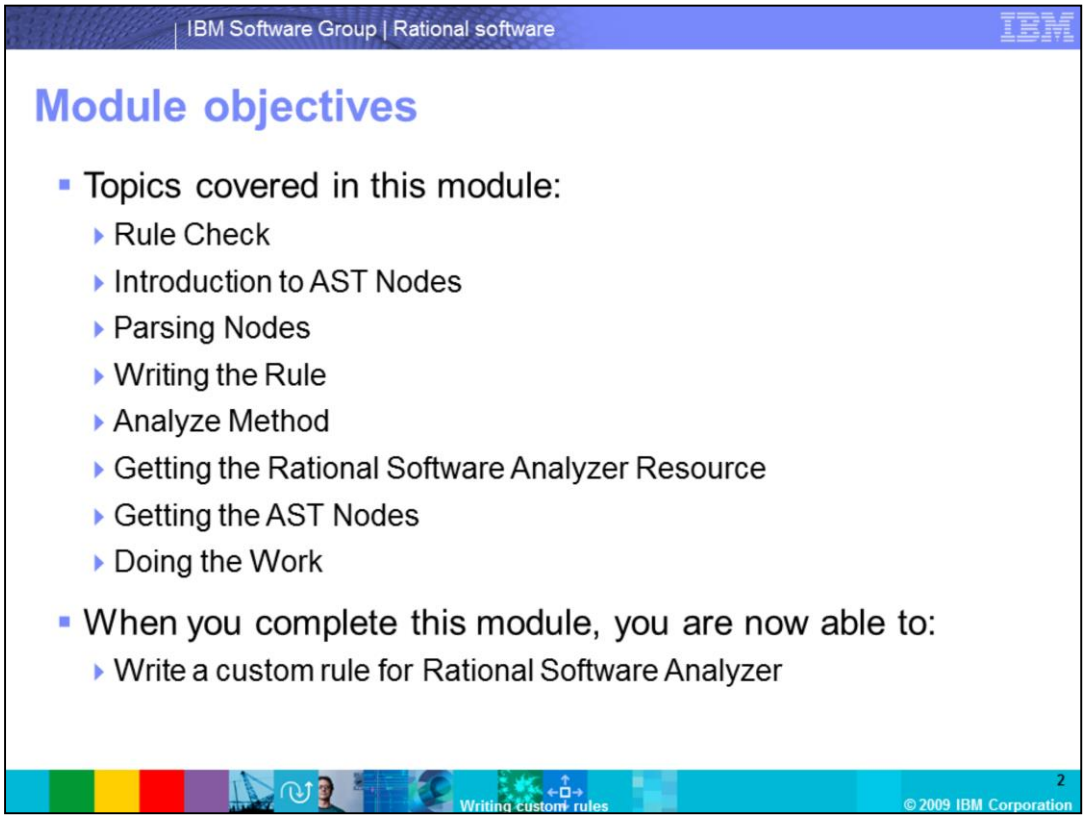
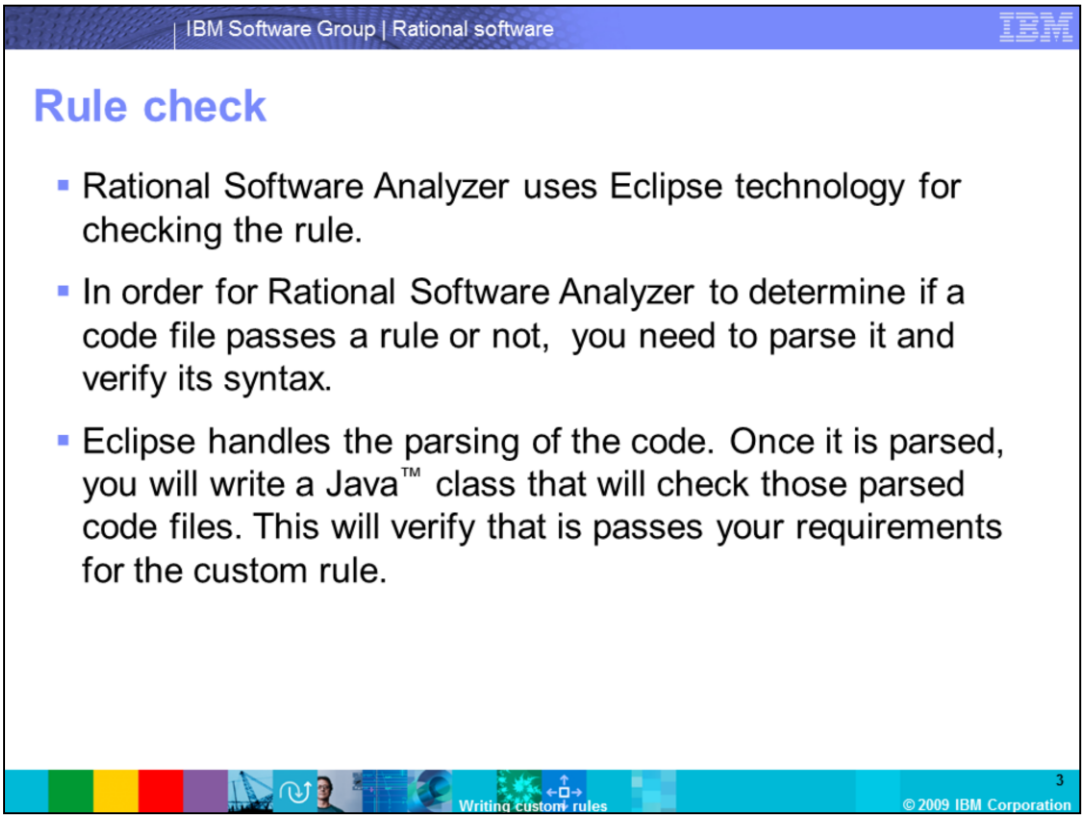
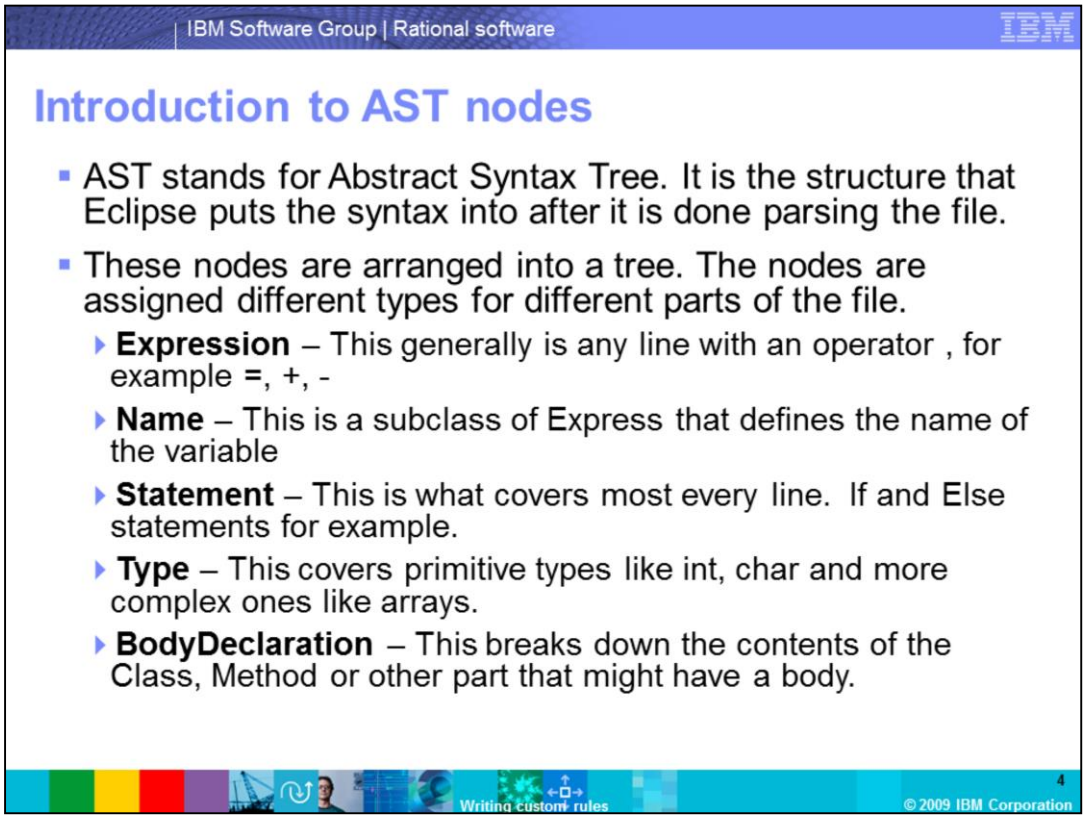
This module covers the topic on how to write a custom rule for IBM Rational® Software Analyzer versions 7.0 and higher.



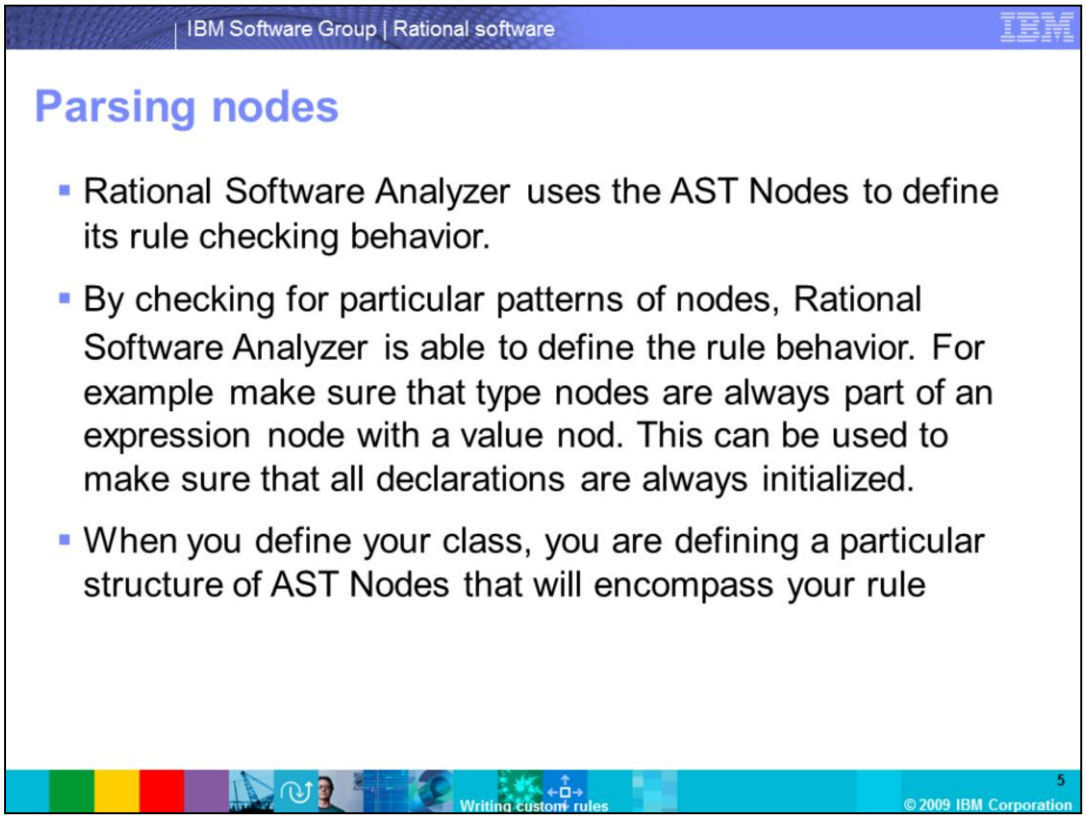
This course covers eight topics. The first three topics provide you with information about a Rule Check, an introduction to AST Nodes and Parsing Nodes. The next few topics outline writing the rule, the Analyze Method, getting the Rational Software Analyzer resource, getting the AST Nodes and doing the work. When you complete this module, you will understand what an AST Node is and how it relates to Rational Software Analyzer. You will also be able to write a custom rule for Rational Software Analyzer.



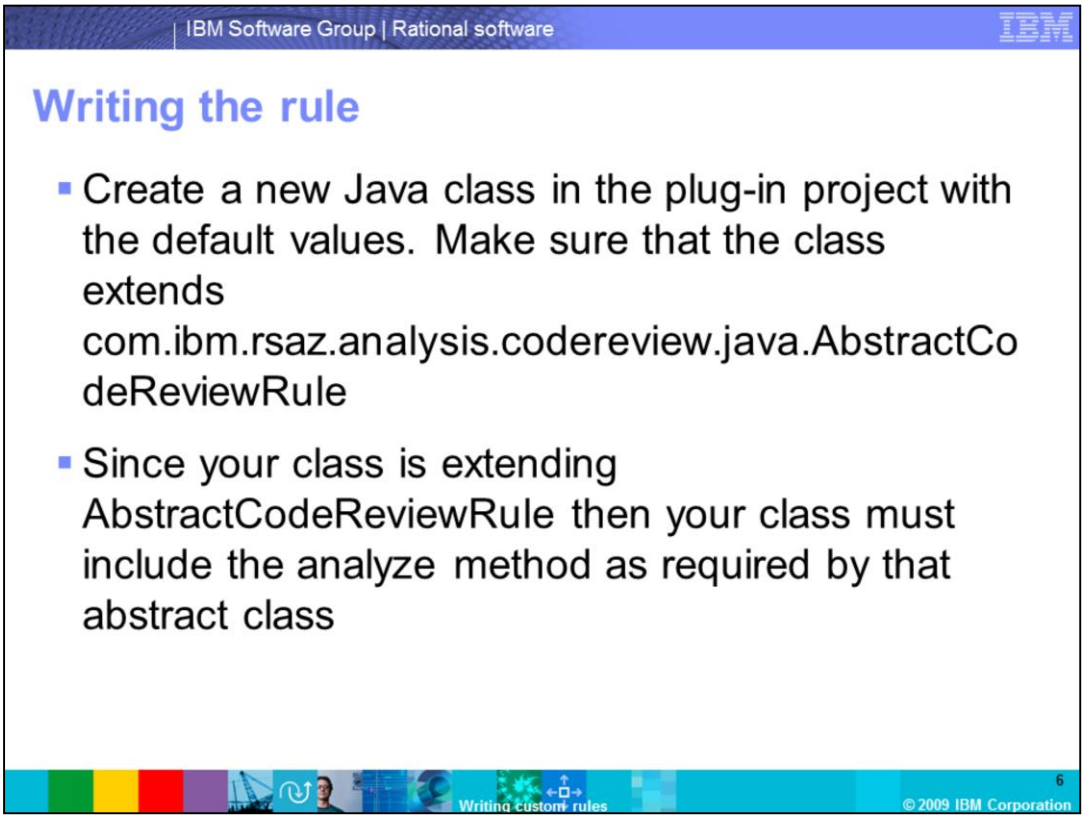
This slide will cover three points. The first point lets you know that Rational Software Analyzer does not have its own code parser, rather is uses the existing Eclipse parsers. Secondly, Rational Software Analyzer uses the results of the parser pass to determine if a rule has passed or failed. And thirdly, you need to write your Java class to look for particular patterns in the code file to make a decision as to whether the file passes the rule.



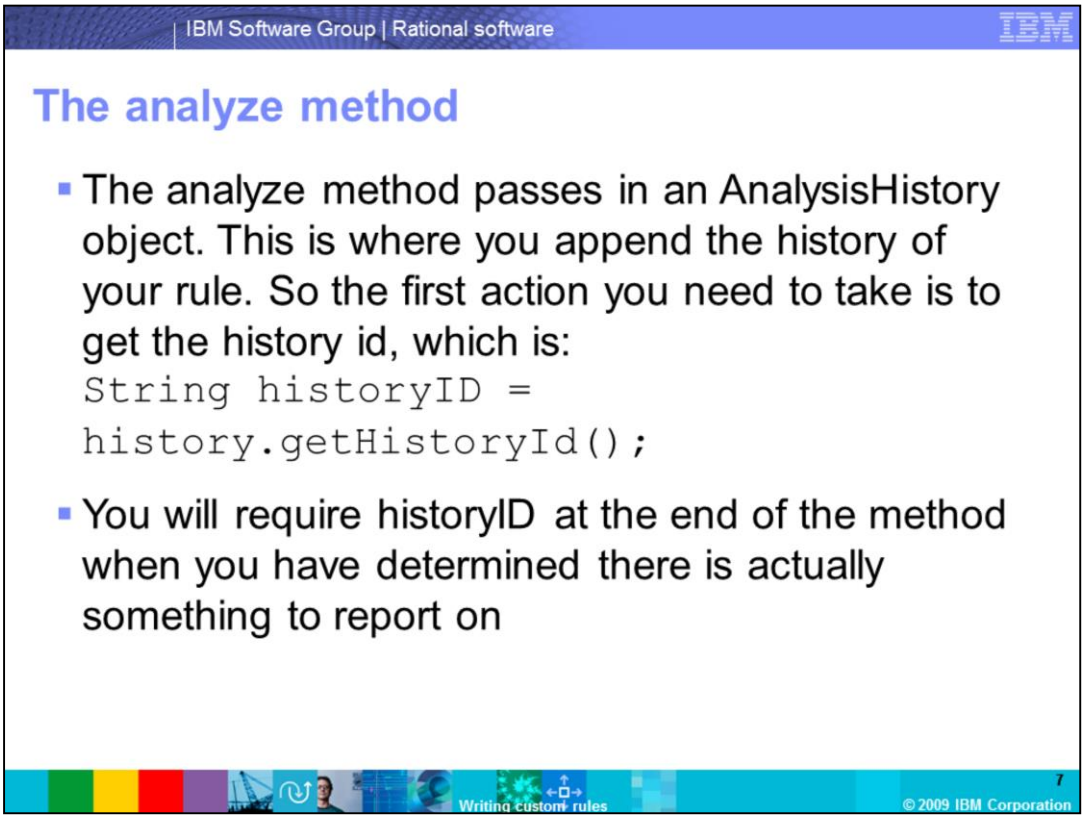
This slide gives an Introduction to AST Nodes. Abstract Syntax Tree (AST) is the data structure that is the result of the Eclipse parser pass. The data structure itself is a product of the Eclipse project, but its purpose is to break code files down into a tree structure that represents the file fully. The tree will start at the highest, broadest scope and through the children of that top level node, create gradually narrower scoped nodes. These nodes will have different types for different parts of the code file, however everything that the parser encounters is represented somehow in this tree.



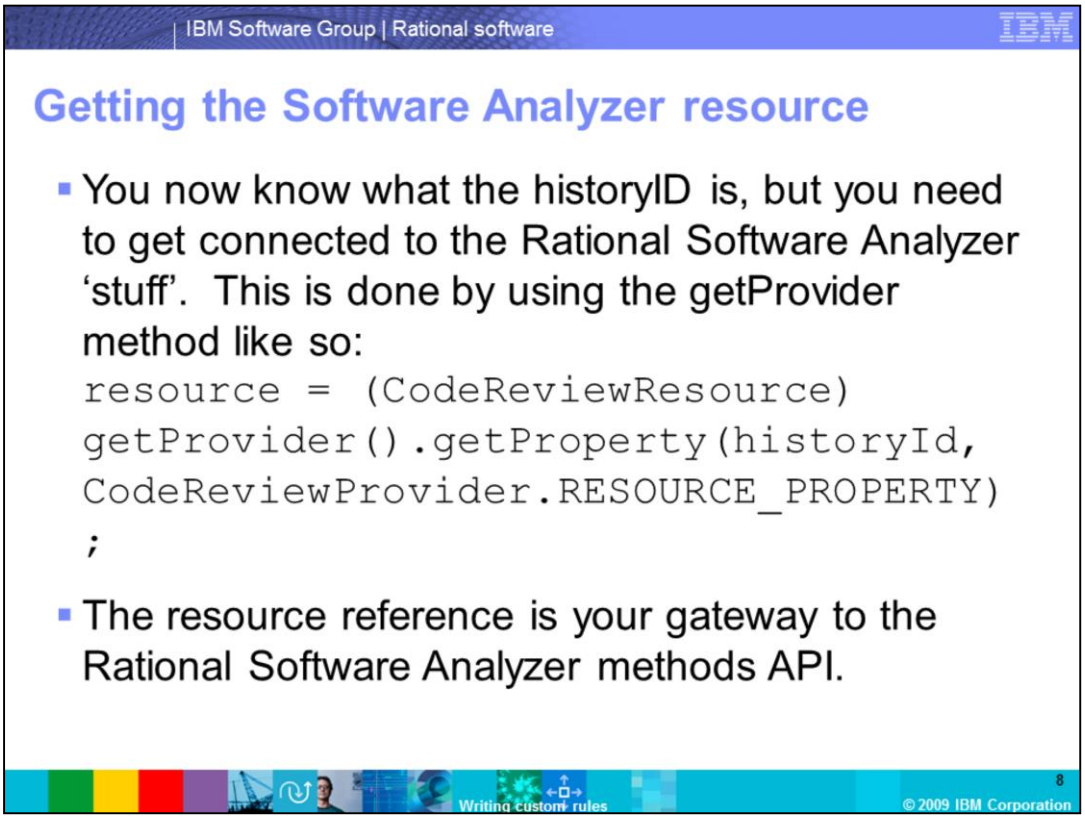
Rational Software Analyzer uses this Abstract Syntax Tree to check its rule syntax. So when you are defining your rule for Rational Software Analyzer, what you are really doing is specifying a particular AST structure that will either be acceptable or unacceptable. So in fact checking the rule will involve traversing the tree and looking for a specific pattern.



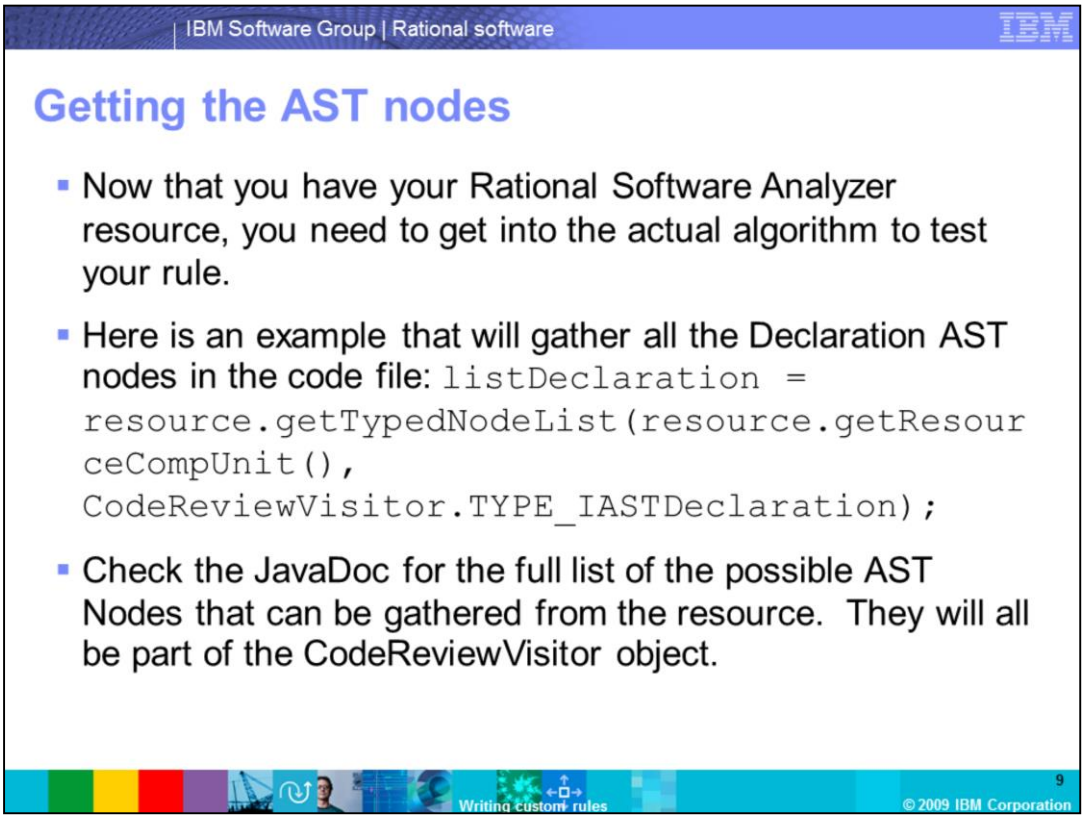
In the previous set of slides, you have set up the custom plug-in, but you did not create the Java class. You already have all the plug-in blocks in place, but you need to write the actual steps to check the code. The Java class needs to extend the AbstractCodeReviewRule object. This Rational Software Analyzer object contains the analyze method which is what Rational Software Analyzer will call to start the scan. Therefore you are putting all of your analysis code there.



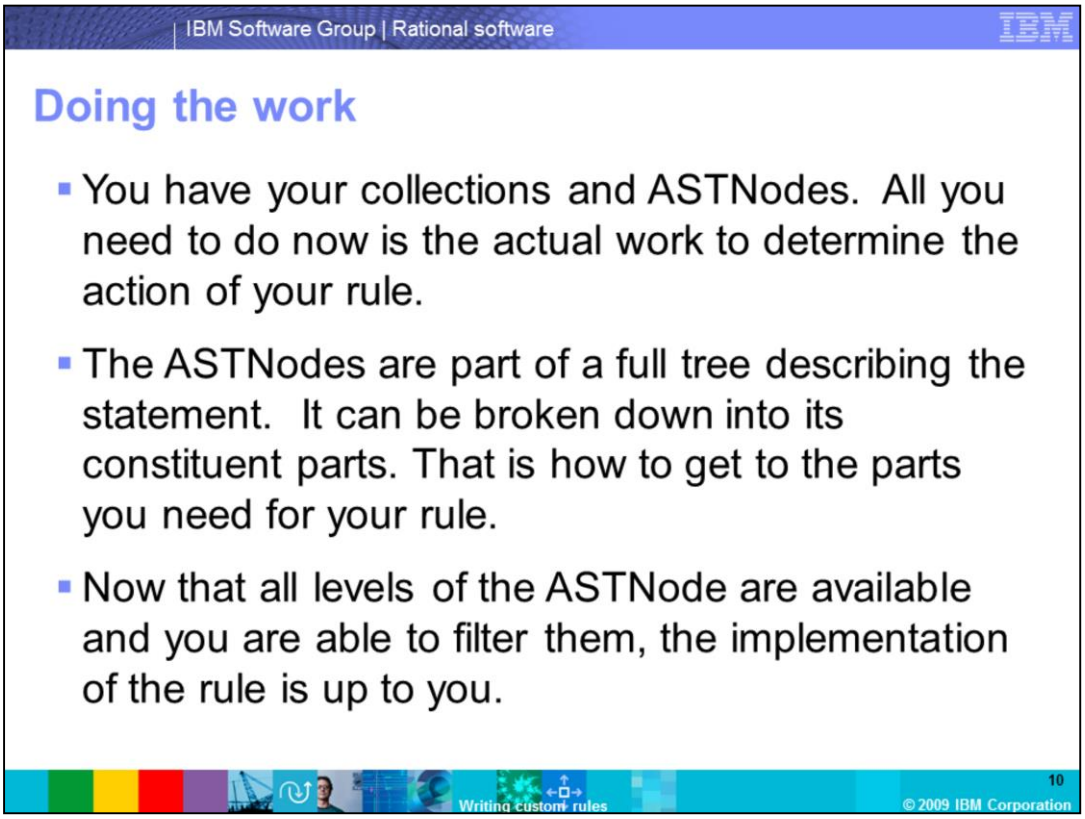
The analyze method will pass in an AnalysisHistory object. This will tell the analyze method what the previous history of the rule scan is. The first action you should always take when you are implementing the rule is to acquire the historyID, so you know where to put the results of this rule. You will reference this later on when you decide whether to add something to your history for this rule.



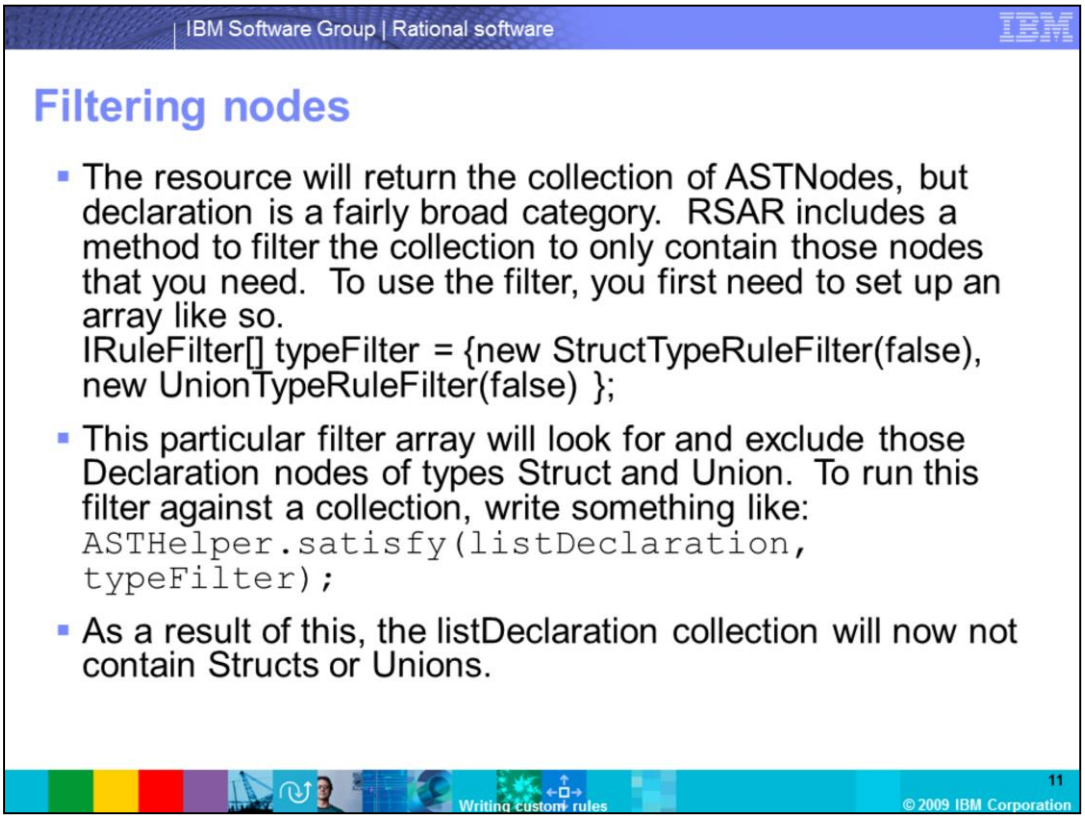
You now have the historyID, but you need more from Rational Software Analyzer. You need to get access to the Rational Software Analyzer API and its associated objects. The way to do this is through the getProvider method. By using the syntax here you are able to get the Rational Software Analyzer CodeReviewResource object which is your window into all things Rational Software Analyzer.



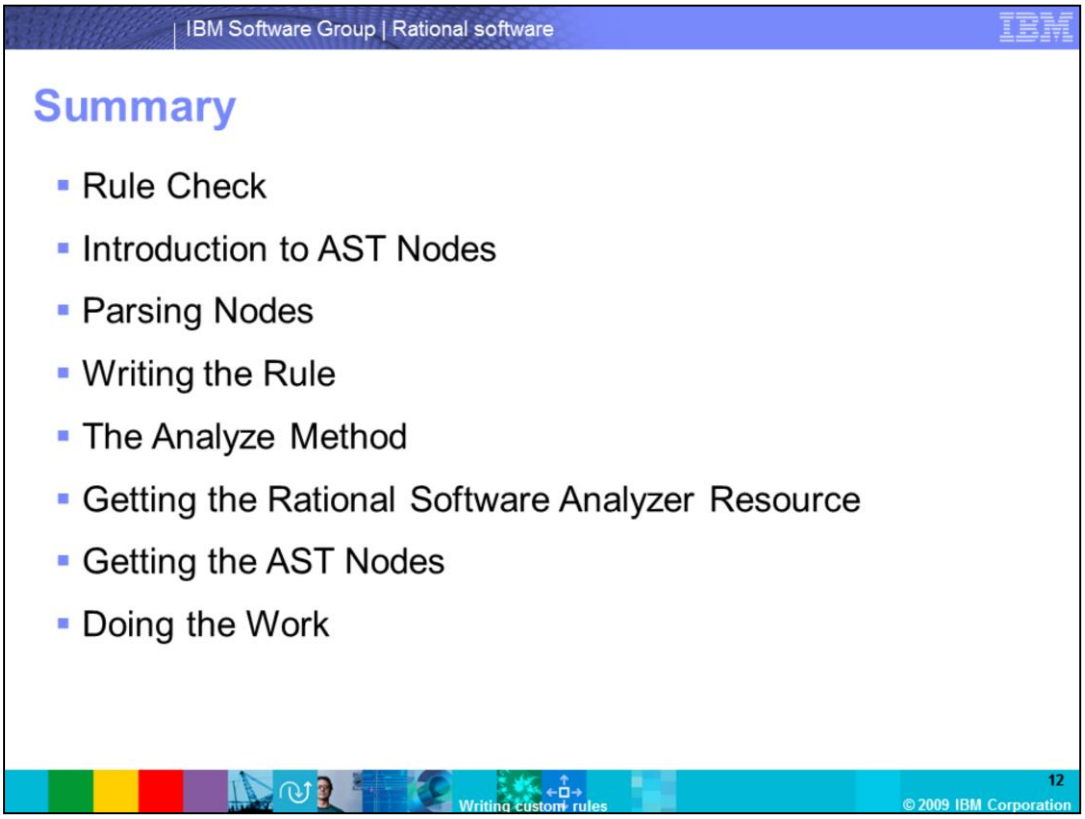
The resource reference is set up and ready to go. The last thing you need to do is set up the code to check the pattern to verify if the rule passes or fails. First you need to collect the AST Nodes that are of interest to you. In the example here you are gathering an array of nodes of the ASTDeclaration type. From the array you can sift through looking for particular declaration nodes of interest.



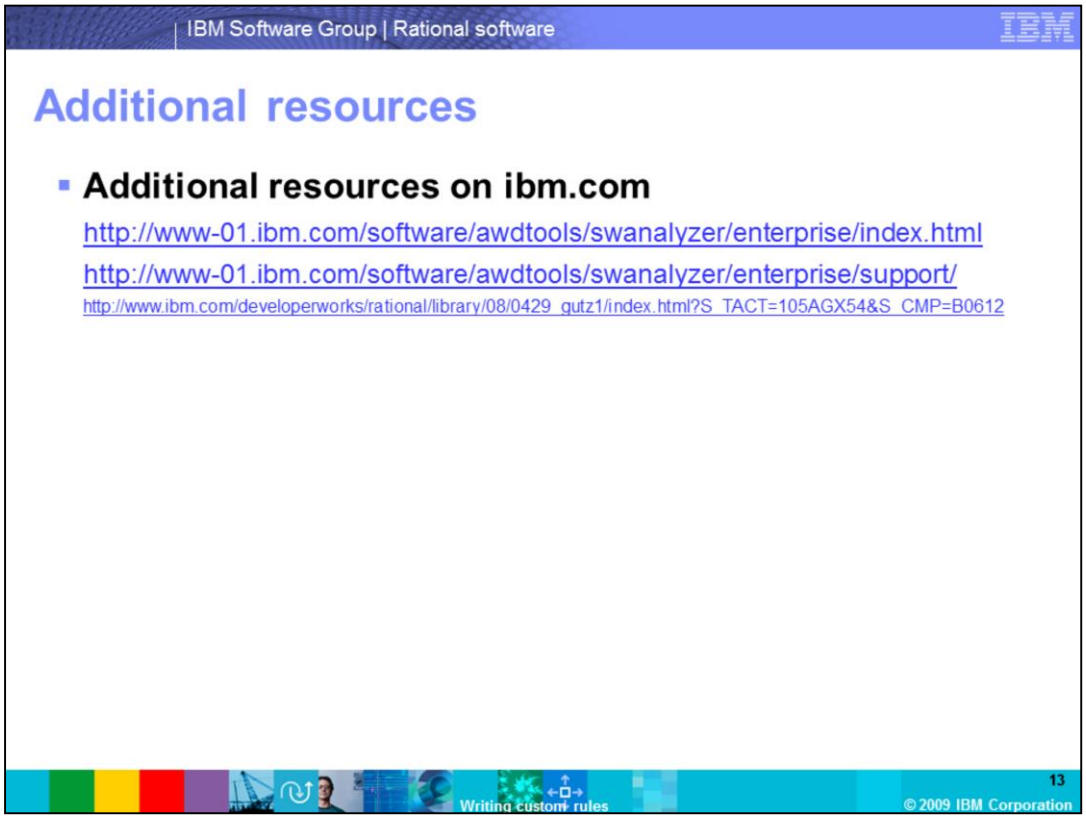
At this point you have the arrays of AST Nodes. Now you have all the parts you need to start checking rules. By checking for particular patterns of nodes in the arrays you collect, you can decide what actions to take to determine the success or failure of your custom rule.



Rational Software Analyzer provides an additional filter method on top of those provided through the ASTNode methods. The IRuleFilter allows not only the particular type of ASTNode to be filtered but it can get more specific implementations of the ASTNode. For example, both Struct and Union declarations are ASTDeclaration ASTNode types. Without iterating through the entire array of Declaration Nodes you do not know what implementations there were. By using the satisfy method with the IRuleFilter array you can check that the collection either inclusively or exclusively contains those particular nodes.



This module provided an overview of writing custom rules in Rational Software Analyzer. You should now be familiar with knowing what an AST Node is and how it relates to Rational Software Analyzer. You are now able to write your own custom rule.



Additional resources can be found on the ibm.com Web site and in the Whitepaper on "**Static analysis IBM Rational Software Analyzer:** Getting started".

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

Rational

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Rational is a trademark of International Business Machines Corporation and Rational Software Corporation in the United States, Other Countries, or both.

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

Other company, product, or service names may be trademarks or service marks of others.

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.

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.