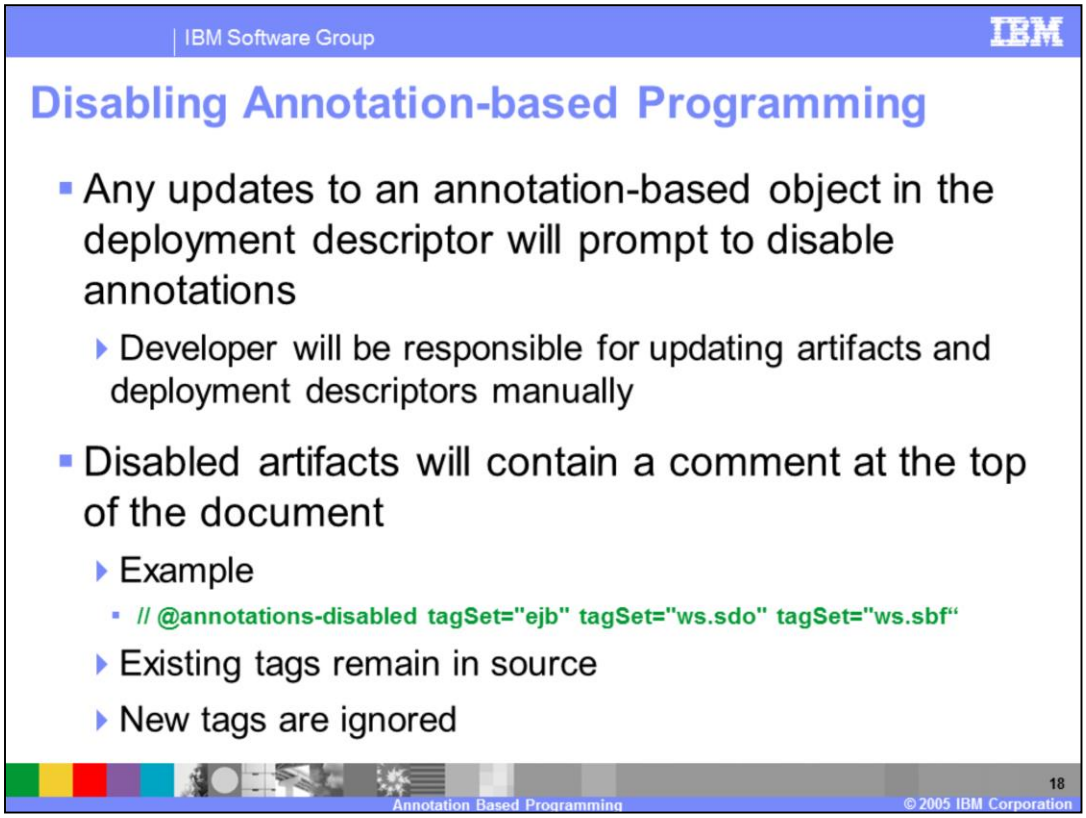
- Artifacts generated by the annotation processor are placed in gen/src folder
- EJB artifacts placed in gen/src folder of EJB project or EJB Client project
- Web artifacts placed in gen/src folder of Web Project
- Underlying directory structure <workspace>\<project>\gen\src



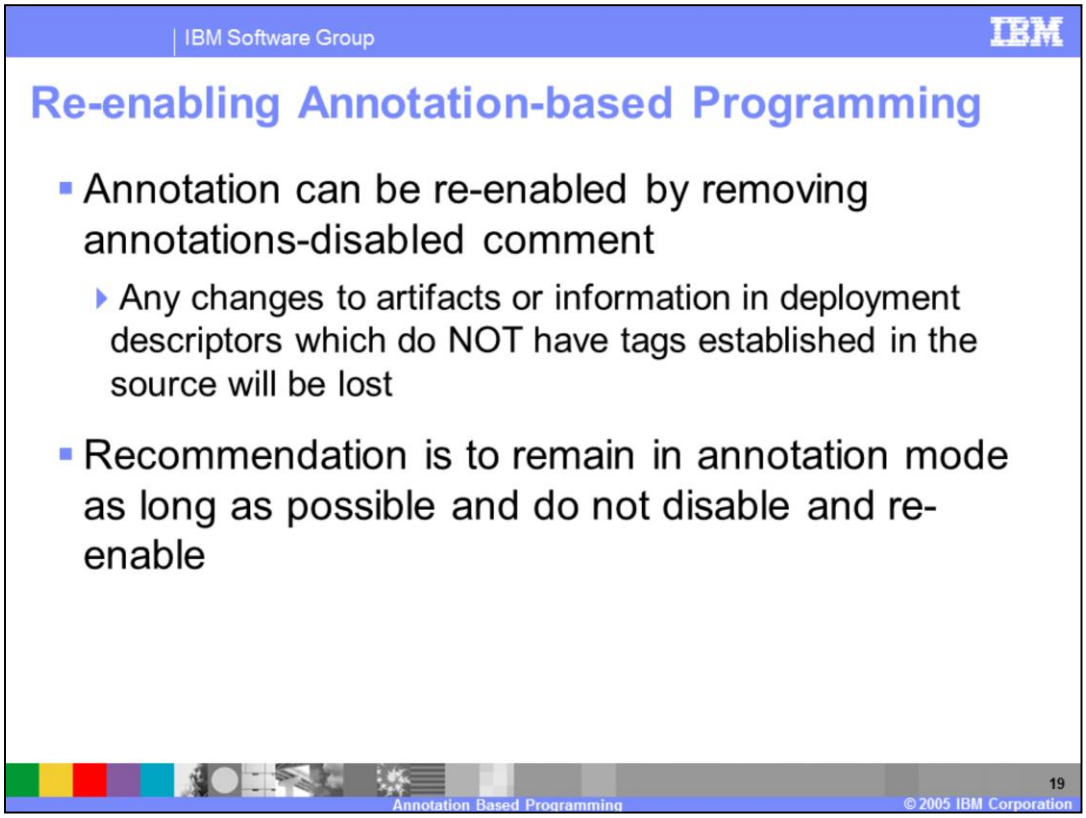




As stated earlier, generated artifacts are placed in a folder named gen/src in the project. If the artifact is in an Enterprise JavaBean or EJB project which has a EJB client JAR file associated with it, the different interface files will be generated in a folder named gen/src in the EJB client JAR project. For artifacts in Web projects, the generated files will be placed in a gen/src folder as well. While the name of the folder appears to be gen/src in the view, it is implemented as a gen directory containing a src directory on the file system. This structure may be important for ANT tasks and other custom build operations.



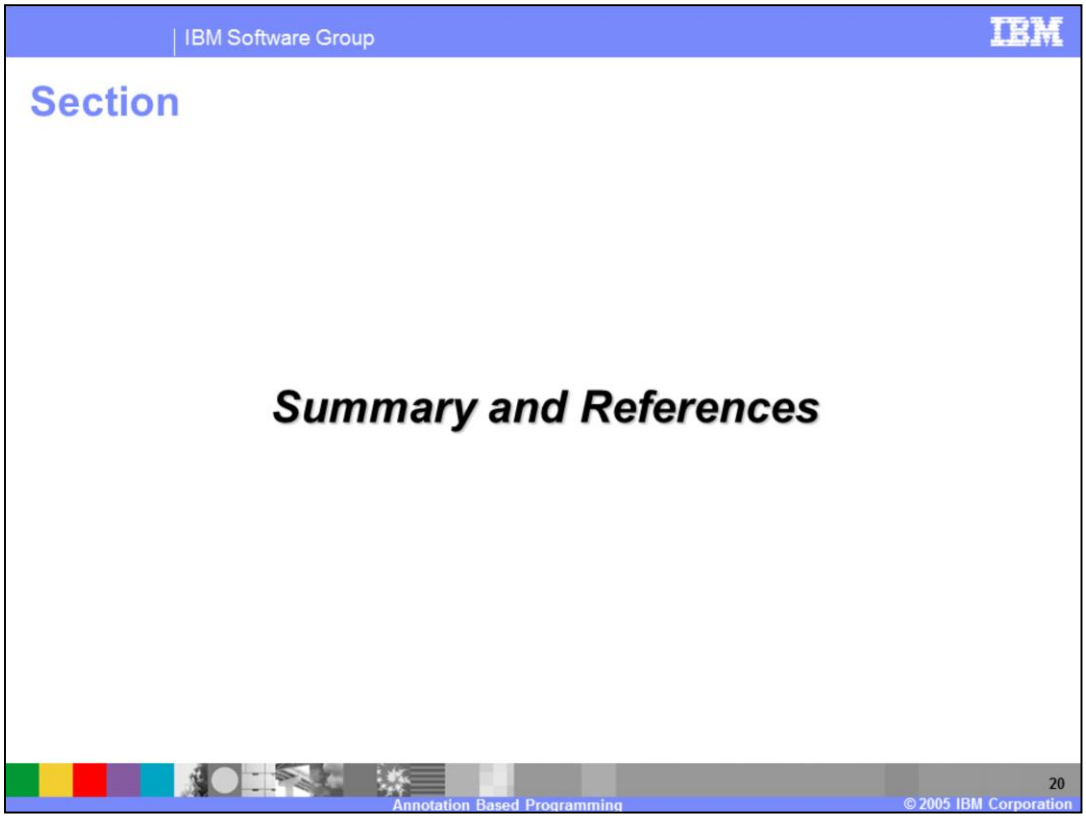

As mentioned earlier, annotation tag processing and thus annotation support for an artifact can be disabled by manually updating a deployment descriptor value which was created from a tag. The deployment descriptor editor will prevent an accidental change of a generated value, by warning the user that a change will disable the annotation support for an artifact. If the user accepts the warning and changes a value, the user will then be responsible for modifying any of the deployment information generated for the artifact as well as maintaining the artifacts which were generated and placed in the gen/src folder. When the annotation support is disable for an artifact, the tags will remain in the file and all generated artifacts will also remain. Any new tags which are added or existing tags changed will have no affect and will be ignored. Within the source file, a comment is added at the top of the class which is recognized by the annotation processor to skip annotation processing on the file when the annotation processor runs on the project.



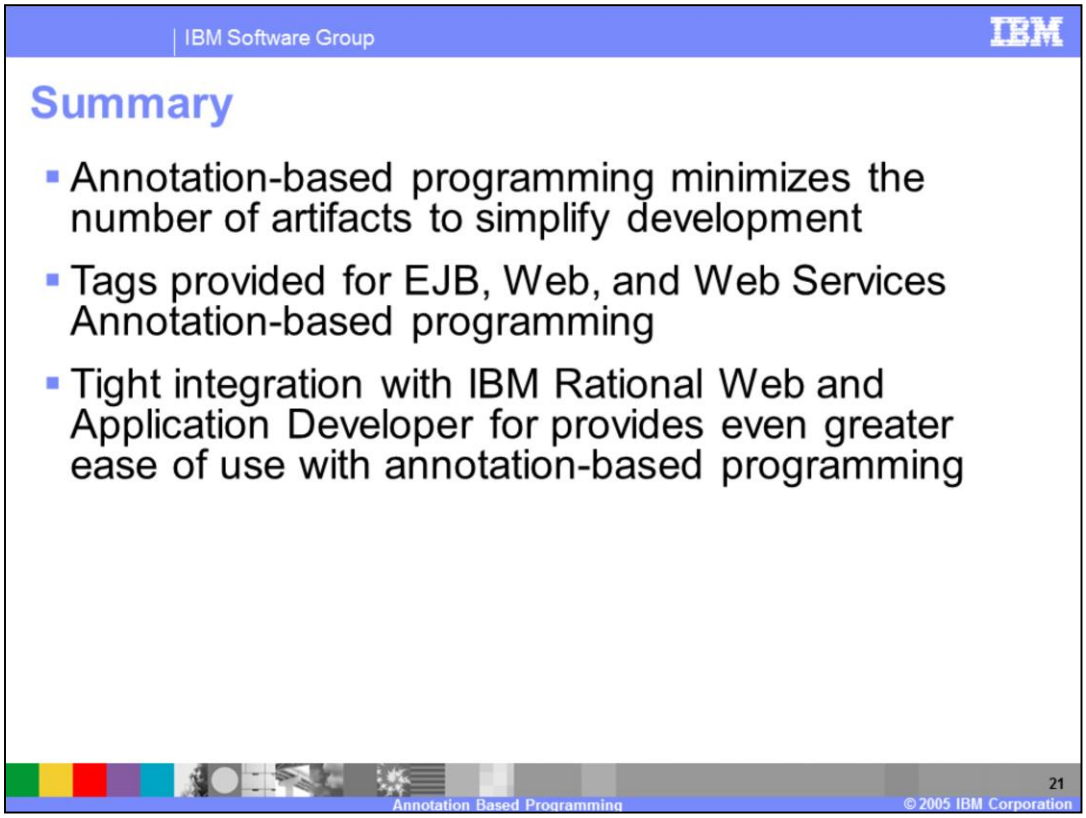

If annotation support is disable for an artifact, annotation processing can be re-enabled by removing the comment which causes the annotation processor to skip the artifact when processing the project. If the comment is removed, on the next build operation, all of the tags will be recognized and the appropriate artifacts will be generated and deployment information will be created. Any changes made to the artifacts or deployment information which are not specified in the tags will be overwritten or lost. Currently there is no way to capture changes and settings from artifacts and deployment descriptors and generate tags.

Although annotation processing can be disabled and re-enabled for an artifact, the recommendation is to remain in annotation mode for as long as possible and once and if the switch is made to normal maintenance and management, to not re-enable annotation processing. Switching back to annotations from normal mode will loose information and can lead to other inconsistencies within the workspace.

Annotation processing can also be disabled by deselecting the annotation processor in the list of builders in the project properties. This will disable generation for all artifacts enabled with annotation tags.




This final section provides the summary to the presentation.




In this presentation, the benefits of annotation based programming were covered as it can reduce the number of artifacts a developer must manage. The different tags which are available and have annotation processing support were discussed along with the tight integration with IBM Rational Web Developer and IBM Rational Application Developer which simplify the use of annotation tags and annotation processing.

# References

- XDoclet information
  - http://xdoclet.sourceforge.net/xdoclet/index.html.