Software Code Quality with UML-based Design Models

Linda Ekberg
DEGREE PROJECT

Software Code Quality with UML-based Design Models
Linda Ekberg

Summary
The Unified Modeling Language has since 1997 come to be a de facto standard for software analysis and design and is always under development to provide better modeling support. But as the notation aims to be as general as possible it must cut back on some features, hence introducing the question of how much that has been left out of the notation and what effect it has on the software product in the end. Therefore, this paper presents an experiment where existing software has been modeled in UML, generated back into code and then compared to the original with the purpose of examining the quality of code which is based on UML diagrams.

Results from the experiment show that UML lacks of modeling support for displaying class visibility, which can jeopardize system safety, and is inadequate for representation of nested conditional statements, return values and exception handling to mention some.
Preface

I would like to thank my examiner, Stefan Christiernin, for his constant support and inspiration. You have always spoken your mind when not content with the work presented and even though I have been quite stubborn sometimes you have not given in. This has enhanced both the way I look at things as well as the paper.

I would also like to thank my advisor, Andreas Boklund, for answering all my questions concerning UML, but also for some very inspiring discussions about the past and future of software engineering. This too has been a great influence on my thoughts and writing and consequently on the end result.
## List of symbols

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CBR</td>
<td>Checklist-based reading technique for software inspections</td>
</tr>
<tr>
<td>Class diagram</td>
<td>UML model for display of classes in a software system</td>
</tr>
<tr>
<td>DBR</td>
<td>Defect-based reading technique for software inspections</td>
</tr>
<tr>
<td>Forward engineering</td>
<td>The process of transforming software code into design models</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>I/O</td>
<td>Input and output of information in a software system</td>
</tr>
<tr>
<td>Java</td>
<td>Object-oriented programming language</td>
</tr>
<tr>
<td>LOC</td>
<td>Lines of code in software code documents</td>
</tr>
<tr>
<td>OMG</td>
<td>Object Management Group, responsible for the UML notation</td>
</tr>
<tr>
<td>OO</td>
<td>Object-oriented, software development approach</td>
</tr>
<tr>
<td>PBR</td>
<td>Perspective-based reading technique for software inspections</td>
</tr>
<tr>
<td>Reverse engineering</td>
<td>The process of generating code from design models</td>
</tr>
<tr>
<td>SBR</td>
<td>Scenario-based reading technique for software inspections</td>
</tr>
<tr>
<td>Sequence diagram</td>
<td>UML model for display of message flow within a software system</td>
</tr>
<tr>
<td>TBR</td>
<td>Traceability-based reading technique for software inspections</td>
</tr>
<tr>
<td>UBR</td>
<td>Usage-based reading technique for software inspections</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language, notation for software analysis and design</td>
</tr>
<tr>
<td>V&amp;V</td>
<td>Verification and validation</td>
</tr>
</tbody>
</table>
Abstract

The Unified Modeling Language has since 1997 come to be a de facto standard for software analysis and design and is always under development to provide better modeling support. But as the notation aims to be as general as possible it must cut back on some features, hence introducing the question of how much that has been left out of the notation and what effect it has on the software product in the end. Therefore, this paper presents an experiment where existing software has been modeled in UML, generated back into code and then compared to the original with the purpose of examining the quality of code which is based on UML diagrams. Results from the experiment show that UML lacks of modeling support for displaying class visibility, which can jeopardize system safety, and is inadequate for representation of nested conditional statements, return values and exception handling to mention some.

1. Introduction

Software development is an ever changing area that has evolved from structured programming to object-oriented programming to component-based development [1, 2]. The personal computer’s entry into the lives of everyday people during the 1980s [1] is sure to have hastened this process as it ignited a spark of interest in computers, which has led to the success of computers today. But software development is not all about programming but is instead a process consisting of different phases for analysis, design, implementation and deployment [3]. And as the competition between software vendors has increased over the years, the need for better processes to produce as competitive products as possible has increased as well.

The Unified Modeling Language, from now on referred to as UML, has over the years become a de facto standard used for software analysis and design. UML defines a set of diagrams intended to facilitate the software development process as each diagram is aimed at different aspects of the analysis and design phases. Today UML is commonly used and is supported by a series of software tools [4], each giving its own interpretation of the ever growing UML notation [5]. It is easy to believe that something as widespread and used by so many must be quite revolutionary. But at the same time, a standard which is not aimed towards a specific programming language or a specific development process must naturally yield in some areas to fit as many as possible. Hence, it is interesting to take a closer look at the UML notation and examine its modeling possibilities and what use the models are to the implementation of the modeled system.

This paper consequently aims to examine software code quality based on UML diagrams. An experiment is conducted where code from a commonly used program is reverse engineered into UML diagrams and then forward engineered back to code to investigate what can and cannot be modeled with UML and more importantly what quality the code produced possesses. The result is then discussed in relation to the severity of the UML drawbacks found, such as unhandled exceptions and class visibility loss, in comparison with the intended purpose of the notation.

The next section puts the investigation into context by addressing the background of the subject, followed by the experimental setup of this paper. The results of the investigation are then presented and discussed followed by a summary of it all in the final conclusion.

2. Background

When object-oriented (OO) programming became common practice in the 1980s, the need for methods to analyze and design software increased [6]. Since no standard for OO analysis and design existed, separate groups of software developers produced their own methods and models to envision the software system to develop. The variety of processes developed during the 1980s and first half of the 1990s made it difficult to
develop software tools to support each individual process and it became obvious, however reluctantly, that a standard was needed.

2.1. The Unified Modeling Language

Rumbaugh and Booch [6], two leading methodologists by the time, found it wise to merge their processes into one, which shortly after was merged together with the process of another methodologist by the name of Jacobson [6, 7, 8]. In 1996 their joint effort had led to a notation called the Unified Modeling Language (UML). In 1997 the Object Management Group (OMG) assumed responsibility for the notation along with parts of processes of leading software vendors and practitioners such as Rational Software, Microsoft and IBM [6]. Later that same year the notation of UML 1.1 was issued as an OMG standard for object-oriented software analysis, design and documentation [9]. Since then, OMG has updated the notation and the current version of UML, version 1.5 [10], was published in March 2003 [5]. Meanwhile a new version is on its way to becoming an OMG standard and that version contains enough changes to earn a major versioning number of its own, namely UML 2.0 [11, 12]. This paper will not elaborate on version details but where the previous versions have focused on analysis and modeling of smaller software projects, version 2.0 supports big-scale projects with architecture modeling [13], the new development approach of component-based development [2, 3] and dynamic behavior descriptions to mention some [11, 14]. In addition to this, the new version has improved the overall syntax and semantics [14], which will lead to better models, and better code generation support [15].

2.2. Forward and reverse engineering

The process of transforming software code into design models is called reverse engineering [8] and is supported by many UML tools today [4, 16]. The tools usually read in the selected source code and create class diagrams displaying classes with their variables, methods and associations to other classes. If everything in the code cannot be transformed into models, such as method content, the tools tend to store the left over information in different ways so that it is not lost if the models are used later to produce new code [17]. The opposite of reverse engineering is the process of generating code from design models and is called forward engineering [8]. Most tools supporting forward engineering fail to produce code from several diagram types at the same time and the code that can be generated is not complete as it only defines the skeleton of the system [18]. A problem with UML is that one diagram type is not enough to model all aspects of a complete software system as e.g. a class diagram typically shows which classes that communicate with each other whereas a sequence diagram is used to show exactly which methods of each class that communicate. This division of the system models aggravates code generation. Due to this, UML and other modeling approaches are often used for system analysis and design to produce a set of diagrams for each developer to follow when writing the code manually so to speak [18]. Since the specifications of the system, whether it be models or plain text, are hard to translate into equivalent code any code generated is usually distrusted [19].

If automatic code generation is to be used, the generated code must be checked carefully to assure that it represents the modeled functionality. In addition, the models themselves must of course be accurate and conform to the requirements of the system so that the required product is built and that it is built correctly [3]. A software development process which embraces software verification and validation (V&V) can help delimit these problems.

2.3. Software verification and validation

The terms verification and validation are easily mixed up due to their similar spelling and signification but there is a difference concerning their area of responsibility. Software verification implies assurance that the software is fully functional and fault free whereas software validation is to assure that the software functionality is in accordance with its requirements [3, 20, 21]. But the distinction between the words is of less importance as they are to be used concurrently to complement each other throughout the development life cycle [22, 23]. Both verification and validation of the software should be considered as early as possible in the development process since defects as well as incorrect requirements found early save time and increase the chances of satisfying the customer in the end. And to assure that changes, e.g. of requirements, that take place in later phases does not cause problems, V&V should of course be considered in all development phases and not just the early ones [3, 20, 21].

It is noticeable with all types of V&V that one approach is insufficient for covering all possible problem areas and consequently there are several ways to verify and validate software. Sommerville [3] and Schultmeyer et al [21] mention static and dynamic V&V which structures the artifacts of a software
system into non-executable statically checked artifacts and executable dynamically checked artifacts. Static V&V is conducted through inspections of software artifacts such as requirements specifications and UML diagrams, but can also be applied to source code even though it can be executable and thus dynamically examinable as well. Inspections can be carried out in a number of ways with a number of participants. In the analysis phase the requirements of the software to develop are elicited and a software requirements specification (SRS) is put together [20]. This document must be inspected by the customer to assure that its content is correct, unambiguous, traceable etc. The same goes for the design documents and UML models in the design phase and the source code in the implementation phase to assure traceability from the models to the SRS, syntactical correctness of the code and logical conformance to the design models etc. Inspection of models and source code does not often include the customer however. Source code inspections are rather conducted either alone by each developer who reviews his/her own code or in a group of various size which reviews the code together and asks the developer questions about different parts of the code. Ideally a group inspection includes developers not involved with the project as well as those who are, so that new ideas and less subjective minds can help increase the quality of the code.

2.4. Reading techniques

Code reviews, as well as all other inspection types, should always follow a well defined specification, which differs depending on which reading technique that is used. Sommerville [3] and Schultmeyer et al [21] suggest that code reviews are to be conducted using the checklist-based reading technique (CBR) where the reader follows a checklist of common programming errors and issues that use to cause problems. The checklist shall of course be updated as new errors are found to keep it up-to-date. CBR is the standard inspection reading technique for software organizations today [24], but a series of experiments on reading techniques conducted by Thelin et al [24, 25, 26], which are summarized and discussed in [27], shows that it is not always the most efficient one. One of their experiments [24] shows that usage-based reading (UBR), where a set of prioritized use-cases are followed, is a much more efficient technique than CBR regarding finding the most critical faults from a user perspective. Similar results are shown in an investigation by Biffl et al [28] where CBR was compared to scenario-based reading (SBR) and led to the conclusion that SBR found more critical faults than CBR. SBR in that experiment was a mix of the two scenario-based techniques of perspective-based reading (PBR), in which the inspected artifact is inspected by different stakeholders each following specific scenarios based on their perspectives, and traceability-based reading (TBR) which is used to examine OO design specifications [24]. Another experiment conducted by Sabaliauskaite et al [29] compared CBR with PBR and concluded that PBR is as efficient as CBR in finding faults but that it does it in less time. Another reading technique worth mentioning is defect-based reading (DBR), which focuses on specific types of faults but mixes CBR checklists with scenarios [24].

2.5. Software testing

For artifacts of the system that can be executed, i.e. software code, dynamic V&V includes all kinds of software testing such as unit tests, integration tests and acceptance tests [3]. A unit test is a test of a small, separable part of the code, typically a single method [30] to ensure that it functions as intended and returns the correct answer given a certain input. Once all methods have been tested individually their integration with other tested methods within the same class can be tested. Then follows integration tests of tested classes, components and sub-systems until all parts of the entire application have been tested with each other. Naturally, whenever a defect is found and removed in a test, all previous tests must be run again to prevent the changes made to correct the defect from introducing new defects in other already tested parts of the code [3]. Once all parts have been tested the running application is tested in a so-called acceptance test where the intended functionality stated in the SRS is checked by the developers and finally by the end users.

The number of different V&V approaches, of which only a few are mentioned here, might all seem tempting to use but not all are appropriate for all kinds of investigations. The same goes for different approaches to UML and code generation. Therefore, the next section covers the experimental setup, which explains the process of choosing the evaluation code, transforming it into UML diagrams, generating new code and deciding on V&V approaches applicable to the investigation. Finally it discusses any threats to the study and how to restrict their effect.

3. Experimental setup

The less detailed version of the experiment would describe how it intends to choose an already existing program, model it with UML diagrams of which code then is generated to be compared to the original one.
Though easily and quickly described, it does however require much more attention to details for its realization. Hence, the following sections discuss each step of the process, different paths to choose and which that was chosen.

### 3.1. Choice of evaluation code

The choice of code to use for the evaluation needs consideration for a number of reasons, the quality of the code being the most important one. An experiment based on faulty code is not worth much, thus one criterion of the code is that it must have been thoroughly verified and validated. That is easily controlled when using one’s own code but quite tricky otherwise since the process of V&V rarely is exposed outside of organizations as that might reveal secret business matters as well as signs of weaknesses to the public, especially to shareholders or potential shareholders. Therefore it is hard to know exactly how thorough the process has been when choosing between third party software. Instead one must rely on less valid proofs like programs well-known and already used by many and programs upgraded a couple of times, hence implying the detection and correction of faults.

Another characteristic to consider is that of the programming language in which the code is written. Naturally it must be an OO language since it is to be modeled with UML which, as mentioned above, is a notation for OO analysis and design. A number of experiments with UML [18, 31, 32] have used the Java programming language in their evaluations and it is a good choice since it has been around since 1995 [33] and therefore is established enough to be supported by most UML tools today [4]. Furthermore, since it has been around since before UML, it is more likely to have influenced the notation than more recent languages.

The code chosen in this investigation is selected parts of the open source text editor jEdit [34] which is written in Java. It has been under development for more than five years with continuous improvements and upgrades that have led to the current stable version of jEdit 4.1, which is also the version used in this investigation. Moreover, it is a well-known editor and as it is open source many people have had the chance to influence its improvement over the years which increases the possibility of some kind of V&V process taking place.

The jEdit program has many features, e.g. “built in macro language, plugins and syntax highlighting for more than 80 languages” [34] to mention some. The code consists of 1200 classes in 682 Java files and consequently the entire program cannot be used in the investigation as it would take a considerable amount of time to model. Luckily enough the program is divided into a number of packages, each containing a smaller amount of classes, and this investigation focuses on one of these packages.

Several things must be taken into consideration when choosing package such as the amount of classes in it, what type of classes it contains, e.g. GUI, I/O etc, and how many external packages and classes it depends on. The ideal package contains 10-15 classes to increase the chances of variation in the code. At the same time that is an amount small enough to manage in UML models and consequently the package ought to have as few out of package dependencies as possible to keep the models maintainable. Therefore it is wise to stay away from graphical interface classes or I/O classes as they tend to involve extensive dependencies of other classes. Once a couple of package candidates have been picked out, their dependencies are compared and the package with least external dependencies is chosen. The chosen package of this investigation is the syntax package, which contains 15 classes in 13 Java files of 3292 lines of code (LOC), as is presented in Appendix A. The files are of various sizes and of various complexities, hence making them interesting to model with UML as they are likely to bring about many different modeling aspects.

### 3.2. UML diagram setup

Once the evaluation code has been chosen it is to be modeled in UML diagrams based on the current official version of the UML notation, namely version 1.5 [5]. There is a variety of tools for UML modeling [4] of which some support both reverse and forward engineering automatically. But as the primary purposes of the tools are not to follow the UML notation at all costs but rather to facilitate software modeling, they might add extra features to the models. Therefore, this investigation will be conducted with a manual approach to both reverse and forward engineering. Hence, tools will only be used as long as they can provide models in accordance with the notation and consequently all information will be entered into the models manually to assure that the notation is followed, since it is the subject of investigation.

For the kind of experiment described in this paper two diagram types are necessary: a class diagram, which displays the classes with their variables and methods, and sequence diagrams for each class showing what happens inside each method when it is called. The class diagram is created by following the list of steps displayed in Table 1 to assure that no information is forgotten in the model.
Table 1. Class diagram setup list

<table>
<thead>
<tr>
<th>Step</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create the syntax package</td>
</tr>
<tr>
<td>2</td>
<td>Add classes to the package, each with full definition</td>
</tr>
<tr>
<td>3</td>
<td>Create other packages and classes based on import statements</td>
</tr>
<tr>
<td>4</td>
<td>Draw associations for each syntax class</td>
</tr>
<tr>
<td>5</td>
<td>Add all variables as attributes</td>
</tr>
<tr>
<td>6</td>
<td>Add all methods as operations</td>
</tr>
</tbody>
</table>

First the syntax package is created and its classes are added. Then other packages on which the syntax package depends are created with their classes based on the import statements of each class and the class definitions. When all packages and classes are created their associations are added with different connectors for different kinds of associations. Though there might be enough associations between classes to draw close to a spider web like picture, it should not be done unless necessary to convey associations which may otherwise be lost. Associations such as inheritance from other classes or implementations of interfaces must of course be displayed in the diagram, and the same goes for associations to other packages which need to be imported. But less obvious associations such as variable types need not be modeled with association connectors if they are displayed in the attribute description. Therefore the next step is to extend the class models by adding their variables and methods, which in UML, and henceforth in this paper, is defined as attributes and operations. In UML only attributes defined outside of operations are displayed in the class models. Operation attributes are not visible in the class diagram except for those defined in the list of operations as parameters in the operation signature. Operation attributes can be displayed in sequence diagrams though if they in some way trigger operation calls.

The sequence diagrams are created with one diagram for each class following the list of steps of Table 2.

Table 2. Sequence diagram setup list

<table>
<thead>
<tr>
<th>Step</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Set up a new diagram for the class and add the class to it</td>
</tr>
<tr>
<td>2</td>
<td>Add an operation call to the first operation of the class</td>
</tr>
<tr>
<td>3</td>
<td>Sequentially add the operation’s calls to other operations or return statements</td>
</tr>
<tr>
<td>4</td>
<td>When necessary, add other classes or objects to the model</td>
</tr>
<tr>
<td>5</td>
<td>Repeat step 2-4 for all operations of the class</td>
</tr>
<tr>
<td>6</td>
<td>Repeat step 1-5 for all classes of the package</td>
</tr>
</tbody>
</table>

The class is added to the model followed by a call to an operation in the class. Thereafter any actions taking place in the operation that involve other operation calls or return statements to the calling class are added to the diagram sequentially. This is repeated for all operations in the class until everything that can be modeled has been. Once all models are created new code shall be generated from them, which is described in the next section.

3.3. Code generation

As with the diagram setup, the code generation is done manually as well and starts with the code generation from the class diagram since it defines the skeleton of the system. Table 3 describes the exact procedure of turning an UML class diagram into code.

Table 3. Code generation list based on UML class diagrams

<table>
<thead>
<tr>
<th>Step</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create a Java file for each class. Inner classes are placed in the same file as their owner. (step 8)</td>
</tr>
<tr>
<td>2</td>
<td>Add package definition with the preceding word &quot;package&quot;</td>
</tr>
<tr>
<td>3</td>
<td>Add import statements based on associations in the diagram. Imports of whole packages are succeeded by &quot;.*&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Add class name, visibility and modifiers</td>
</tr>
<tr>
<td>5</td>
<td>Add class inheritance: If a class generalizes another class it is added as &quot;extends&quot; in the code If a class realizes an interface it is added as &quot;implements&quot; in the code</td>
</tr>
<tr>
<td>6</td>
<td>Add all attributes in full definition</td>
</tr>
<tr>
<td>7</td>
<td>Add all operation signatures in full definition</td>
</tr>
<tr>
<td>8</td>
<td>Add any inner classes and repeat step 4-7</td>
</tr>
</tbody>
</table>

Each class is assigned a Java file of its own, except for inner classes who are placed in the same file as its parent. The first thing added to the file however is the package name of which the class belongs. Then follows all import statements based on the associations between the class and other classes before the actual class definition is created. The definition consists of the class name preceded by visibility and modifiers and is thereafter followed by inheritance of other classes or interfaces. Then the rest of the information available in the diagram for that class is added to the code in the specified order, namely attributes first followed by operation signatures. The attributes and operations are naturally extended with visibility and modifiers too. Finally, any inner classes are added at the very end of
the owner class and the same procedure is repeated for the inner classes as well. When all classes of the syntax package have been defined in Java files along with their attributes and operations, it is time to generate code from the sequence diagrams to define as much as possible of the operation bodies.

Just as with the code generation of the class diagram, the generation based on the sequence diagrams follows a list of steps, which is displayed in Table 4. As sequence diagrams display information in the order it is to be executed, the code generated from these models is easily placed in the file by sequentially adding the actions that take place within an operation to the operation body in the corresponding Java file.

Table 4. Code generation list based on UML sequence diagrams

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Go through the sequence diagrams for each class</td>
</tr>
<tr>
<td>2</td>
<td>Choose the first operation called upon</td>
</tr>
<tr>
<td>3</td>
<td>Add messages, i.e. operation calls or return statements, sequentially to the operation body in the Java file</td>
</tr>
<tr>
<td>4</td>
<td>Repeat step 2-3 for all operations of the class</td>
</tr>
<tr>
<td>5</td>
<td>Repeat step 1-4 for all classes</td>
</tr>
</tbody>
</table>

Once all information of the UML diagrams has been forward engineered into code, the generated code shall be compared to the original and its result shall be discussed. But to assure that the result is trustworthy all parts of the investigation so far must have been continuously verified. Consequently, the next section describes the V&V process of this investigation.

3.4. V&V of the investigation

The previous sections have described the procedure of how the result of a code comparison is to be reached but it mentions less about the V&V process conducted in parallel to delimit mistakes which might affect the result. Even though the lists of Table 1-4 prevent several mistakes being realized as the setup follows well defined steps, the tables themselves might contain errors such as missing steps or lack of details. Therefore the setup of the experiment described above has been verified by the use of checklists (Appendix B-D) in inspections of the diagrams created and the code generation.

Though other reading techniques such as UBR and SBR are considered more efficient [27] than CBR in many ways, CBR is the best suited reading technique for this investigation. UBR and SBR are based on use-cases and scenarios to find errors that are most important from a user’s point of view, but as only a small part of the jEdit program is used in the investigation, neither use-cases nor scenarios are available. In addition, the purpose of the inspections is simply to make sure that all parts of the code have been modeled correctly and that all parts of the models thereafter have been forward engineered correctly. Hence, a user’s point of view is less important here as the inspections rather focus on everything being as similar to the original artifact as possible.

CBR is also used to compare the generated code to the original, following a code review checklist (Appendix E). The checklist used in the comparison does not only look for resemblance to the original code though but investigates both logical and functional resemblance and diversity as well as maintainability of the code, i.e. what it looks like. This comparison along with experiences and impressions from previous parts of the investigation will all add up to a discussion of UML modeling possibilities including what can be modeled, what cannot and why, but most importantly a discussion of the quality of the code generated.

Testing is not used in this investigation however due to lack of testing techniques, and technology for that matter, efficient enough to evaluate models and code from the perspectives applicable in the experiment presented here. Conducting tests on the code comparison for instance would require some kind of artificial intelligence to understand e.g. that some code parts may be logically different but functionally equal, but unfortunately such intelligence is yet to be developed or at least not available to this investigation.

Even the most thorough V&V process cannot avoid all problems possible and some problems cannot always be avoided at all. But acknowledging their existence and thinking about preventing actions might at least delimit their effect somewhat. Thus, the next section discusses the threats of the investigation.

3.5. Threats to the study

Since software in itself is hard to perfect due to its abundance of syntactical details that must be correct for it to work, the number of errors to make is naturally higher in studies concerning software than in other computer related studies that does not. That is a threat of its own but a more obvious, yet related one, is that of misinterpretations of the evaluation code. One purpose of the experiment is to evaluate how similar the original code is to the generated one. This is a major threat to the study since even small errors when modeling the code might lead to big diversions. And even if the interpretation of the code is correct, the modeling of the code is still a threat to the study as the UML diagrams require just as much attention to detail
as software does for them to be correct. These threats can be partly avoided through the use of a thorough V&V process. In this investigation the threats are delimited by the use of step-by-step lists (Table 1-4) for the setup of each artifact followed by inspections conducted using the well defined check lists of Appendix B-E.

Another threat which is much harder to control is that the choice of code might turn out to be bad, e.g. that the code is faulty or that it is hard to model due to its character. The effect of faulty code however is of less importance to the code comparison since the quality of the generated code only is to be measured against the original. Consequently, any errors in the original should be passed on to the models and from there to the generated code as long as they are not e.g. misspellings of language keywords which sometimes cannot be modeled faulty according to UML syntax.

The tools used in the investigation impose a threat of their own as they might be inefficient and unable to model everything exactly in accordance with the UML notation [5]. This problem though is easily solved by simply not using UML tools unless they do follow the notation, and in worst case that might mean drawing parts or everything “by hand” in a drawing program.

A more severe risk however is that both the reverse and forward engineering is conducted by the same person which might lead to mistakes such as things being added to the generated code that is not in the models due to knowledge of the original code. But as the original code consists of more than 3000 LOC (Appendix A) it is nearly impossible to memorize even smaller parts of it. In addition, the code generation is based only on the UML diagrams created and thus the original code is not even looked at until the code generation is done and has been verified.

Now that the practical approach of the investigation has been described it is time to address the result of it all in the following section. There the UML diagrams created and the generated code are described followed by the code comparison and its results.

4. Results

The results of the investigation consist of the UML class diagram (Appendix F-G) displaying the overall picture of the syntax package, the UML sequence diagrams (Appendix H) that capture parts of the operation bodies of each class and then the new code (Appendix I) generated from the diagrams. The results also include a comparison of the original code to the generated one. But first, the next section will describe the models created in the UML diagram setup and discuss what could and could not be modeled in UML.

4.1. UML diagrams

The steps described in Table 1 of section 3.2 was followed to create the class diagram (Appendix F) which displays the syntax package and its classes surrounded by packages and classes imported by them. To keep the class diagram as uncluttered as possible it does not display class details such as attributes and operations. These details are instead individually presented in Appendix G. The creation of the class diagram did not bring about much difficulty as the notation has well defined support for class and package representation. The only thing found in this experiment that is missing in the notation concerning class diagrams is how to visualize the visibility, i.e. public, private or protected, and the static modifier of classes. The notation describes in detail how it is possible to fully model visibility and modifiers of attributes and operations but mentions less of the kind concerning classes. Therefore the following code fragment would result in the class model of Figure 1.

```java
public class A {
    public String name;
    protected int amount;
    private int number;

    public static final int getId() {
        return id;
    }
}
```

For modifiers of classes, the notation only addresses how to model that a class is abstract or final, as seen in Figure 2, but mentions nothing of the display of e.g. a static class.

```
abstractClass
```

```
finalClass
{leaf}
```

Figure 1. A class model displays visibility and modifiers of attributes and operations but not for the class itself

Figure 2. Abstract and final class in UML
When the class visibility is not defined the class is considered to have so-called package visibility [35] which means that only other classes of the same package can see it. This is not the case when a class is public as it then is seen by all classes in the program, regardless of package. The problem is even more severe though if some classes are meant to be private, e.g. for security reasons, and thus should not be accessible even by other classes of the same package. This could easily be solved by allowing the model to add a visibility sign before the class name, but that is not the case with the current official UML notation [5].

The creation of the sequence diagrams, which followed the step-by-step list of Table 2, presented more modeling problems, which is natural as the complexity of the code usually is greater inside of operations. One thing found was that there is no standard for try-catch statements. It is possible to model exceptions in UML but only as return statements from the class called in the try-statement. Thus this code would be modeled as in Figure 3.

```
try
    title = b.getTitle();
} catch (Exception e)
    error("no-title");
```

![Figure 3. It is not possible to fully model a try-catch statement in UML. The exception is modeled as a return statement which it is up to the receiving class to deal with.](image)

As the figure explains, the exception is simply returned back to the calling class and it is up to that class to do something about it. The try-statement is not at all visible in the model.

The next thing found during modeling of the sequence diagrams was that if-statements can cause quite a modeling problem. A single if-statement is not a problem to model but when a series of if-statements are nested each statement must be modeled by its full conditional path. Furthermore, the notation does not present a way to model if a conditional statement should be an else-statement which then must be modeled either as the opposite of the preceding if-statements on the same level or without any conditional information at all. The opposite approach works when there is only one if-statement to negate, but when it is preceded by a series of if-else statements it cannot be done without negating all of the preceding statements. Then it is better to leave the condition out completely for that level. To illustrate the problem of nested conditional statements, consider the following code example and its resulting model of Figure 4.

```
if (x == 0)
    { if (y != 13)
        { String name = b.getName();
            if (z)
                { int amount = b.getTotal();
                    }
        else
            { int number = b.getNumber();
                }
        }
    }
```

![Figure 4. Nested if-statements must be modeled with their full conditional path in sequence diagrams. Else-statements can be modeled by negating the preceding if-statement of the code.](image)

With long conditional declarations, which on top of it all are nested in several levels, it becomes quite a struggle to model them in a sequence diagram. Even though all if-statements do not contain operation calls and thus cannot be modeled, the ones that do will deteriorate the models if they are too complex. The same problem is faced with switch-case statements as the UML notation does not single them out from other conditional statements. This means that they too must be transformed to if-statements to be modeled.
Another thing partly overlooked in UML is return messages. When an operation call from one class to another results in a return value, it is modeled as a dashed arrow back to the calling class with the return value above it. But when a class calls an operation located within itself, e.g. calls to "this", UML does not support modeling of a prospective return value. For example, this code fragment is modeled as the message sequence of Figure 5.

```java
class A {
    ...b.getName();
    ...this.getId();
    getId()
    {
        return id;
    }
}
class B {
    getName()
    {
        return name;
    }
}
```

Figure 5. Return messages from calls to "self" or "this" cannot be modeled.

As sequence diagrams are made to picture the communication flow of a system it relies on all operation calls being modeled. But as this investigation goes from existing code to UML diagrams, opposed to the common way of modeling before implementing, programming shortcuts in the code have been found which cannot be modeled without tampering with the original code setup. One example of this is when operation calls are put as parameters in other operation calls. To model this in UML one would have to perform the parameter operation call first, store it in a variable and then insert the variable as a parameter in the other operation call. This has not been done in the models however and is not really a problem, but it is an interesting aspect though as such shortcuts are used more or less depending on code conventions etc.

Another related issue is the so-called static initiator [35] of the Java language which cannot be fully modeled in a sequence diagram. A static initiator, also called a class initiator, is a special way of initializing class variables. The example below is taken from the ParserRuleSet class of the syntax package and shows how a class variable can be initialized.

```java
private static ParserRuleSet[] standard;
static {
    standard = new ParserRuleSet[Token.ID_COUNT];
    for(byte i = 0; i < standard.length; i++)
    {
        standard[i] = new ParserRuleSet(null,null);
        standard[i].setDefault(i);
    }
}
```

Since the static initiator is not an operation it cannot be modeled as one. And as it is a class variable it could not be placed in e.g. the constructor of the class as its purpose is to initialize variables of a certain instance of the class [35]. Hence, its content in this case can be modeled in a sequence diagram as it contains operation calls, but its location in the code cannot be modeled.

The last thing found that could not be modeled in a sequence diagram was calls to the super-class from a sub-class using the keyword "super". All operation calls are aimed at the class containing the operation and therefore a call to super would become a call aimed at the super class with its real name. If for example class A extends the class C, a call to super.init() would be modeled as pictured in Figure 6.

Figure 6. Calls to a super-class using the keyword "super" is modeled as a call to the actual name of the super-class

One thing deliberately left out of the created UML diagrams, both the class diagram and the sequence diagrams, is the use of notes. The reason for this is that there is no definite specification for what is to be put in notes. The UML notation [5] allows anything to be put in notes and consequently the entire code of the syntax package could have been put in one. The notation mentions that notes for example can be used to describe operation bodies but recommends that this should only be done for shorter code passages. But as the notation is as vague as it is about notes they have been completely kept out of the models and are instead only discussed in section 5. But before that the
remaining results are to be accounted for, starting with the code generation in the following section.

4.2. Generated code

The code generation from the UML diagrams was conducted by following the steps described in the lists of Table 3 and 4 and resulted in the code presented in Appendix I. This did not result in even nearly the same amount of problems described in the previous section but was rather a matter of sequentially converting each part of the models into code. As the UML notation allows e.g. operation signatures to be modeled according to any programming language of choice, type definitions need not be converted. Instead the conversion included translating model syntax into Java syntax so that e.g. import dependencies to whole packages in the class diagram were appended with “.*” or that generalizations and realizations were translated to “extends” and “implements”. Another example is exceptions. As the previously described problem with modeling try-catch statements exists, any exceptions returned from a class in the sequence diagrams have been translated as “throw Exception” whereas exceptions returned back to a class are generated as “catch (Exception)”. Naturally UML specific things, such as variable types being modeled at the end of the message declarations, are placed in the correct order again in the code according to the programming language syntax.

The nested if-statements that had been modeled with full path conditions in the sequence diagrams came to be a series of long and complex if-statements. This along with the lack of if-else statements will result in a slower execution of the program as each statement must be checked instead of the original case where statements only are checked until the right condition is found. This is even more severe in this case where Java has been used as Java in itself reduces the performance of program executions by interpreting the programs [35]. Succeeding messages with the same conditional statements however have been put in the same if-statement in the generated code as it is obvious that these messages are on the same level.

Once the code had been fully generated and verified, as described in section 3.4, it was compared to the original code. The result of this comparison is addressed in the next section.

4.3. Code comparison

The generated code (Appendix I) was compared to the original (Appendix A) following the check list of Appendix E. The comparison addressed issues such as functionality, maintainability and performance as well as modeling possibility and is presented in Table 5.

The table shows that even though the functionality of the nested if-statements is the same as for the original code, the maintainability as well as the performance is decreased since the code is much harder to read and as it will slow down the program execution. The same goes for the switch-case structures as they must be modeled the same way. If-else statements and single else-statements are affected the same way in the generated code as each of them will be checked unlike the original code where the program breaks out of that conditional level once a condition is fulfilled.

As mentioned in previous sections, the functionality of try-catch statements is only partially equal to the original code since exceptions can be modeled in UML. The maintainability of the exceptions though is worse as exceptions are not handled properly anymore.

The succeeding parts of the table show brighter results since iterations and all operation features but operation bodies are equal to the original code. The operation bodies are only partially functional and the maintainability of them as well as performance is worse as all logic between operation calls is gone. The brighter results continue though, as all variable features except for operation variables and the static initiator is of equal functionality, maintainability and performance as the original code. The operation variables are part of the operation bodies and consequently this feature looses quality in the same areas. Furthermore, the static initiator can be modeled if it contains operation calls, but as its skeleton cannot be modeled it will never be executed in the generated code and is therefore unequal to the original in the remaining areas of the table.

Classes and interfaces have already been discussed in section 4.1 and as the table below shows it is only visibility and modifiers of their definition that cannot be modeled and that show non-equal results. But as the previous discussion explains, they are severe features of a program and thus ought to present better results than those visible as they can reduce system safety.

The last feature of the table is that of return statements and that operations called by their owner does not include the return values in the generated code as these cannot be modeled in UML.

Have these issues presented here been intentionally overlooked in the notation or are they just the result of an ever growing standard which cannot keep up with the increased demands of the software development society? When such an amount of problems are present as have been described in this and the previous sections, one cannot help but wonder why it is so. That question is discussed in the next section where the results are summarized and put into context of the real world.
<table>
<thead>
<tr>
<th>Feature</th>
<th>Occurrences</th>
<th>Functionality</th>
<th>Maintainability</th>
<th>Performance</th>
<th>Modeling possibility</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>if</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>single</td>
<td>43</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>nested</td>
<td>123</td>
<td>equal</td>
<td>↓</td>
<td>↓</td>
<td>✓</td>
</tr>
<tr>
<td><strong>else if</strong></td>
<td>62</td>
<td>equal</td>
<td>↓</td>
<td>↓</td>
<td>As multiple if-statements</td>
</tr>
<tr>
<td><strong>else</strong></td>
<td>38</td>
<td>equal</td>
<td>↓</td>
<td>↓</td>
<td>As if-statement or without condition</td>
</tr>
<tr>
<td>switch-case</td>
<td>5</td>
<td>equal</td>
<td>↓</td>
<td>↓</td>
<td>As multiple if-statements</td>
</tr>
<tr>
<td><strong>try-catch</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>try</td>
<td>5</td>
<td>non-equal</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>catch Exception</td>
<td>5</td>
<td>partially</td>
<td>↓</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td><strong>iterations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>for</td>
<td>9</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>while</td>
<td>7</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td><strong>operations</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>visibility</td>
<td>116</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>modifiers</td>
<td>14</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>name</td>
<td>116</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>parameters</td>
<td>223</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>throws exceptions</td>
<td>6</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>calls</td>
<td>215</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>bodies</td>
<td>116</td>
<td>partially</td>
<td>↓</td>
<td>↓</td>
<td>Method calls and return statements</td>
</tr>
<tr>
<td><strong>variables</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>visibility</td>
<td>149</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>modifiers</td>
<td>81</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>instance</td>
<td>111</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>class</td>
<td>38</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>operation</td>
<td>76</td>
<td>partially</td>
<td>↓</td>
<td>↓</td>
<td>If they equal method calls</td>
</tr>
<tr>
<td>static initiator</td>
<td>1</td>
<td>non-equal</td>
<td>-</td>
<td>-</td>
<td>If it contains operation calls</td>
</tr>
<tr>
<td><strong>class</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>visibility</td>
<td>14</td>
<td>non-equal</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>modifiers</td>
<td>1</td>
<td>partially</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>14</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>inheritance</td>
<td>6</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>import</td>
<td>35</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>package</td>
<td>13</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td><strong>interface</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>visibility</td>
<td>1</td>
<td>non-equal</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>modifiers</td>
<td>0</td>
<td>partially</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>name</td>
<td>1</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td><strong>return</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>from other class</td>
<td>133</td>
<td>equal</td>
<td>equal</td>
<td>equal</td>
<td>✓</td>
</tr>
<tr>
<td>from self</td>
<td>21</td>
<td>non-equal</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
5. Discussion

The experiment described in this paper has examined what can and cannot be modeled with UML by modeling parts of the existing jEdit text editor. Thereafter the models have been generated into new code and compared to the original to examine the code quality of UML models. The results have come to show that there are many missing modeling features in UML and in sequence diagrams in particular. An UML class diagram can model all aspects of class definitions but for visibility and modifiers of the class. The sequence diagrams however lack of modeling possibilities for a number of features where nested conditional statements and exception handling are the worst. Other problems encountered are programming language specific as the Java static initiator and cannot be fully blamed on the UML notation as it foremost is a general standard regardless of programming language. This however brings about the aspect of the purpose of UML and perhaps all things are not supposed to be modeled.

As discussed in the Background, UML is a notation for software analysis and design but not for implementation. Furthermore, the models are supposed to help visualize the system by displaying a general view in the class diagram and the flow of actions in the sequence diagrams. Thus, details such as try-catch statements and series of conditional statements maybe should not be modeled unless they bring clarity to the system overview. On the other hand, the clarity of those statements would improve if the notation had better support for their modeling. The experiment in this paper which contains 123 nested if-statements, as declared in Table 5, could not visualize the message flow without displaying these statements since the flow of this particular code is dependent on the values sent into the different operations. The severity of this increases though considering that the code modeled here only represents a small part of the entire jEdit program and that the total amount of nested if-statements would turn out to be a much bigger number.

Nevertheless, it is easier to see why these things have been partly overlooked in the notation. But notation errors such as return statements being visible in some cases but not in others are not as easily disregarded since that implies sort of a logical ambiguity of the notation. Thus, the notation is not always as strict as a standard ought to be as it seems to rather provide mediocre support for features that are not of high priority instead of ruling them out completely. Other parts that cannot be modeled in UML are easier disregarded such as the content of operations since that is part of the implementation phase in a software development cycle. So even though they can be modeled using notes, it really does not agree with the primary purposes of UML.

There is no justification though for the class diagram lacking of ways to model class visibility as that is an important feature considered already in the analysis of a software system. Safety critical classes should be modeled as such in the system overview to ensure that they are implemented later by the correct definition since the loss of such a definition allows all classes within a package to access each other.

Another issue which must be attended in this discussion is whether or not the evaluation program used was based on UML models or not during its implementation. There is no mentioning of this on the program web site [34] but it does however reveal that the program core has been developed primarily by one person whereas succeeding development involves developers of various developing teams. And as the program is open source it allows for anyone to give suggestions for its improvement and if the suggestions are good they are implemented. Based on this it is obvious that upgraded parts of the jEdit program cannot have been modeled before their implementation, but no such judgment can be made of the early system core. Had UML been used for all parts of the system though it is likely that the program today would present a code quite different from the actual one.

6. Conclusion

The purpose of this paper was to examine the quality of code based on UML models by reverse engineering already existing code into models from which new code then was forward engineered. The experiment conducted shows that there are both minor and major drawbacks of the UML notation where class visibility being left out of the models, and consequently the generated code, being of the more severe kind. Loss of class visibility aggravates the system analysis and may jeopardize security as safety critical classes cannot be modeled as such. In extensive systems such information can be easily forgotten during implementation if not present in the design models.

Furthermore, the notation has only partial support for nested conditional statements, which are sometimes required to display the flow of the modeled system. But to model this in UML each statement must be modeled by its full conditional path, which can become quite a struggle if the number of statements is great. This brings about recurrences of information which then are passed on from the models to the code, thus deteriorating the program performance in the end. Moreover, the notation is only partially supportive of
the representation of return statements and exception handling. Even though some of these features may not entirely belong in the analysis and design phase, it seems illogical that the notation supports some features mediocre rather than not supporting them at all, especially if they do not fit into the notation intentions.

These and other drawbacks of the notation are partly compensated for by its advantages, and several modeling setbacks can be disregarded as they lay on the outer borders of the notation purposes. Yet, the most severe problems lay closer to the core and cannot be as easily disregarded.

7. Future work

A logical continuation of the investigation presented in this paper would be to perform the same experiment using code of other programming languages as that would present a more widespread view of the notation possibilities as well as its shortages. In addition, it would then be interesting to choose an evaluation program known to have been modeled with UML during its development to see if it would generate other aspects than those covered here. The ultimate investigation would conduct the experiment using two evaluation programs of highly similar functionality where only one of them initially had been modeled, to bring about more concrete code diversities.

As most software analysis and design is done using different tools a related research area would be to conduct the same experiment using a set of different UML tools with the purpose of examining how well the tools live up to the UML notation. This could be done by manually setting up the UML diagrams in the tool environment and then use the built in forward engineering features to generate code. It could also be done by letting the tools taking care of both reverse and forward engineering to examine the translation of the code into diagrams and the diagrams into code.

8. Acknowledgements

I would like to thank my examiner, Stefan Christiernin, for his constant support and inspiration. You have always spoken your mind when not content with the work presented and even though I have been quite stubborn sometimes you have not given in. This has enhanced both the way I look at things as well as the paper.

I would also like to thank my advisor, Andreas Boklund, for answering all my questions concerning UML, but also for some very inspiring discussions about the past and future of software engineering. This too has been a great influence on my thoughts and writing and consequently on the end result.

9. References


[34] jEdit - Open Source programmer's text editor http://www.jedit.org/ (2004-04-02)

package org.gjt.sp.jedit.syntax;

package org.gjt.sp.jedit.syntax;

import javax.swing.text.*;
import java.awt.font.*;
import java.awt.geom.*;
import java.awt.*;
import org.gjt.sp.jedit.syntax.*;

//{{{ paintChunkList() method
/**
 * Paints a chunk list.
 * @param lineText The line text
 * @param chunks The chunk list
 * @param gfx The graphics context
 * @param x The x co-ordinate
 * @param y The y co-ordinate
 * @param background The background color of the painting area,
 * used for the background color hack
 * @return The width of the painted text
 * @since jEdit 4.1pre1
 */
public static float paintChunkList(Segment lineText, Chunk chunks, Graphics2D gfx, float x, float y, Color background, boolean glyphVector)
{
    FontMetrics forBackground = gfx.getFontMetrics();

    float _x = 0.0f;
    for(;;)
    {
if (chunks == null)
    return _x;

//{{{( find run of chunks with the same token type
Chunk start = chunks;
        float width = 0.0f;
        int length = 0;
        while(chunks != null
            && start.style == chunks.style
            && (start.visible == chunks.visible)
            && (start.accessable == chunks.accessable))
    {
        length += chunks.length;
        width += chunks.width;
        chunks = (Chunk)chunks.next;
    } }}}

// Useful for debugging purposes
if (DEBUG)
    {
        gfx.draw(new Rectangle2D.Float(x + _x, y - 10,
                width,10));
    }
if (start.accessable)
    {
        gfx.setFont(start.style.getFont());
        gfx.setColor(start.style.getForegroundColor());
        if (glyphVector & start.gv != null & start.next == chunks)
            gfx.drawGlyphVector(start.gv,x + _x,y);
        else
            {
                gfx.drawChars(lineText.array,
                        lineText.offset
                        + start.offset,length,
                        (int)(x + _x),(int)y);
            }
        }
    }
    _x += width;
package and File

// for return statement see top of for() loop...
} //}}}

//{{{ offsetToX() method
/**
 * Converts an offset in a chunk list into an x co-ordinate.
 * @param chunks The chunk list
 * @param offset The offset
 * @since jEdit 4.1pre1
 */
public static float offsetToX(Chunk chunks, int offset)
{
    if(chunks != null && offset < chunks.offset)
    {
        throw new ArrayIndexOutOfBoundsException(offset + " < " + chunks.offset);
    }

    float x = 0.0f;

    while(chunks != null)
    {
        if(chunks.accessable && offset < chunks.offset + chunks.length)
            return x + chunks.offsetToX(offset - chunks.offset);

        x += chunks.width;
        chunks = chunks.next;
    }

    return x;
} //}}}

//{{{ xToOffset() method
/**
 * Converts an x co-ordinate in a chunk list into an offset.
 * @param chunks The chunk list
 * @param x The x co-ordinate
 * @param round Round up to next letter if past the middle of a letter?
 * @return The offset within the line, or -1 if the x co-ordinate is too far to the right
 * @since jEdit 4.1pre1
 */
public static int xToOffset(Chunk chunks, float x, boolean round)
{
    float _x = 0.0f;

    while(chunks != null)
    {
        if(chunks.accessable && x < _x + chunks.width)
            return chunks.xToOffset(_x - x, round);

        _x += chunks.width;
        chunks = (Chunk)chunks.next;
    }

    return -1;
} //}}}

//{{{ Instance variables
public boolean accessible;
public boolean visible;

public boolean monospaced;
public float charWidth;

// set up after init()
public SyntaxStyle style;
public float width;
public GlyphVector gv;

//{{{
public Chunk(float width, int offset, ParserRuleSet rules)
{
    super(Token.NULL, offset, 0, rules);
    this.width = width;
} //}}}

public Chunk(byte id, int offset, int length, ParserRuleSet rules)
{
    super(id, offset, length, rules);
    accessible = true;
} //}}}

public final float[] getPositions()
{
    if (gv == null)
        return null;

    if (positions == null)
        positions = gv.getGlyphPositions(0, length, null);

    return positions;
} //}}}

public final float offsetToX(int offset)
{
    if (!visible)
        return 0.0f;
    else if (monospaced)
        return offset * charWidth;
    else
        return getPositions()[offset * 2];
} //}}}

public final int xToOffset(float x, boolean round)
{
    if (!visible)
    {
        if (round && width - x < x)
            return offset + length;
        else
            return offset;
    }
    else if (monospaced)
    {
        x = Math.max(0, x);
        float remainder = x % charWidth;
        int i = (int)(x / charWidth);
        if (round && remainder > charWidth / 2)
            return offset + i + 1;
        else
            return offset + i;
    }
    else
    {
        float[] pos = getPositions();
        for (int i = 0; i < length; i++)
        {
float glyphX = pos[i*2];
float nextX = (i == length - 1
    ? width : pos[i*2+2]);

if(nextX > x)
{
    if(!round || nextX - x > x - glyphX)
        return offset + i;
    else
        return offset + i + 1;
}

// wtf?
return -1;
} //}}}

//{{{ init() method
public void init(Segment seg, TabExpander expander, float x,
    SyntaxStyle[] styles, FontRenderContext fontRenderContext,
    byte defaultID, float charWidth)
{
    style = styles[(id == Token.WHITE || id == Token.TAB)
        ? defaultID : id];

    if(length == 1 && seg.array[seg.offset + offset] == '\t')
    {
        visible = false;
        float newX = expander.nextTabStop(x,offset + length);
        width = newX - x;
    }
    else if(charWidth != 0.0f)
    {
        visible = monospaced = true;
        this.charWidth = charWidth;
        width = charWidth * length;
    }
    else
    {
        visible = true;
        String str = new String(seg.array,seg.offset + offset,length);
        gv = style.getFont().createGlyphVector(
                fontRenderContext,str);
        width = (float)gv.getLogicalBounds().getWidth();
    }
} //}}}

//{{{{ Private members
private float[] positions;
//}}}}
}
package org.gjt.sp.jedit.syntax;

/**
 * Builds a linked list of tokens without any additional processing.
 * @author Slava Pestov
 * @version $Id: DefaultTokenHandler.java,v 1.14 2003/02/07 21:57:43 spestov
 * Exp $ @since jEdit 4.1pre1
 */

public class DefaultTokenHandler implements TokenHandler
{

    /*{ reset() method */
    /**
     * Clears the list of tokens.
     */
    public void init()
    {
        lastToken = firstToken = null;
    }

    /*{ getTokens() method */
    /**
     * Returns the first syntax token.
     * @since jEdit 4.1pre1
     */
    public Token getTokens()
    {
        return firstToken;
    }

    /*{ handleToken() method */
    /**
     * Called by the token marker when a syntax token has been parsed.
     * @param id The token type (one of the constants in the
     * {@link Token} class).
     * @param offset The start offset of the token
     * @param length The number of characters in the token
     * @param context The line context
     * @since jEdit 4.1pre1
     */
    public void handleToken(byte id, int offset, int length,
                            TokenMarker.LineContext context)
    {
        Token token = createToken(id,offset,length,context);
    }
}
if (token != null)
    addToken(token, context);
} ///}}

//{{{{ Protected members
protected Token firstToken, lastToken;

//{{{{ getParserRuleSet() method
protected ParserRuleSet getParserRuleSet(TokenMarker.LineContext context)
{
    while (context != null)
    {
        if (context.rules.getMode() != null)
            return context.rules;
        context = context.parent;
    }
    return null;
} ///}}}

//{{{{ createToken() method
protected Token createToken(byte id, int offset, int length,
    TokenMarker.LineContext context)
{
    return new Token(id, offset, length, getParserRuleSet(context));
} ///}}}

//{{{{ addToken() method
protected void addToken(Token token, TokenMarker.LineContext context)
{
    if (firstToken == null)
    {
        firstToken = lastToken = token;
    }
    else
    {
        lastToken.next = token;
        lastToken = lastToken.next;
    }
} ///}}}

//}}}}}
/*
 * DisplayTokenHandler.java - converts tokens to chunks
 * :tabSize=8:indentSize=8:noTabs=false:
 * :folding=explicit:collapseFolds=1:
 * Copyright (C) 2002 Slava Pestov
 *
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */

package org.gjt.sp.jedit.syntax;

//{{{ Imports
import javax.swing.text.*;
import java.awt.font.*;
import org.gjt.sp.jedit.syntax.*;
//}}}

/**
* Creates {link Chunk} objects that can be painted on screen.
* @since jEdit 4.1pre1
*/
public class DisplayTokenHandler extends DefaultTokenHandler {

//{{{ init() method
    public void init(Segment seg, SyntaxStyle[] styles,
                      FontRenderContext fontRenderContext,
                      TabExpander expander)
    {
        super.init();

        x = 0.0f;

        this.seg = seg;
        thisstyles = styles;
        thisfontRenderContext = fontRenderContext;
        thisexpander = expander;
    } //}}}

//{{{ setMonospacedCharWidth() method
    public void setMonospacedCharWidth(float charWidth)
    {
        this.charWidth = charWidth;
    } //}}}

//{{{ getChunks() method
    /**
     * Returns the first chunk.
     * @since jEdit 4.1pre1
     */
    public Chunk getChunks()
    {
        return (Chunk)firstToken;
    } //}}}

//}}{

//{{{ Protected members

}}}}}
protected Segment seg;
protected SyntaxStyle[] styles;
protected FontRenderContext fontRenderContext;
protected TabExpander expander;
protected float x;
protected float charWidth;

/** createToken() method */
protected Token createToken(byte id, int offset, int length,
TokenMarker.LineContext context)
{
    if (id == Token.END)
        return null;

    Chunk chunk = new Chunk(id, offset, length, getParserRuleSet(context));
    chunk.init(seg, expander, x, styles, fontRenderContext,
               context.rules.getDefault(), charWidth);

    x += chunk.width;

    return chunk;
} /*}}}*/

/*}}}*/
package org.gjt.sp.jedit.syntax;

/**
 * A dummy token handler that discards tokens.
 * @author Slava Pestov
 * @version $Id: DummyTokenHandler.java,v 1.4 2003/02/07 21:57:43 spestov Exp $
 * @since jEdit 4.1pre1
 */

public class DummyTokenHandler implements TokenHandler {

    /**
     * To avoid having to create new instances of this class, use
     * this variable. This is allowed because instances of this
     * class do not store any state.
     */

    public static final DummyTokenHandler INSTANCE = new DummyTokenHandler();

    //{{ handleToken() method

    /**
     * Called by the token marker when a syntax token has been parsed.
     * @param id The token type (one of the constants in the
     *     (Token) class).
     * @param offset The start offset of the token
     * @param length The number of characters in the token
     * @param context The line context
     * @since jEdit 4.1pre1
     */

    public void handleToken(byte id, int offset, int length,
                             TokenMarker.LineContext context) {}
import javax.swing.text.Segment;
import java.util.Vector;
import org.gjt.sp.jedit.TextUtilities;

/**
 * A <code>KeywordMap</code> is similar to a hashtable in that it maps keys
to values. However, the "keys" are Swing segments. This allows lookups of
text substrings without the overhead of creating a new string object.
 *
 * @author Slava Pestov, Mike Dillon
 * @version $Id: KeywordMap.java,v 1.5 2002/06/05 02:13:56 spestov Exp $
 */

public class KeywordMap
{

    //{{{ KeywordMap constructor
    /**
     * Creates a new <code>KeywordMap</code>.
     * @param ignoreCase True if keys are case insensitive
     */
    public KeywordMap(boolean ignoreCase)
    {
        this(ignoreCase, 52);
        this.ignoreCase = ignoreCase;
        noWordSep = new StringBuffer();
    } //}}}

    //{{{ KeywordMap constructor
    /**
     * Creates a new <code>KeywordMap</code>.
     * @param ignoreCase True if the keys are case insensitive
     * @param mapLength The number of "buckets" to create.
     * A value of 52 will give good performance for most maps.
     */
    public KeywordMap(boolean ignoreCase, int mapLength)
    {
        this.mapLength = mapLength;
        this.ignoreCase = ignoreCase;
        map = new Keyword[mapLength];
    } //}}}

    //{{{ lookup() method
    /**
     * Looks up a key.
     */
    public int lookup(Segment key)
public byte lookup(Segment text, int offset, int length) {
    if(length == 0)
        return Token.NULL;
    Keyword k = map[getSegmentMapKey(text, offset, length)];
    while(k != null)
    {
        if(length != k.keyword.length)
        {
            k = k.next;
            continue;
        }
        if(TextUtilities.regionMatches(ignoreCase, text, offset, k.keyword))
            return k.id;
        k = k.next;
    }
    return Token.NULL;
}

public void add(String keyword, byte id) {
    int key = getStringMapKey(keyword);
    char[] chars = keyword.toCharArray();
    // complete-word command needs a list of all non-alphanumeric
    // characters used in a keyword map.
    loop: for(int i = 0; i < chars.length; i++)
    {
        char ch = chars[i];
        if(!Character.isLetterOrDigit(ch))
        {
            for(int j = 0; j < noWordSep.length(); j++)
            {
                if(noWordSep.charAt(j) == ch)
                    continue loop;
            }
            noWordSep.append(ch);
        }
    }
    noWordSepStr = null;
    map[key] = new Keyword(chars, id, map[key]);
}

public String getNonAlphaNumericChars() {
    return noWordSep.toString();
}
public String[] getKeywords()
{
    Vector vector = new Vector(100);
    for (int i = 0; i < map.length; i++)
    {
        Keyword keyword = map[i];
        while (keyword != null)
        {
            vector.addElement(new String(keyword.keyword));
            keyword = keyword.next;
        }
    }
    String[] retVal = new String[vector.size()];
    vector.copyInto(retVal);
    return retVal;
}

public boolean getIgnoreCase()
{
    return ignoreCase;
}

public void setIgnoreCase(boolean ignoreCase)
{
    this.ignoreCase = ignoreCase;
}

private int mapLength;
private Keyword[] map;
private boolean ignoreCase;
private StringBuffer noWordSep;
private String noWordSepStr;

private int getStringMapKey(String s)
{
    return (Character.toUpperCase(s.charAt(0)) +
            Character.toUpperCase(s.charAt(s.length() - 1)))
        % mapLength;
}

protected int getSegmentMapKey(Segment s, int off, int len)
{
    return (Character.toUpperCase(s.array[off]) +
            Character.toUpperCase(s.array[off + len - 1]))
        % mapLength;
}
```csharp
public class Keyword
{
    public Keyword(char[] keyword, byte id, Keyword next)
    {
        this.keyword = keyword;
        this.id = id;
        this.next = next;
    }

    public char[] keyword;
    public byte id;
    public Keyword next;
}
```
package org.gjt.sp.jedit.syntax;

import gnu.regexp.*;
import org.gjt.sp.jedit.search.RESearchMatcher;

/**
 * A parser rule.
 * @author mike dillon, Slava Pestov
 * @version $Id: ParserRule.java,v 1.10 2003/01/31 02:19:55 spestov Exp $
 */

public class ParserRule
{
    // Major actions (total: 8)
    public static final int MAJOR_ACTIONS = 0x000000FF;
    public static final int SEQ = 0;
    public static final int SPAN = 1 << 1;
    public static final int MARK_PREVIOUS = 1 << 2;
    public static final int MARK_FOLLOWING = 1 << 3;
    public static final int EOL_SPAN = 1 << 4;
    public static final int MAJOR_ACTION_5 = 1 << 5;
    public static final int MAJOR_ACTION_6 = 1 << 6;
    public static final int MAJOR_ACTION_7 = 1 << 7;

    // Action hints (total: 8)
    public static final int ACTION_HINTS = 0x0000FF00;
    public static final int EXCLUDE_MATCH = 1 << 8;
    public static final int AT_LINE_START = 1 << 9;
    public static final int AT_WHITESPACE_END = 1 << 10;
    public static final int AT_WORD_START = 1 << 11;
    public static final int NO_LINE_BREAK = 1 << 12;
    public static final int NO_WORD_BREAK = 1 << 13;
    public static final int IS_ESCAPE = 1 << 14;
    public static final int REGEXP = 1 << 15;

    // Instance variables
    public final char hashChar;
    public final char[] start;
    // only for SEQ_REGEXP and SPAN_REGEXP rules
    public final RE startRegexp;

    public final char[] end;
public final int action;
public final byte token;

public ParserRule next;
//}}}

//{{{ getDelegateRuleSet() method
/**
 * Returns the parser rule set used to highlight text matched by this
 * rule. Only applicable for <code>SEQ</code>, <code>SPAN</code>,
 * <code>EOL_SPAN</code>, and <code>MARK_FOLLOWING</code> rules.
 *
 * @param tokenMarker The token marker
 */
public ParserRuleSet getDelegateRuleSet(TokenMarker tokenMarker) {
    // don't worry
    if (delegate == null)
    {
        if ((action & MAJOR_ACTIONS) == SEQ)
            return null;
        else
            return ParserRuleSet.getStandardRuleSet(token);
    }
    else
    {
        ParserRuleSet delegateSet = tokenMarker.getRuleSet(delegate);
        if (delegateSet == null)
        {
            return ParserRuleSet.getStandardRuleSet(Token.NULL);
        }
        else
            return delegateSet;
    }
} //}}}

//{{{ createSequenceRule() method
public static final ParserRule createSequenceRule(String seq, String delegate, byte id, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart)
{
    int ruleAction = SEQ |
    ( (atLineStart) ? AT_LINE_START : 0 ) |
    ( (atWhitespaceEnd) ? AT_WHITESPACE_END : 0 ) |
    ( (atWordStart) ? AT_WORD_START : 0 );

    return new ParserRule(ruleAction, seq.charAt(0), seq.toCharArray(), null, null, delegate, id);
} //}}}

//{{{ createRegexpSequenceRule() method
public static final ParserRule createRegexpSequenceRule(char hashChar, String seq, String delegate, byte id, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean ignoreCase) throws REException
{
    int ruleAction = SEQ | REGEXP |
    ( (atLineStart) ? AT_LINE_START : 0 ) |
    ( (atWhitespaceEnd) ? AT_WHITESPACE_END : 0 ) |
    ( (atWordStart) ? AT_WORD_START : 0 );

    return new ParserRule(ruleAction, hashChar, null, new RE("\A" + seq,(ignoreCase ? RE.REG_ICASE : 0), RESearchMatcher.RE_SYNTAX_JEDIT),
null, delegate, id);
}

//{创建Span规则方法
public static final ParserRule createSpanRule(String begin, String end, String delegate, byte id, boolean noLineBreak, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean excludeMatch, boolean noWordBreak)
{
    int ruleAction = SPAN |
    ((noLineBreak) ? NO_LINE_BREAK : 0) |
    ((atLineStart) ? AT_LINE_START : 0) |
    ((atWhitespaceEnd) ? AT_WHITESPACE_END : 0) |
    ((atWordStart) ? AT_WORD_START : 0) |
    ((excludeMatch) ? EXCLUDE_MATCH : 0) |
    ((noWordBreak) ? NO_WORD_BREAK : 0);

    return new ParserRule(ruleAction, begin.charAt(0),
        begin.toCharArray(), null, end.toCharArray(), delegate, id);
}

//{创建RegexpSpan规则方法
public static final ParserRule createRegexpSpanRule(char hashChar, String begin, String end, String delegate, byte id, boolean noLineBreak, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean excludeMatch, boolean noWordBreak, boolean ignoreCase)
throws REException
{
    int ruleAction = SPAN | REGEXP |
    ((noLineBreak) ? NO_LINE_BREAK : 0) |
    ((atLineStart) ? AT_LINE_START : 0) |
    ((atWhitespaceEnd) ? AT_WHITESPACE_END : 0) |
    ((atWordStart) ? AT_WORD_START : 0) |
    ((excludeMatch) ? EXCLUDE_MATCH : 0) |
    ((noWordBreak) ? NO_WORD_BREAK : 0);

    return new ParserRule(ruleAction, hashChar,
        null, new RE("\A" + begin,(ignoreCase ? RE.REG_ICASE : 0),
            RESearchMatcher.RE_SYNTAX_JEDIT),
        end.toCharArray(), delegate, id);
}

//{创建EOL Span规则方法
public static final ParserRule createEOLSpanRule(String seq, String delegate, byte id, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean excludeMatch)
{
    int ruleAction = EOL_SPAN |
    ((atLineStart) ? AT_LINE_START : 0) |
    ((atWhitespaceEnd) ? AT_WHITESPACE_END : 0) |
    ((atWordStart) ? AT_WORD_START : 0) |
    ((excludeMatch) ? EXCLUDE_MATCH : 0) |
    NO_LINE_BREAK;

    return new ParserRule(ruleAction, seq.charAt(0),
        seq.toCharArray(), null, null, delegate, id);
}

//{创建Regexp EOL Span规则方法
public static final ParserRule createRegexpEOLSpanRule(char hashChar, String seq, String delegate, byte id, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean excludeMatch, boolean ignoreCase)
throws REException
{
    int ruleAction = EOL_SPAN | REGEXP |
        ((atLineStart) ? AT_LINE_START : 0) |
        ((atWhitespaceEnd) ? AT_WHITESPACE_END : 0) |
        ((atWordStart) ? AT_WORD_START : 0) |
        ((excludeMatch) ? EXCLUDE_MATCH : 0) |
        NO_LINE_BREAK;

    return new ParserRule(ruleAction, hashChar, 
        null, new RE("\A" + seq,(ignoreCase ? RE.REG_ICASE : 0), 
        RESearchMatcher.RE_SYNTAX_JEDIT), null, 
        delegate, id);
} //++){

//{{( createMarkFollowingRule() method
public static final ParserRule createMarkFollowingRule(String seq, 
    byte id, boolean atLineStart, boolean atWhitespaceEnd, 
    boolean atWordStart, boolean excludeMatch)
{
    int ruleAction = MARK_FOLLOWING |
        ((atLineStart) ? AT_LINE_START : 0) |
        ((atWhitespaceEnd) ? AT_WHITESPACE_END : 0) |
        ((atWordStart) ? AT_WORD_START : 0) |
        ((excludeMatch) ? EXCLUDE_MATCH : 0);

    return new ParserRule(ruleAction, seq.charAt(0), 
        seq.toCharArray(), null, null, 
        null, id);
} //}))

//{{( createMarkPreviousRule() method
public static final ParserRule createMarkPreviousRule(String seq, 
    byte id, boolean atLineStart, boolean atWhitespaceEnd, 
    boolean atWordStart, boolean excludeMatch)
{
    int ruleAction = MARK_PREVIOUS |
        ((atLineStart) ? AT_LINE_START : 0) |
        ((atWhitespaceEnd) ? AT_WHITESPACE_END : 0) |
        ((atWordStart) ? AT_WORD_START : 0) |
        ((excludeMatch) ? EXCLUDE_MATCH : 0);

    return new ParserRule(ruleAction, seq.charAt(0), 
        seq.toCharArray(), null, null, 
        null, id);
} //}))

//{{( createEscapeRule() method
public static final ParserRule createEscapeRule(String seq)
{
    int ruleAction = IS_ESCAPE;

    return new ParserRule(ruleAction, seq.charAt(0), 
        seq.toCharArray(), null, null, 
        null, Token.NULL);
} //}))

//{{( Private members
private String delegate;

private ParserRule(int action, char hashChar, char[] start, 
    RE startRegexp, char[] end, String delegate, byte token)
{
    this.hashChar = hashChar;
    this.start = start;
    this.startRegexp = startRegexp;
    this.end = end;
this.delegate = delegate;
this.action = action;
this.token = token;
} ///)}}
}
package org.gjt.sp.jedit.syntax;

//{{{ Imports
import gnu.regexp.RE;
import java.util.*;
import org.gjt.sp.jedit.Mode;
//}}}

/**
 * A set of parser rules.
 * @author mike dillon
 * @version $Id: ParserRuleSet.java,v 1.20 2003/02/23 04:05:21 spestov Exp $
 */
public class ParserRuleSet
{

//{{{ getStandardRuleSet() method
/**
 * Returns a parser rule set that highlights everything with the
 * specified token type.
 * @param id The token type
 */
public static ParserRuleSet getStandardRuleSet(byte id)
{
    return standard[id];
}
//}}}

//{{{ ParserRuleSet constructor
public ParserRuleSet(String name, Mode mode)
{
    this.name = name;
    this.mode = mode;
    ruleMapFirst = new ParserRule[RULE_BUCKET_COUNT];
    ruleMapLast = new ParserRule[RULE_BUCKET_COUNT];
}
//}}}

//{{{ getName() method
public String getName()
{
    return name;
}
//}}}

//{{{ getMode() method
public Mode getMode()
{
```java
    return mode;
} //}}}

//{{{{ getProperties() method
public Hashtable getProperties()
{
    return props;
} //}}}

//{{{{ setProperties() method
public void setProperties(Hashtable props)
{
    this.props = props;
    noWordSep = null;
} //}}}

//{{{{ addRule() method
public void addRule(ParserRule r)
{
    ruleCount++;

    int key = Character.toUpperCase(r.hashChar) % RULE_BUCKET_COUNT;
    ParserRule last = ruleMapLast[key];
    if(last == null)
        ruleMapFirst[key] = ruleMapLast[key] = r;
    else
    {
        last.next = r;
        ruleMapLast[key] = r;
    }
} //}}}

//{{{{ getRules() method
public ParserRule getRules(char ch)
{
    int key = Character.toUpperCase(ch) % RULE_BUCKET_COUNT;
    return ruleMapFirst[key];
} //}}}

//{{{{ getRuleCount() method
public int getRuleCount()
{
    return ruleCount;
} //}}}

//{{{{ getTerminateChar() method
public int getTerminateChar()
{
    return terminateChar;
} //}}}

//{{{{ setTerminateChar() method
public void setTerminateChar(int atChar)
{
    terminateChar = (atChar >= 0) ? atChar : -1;
} //}}}

//{{{{ getIgnoreCase() method
public boolean getIgnoreCase()
{
    return ignoreCase;
} //}}}

//{{{{ setIgnoreCase() method
public void setIgnoreCase(boolean b)
{
    // code here
}
```
ignoreCase = b;
} //}}

//{{{{ getKeywords() method
public KeywordMap getKeywords()
{
    return keywords;
} //}}}

//{{{{ setKeywords() method
public void setKeywords(KeywordMap km)
{
    keywords = km;
    _noWordSep = null;
} //}}}

//{{{{ getHighlightDigits() method
public boolean getHighlightDigits()
{
    return highlightDigits;
} //}}}

//{{{{ setHighlightDigits() method
public void setHighlightDigits(boolean highlightDigits)
{
    this.highlightDigits = highlightDigits;
} //}}}

//{{{{ getDigitRegexp() method
public RE getDigitRegexp()
{
    return digitRE;
} //}}}

//{{{{ setDigitRegexp() method
public void setDigitRegexp(RE digitRE)
{
    this.digitRE = digitRE;
} //}}}

//{{{{ getEscapeRule() method
public ParserRule getEscapeRule()
{
    return escapeRule;
} //}}}

//{{{{ setEscapeRule() method
public void setEscapeRule(ParserRule escapeRule)
{
    addRule(escapeRule);
    this.escapeRule = escapeRule;
} //}}}

//{{{{ getDefault() method
public byte getDefault()
{
    return defaultToken;
} //}}}

//{{{{ setDefault() method
public void setDefault(byte def)
{
    defaultToken = def;
} //}}}

//{{{{ getNoWordSep() method
public String getNoWordSep()
if (_noWordSep == null)
{
    _noWordSep = noWordSep;
    if (noWordSep == null)
        noWordSep = "";
    if (keywords != null)
        noWordSep += keywords.getNonAlphaNumericChars();
}
return noWordSep;

//{{}}
//}}

//{{{ setNoWordSep() method
public void setNoWordSep(String noWordSep)
{
    this.noWordSep = noWordSep;
    noWordSep = null;
} //}}}

//{{{ toString() method
public String toString()
{
    return getClass().getName() + "[" + (mode == null ? "" : mode.getName()) + ":" + name + "]";
} //}}}

//{{{ Private members
private static ParserRuleSet[] standard;

static
{
    standard = new ParserRuleSet[Token.ID_COUNT];
    for (byte i = 0; i < standard.length; i++)
    {
        standard[i] = new ParserRuleSet(null, null);
        standard[i].setDefault(i);
    }
}

private static final int RULE_BUCKET_COUNT = 128;

private String name;
private Mode mode;
private Hashtable props;

private KeywordMap keywords;

private int ruleCount;

private ParserRule[] ruleMapFirst;
private ParserRule[] ruleMapLast;

private int terminateChar = -1;
private boolean ignoreCase = true;
private byte defaultToken;
private ParserRule escapeRule;

private boolean highlightDigits;
private RE digitRE;

private String _noWordSep;
private String noWordSep;
//}}}
package org.gjt.sp.jedit.syntax;

//{{{ Imports
import javax.swing.text.*;
import java.awt.font.*;
import java.util.List;
import org.gjt.sp.jedit.syntax.*;
//}}}

/**
 * Splits token lists to fit within a specified margin.
 */
public class SoftWrapTokenHandler extends DisplayTokenHandler {

//{{{ init() method

public void init(Segment seg, SyntaxStyle[] styles,
                FontRenderContext fontRenderContext,
                TabExpander expander, List out,
                float wrapMargin)
{
    super.init(seg, styles, fontRenderContext, expander);

    // SILLY: allow for anti-aliased characters' "fuzz"
    this.wrapMargin = wrapMargin += 2.0f;

    this.out = out;
    initialSize = out.size();

    seenNonWhitespace = addedNonWhitespace = false;
    endX = endOfWhitespace = 0.0f;
    end = null;
} //}}}

//{{{ getChunks() method

/**
 * Returns the list of chunks.
 * @since jEdit 4.1pre7
 */
public List getChunkList()
{
    return out;
} //}}}

//{{{ handleToken() method

/**
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */
public void handleToken(byte id, int offset, int length, LineContext context) {
    Token token = createToken(id, offset, length, context);
    if (token != null) {
        addToken(token, context);
        if (id == Token.WHITESPACE || id == Token.TAB) {
            if (!seenNonWhitespace) {
                endOfWhitespace = x;
            }
        } else {
            seenNonWhitespace = true;
            if (out.size() == initialSize) out.add(firstToken);
            else if (id == Token.WHITESPACE || id == Token.TAB) {
                if (out.size() != initialSize) {
                    end = lastToken;
                    endX = x;
                }
            } else if (x > wrapMargin && end != null && addedNonWhitespace) {
                Chunk blankSpace = new Chunk(endOfWhitespace,
                                               end.offset + end.length,
                                               getParserRuleSet(context));
                blankSpace.next = end.next;
                end.next = null;
                x = x - endX + endOfWhitespace;
                out.add(blankSpace);
                end = null;
                endX = x;
            }
            addedNonWhitespace = seenNonWhitespace;
        }
    }
} //}}}

//{{{{ Private members
private List out;
private float wrapMargin;
private float endX;
private Token end;
private boolean seenNonWhitespace;

private Token createToken(byte id, int offset, int length, LineContext context) {
    // Constructor implementation
}
private boolean addedNonWhitespace;
private float endOfWhitespace;
private int initialSize;

//}}}}
package org.gjt.sp.jedit.syntax;

import java.awt.Color;
import java.awt.Font;

/**
 * A simple text style class. It can specify the color, italic flag,
 * and bold flag of a run of text.
 * @author Slava Pestov
 * @version $Id: SyntaxStyle.java,v 1.4 2002/06/04 08:48:13 spestov Exp$
 */
public class SyntaxStyle
{

/**
 * Creates a new SyntaxStyle.
 * @param fgColor The text color
 * @param bgColor The background color
 * @param font The text font
 */
public SyntaxStyle(Color fgColor, Color bgColor, Font font)
{
    this.fgColor = fgColor;
    this.bgColor = bgColor;
    this.font = font;
}

/**
 * Returns the text color.
 */
public Color getForegroundColor()
{
    return fgColor;
}

/**
 * Returns the background color.
 */
public Color getBackgroundColor()
{
    return bgColor;
}

/**
 * Returns the style font.
 */
public Font getFont()
{ 
    return font;
}

// private members
private Color fgColor;
private Color bgColor;
private Font font;
package org.gjt.sp.jedit.syntax;

/**
 * A linked list of syntax tokens.
 * @author Slava Pestov
 * @version $Id: Token.java,v 1.9 2002/05/26 07:38:43 spestov Exp $
 */
public class Token {

    //{{{ stringToToken() method
    /**
     * Converts a token type string to a token type constant.
     * @param value The token type
     * @since jEdit 4.1pre1
     */
    public static byte stringToToken(String value) {
        value = value.intern();
        if (value == "NULL")
            return Token.NULL;
        else if (value == "COMMENT1")
            return Token.COMMENT1;
        else if (value == "COMMENT2")
            return Token.COMMENT2;
        else if (value == "LITERAL1")
            return Token.LITERAL1;
        else if (value == "LITERAL2")
            return Token.LITERAL2;
        else if (value == "LABEL")
            return Token.LABEL;
        else if (value == "KEYWORD1")
            return Token.KEYWORD1;
        else if (value == "KEYWORD2")
            return Token.KEYWORD2;
        else if (value == "KEYWORD3")
            return Token.KEYWORD3;
        else if (value == "FUNCTION")
            return Token.FUNCTION;
        else if (value == "MARKUP")
            return Token.MARKUP;
        else if (value == "OPERATOR")
            return Token.OPERATOR;
        else if (value == "DIGIT")
            return Token.DIGIT;
    }
}
else if (value == "INVALID")
    return Token.INVALID;
else
    return -1;
}

//}}}

//{{{( Token types
public static final byte NULL = 0;
public static final byte COMMENT1 = 1;
public static final byte COMMENT2 = 2;
public static final byte LITERAL1 = 3;
public static final byte LITERAL2 = 4;
public static final byte LABEL = 5;
public static final byte KEYWORD1 = 6;
public static final byte KEYWORD2 = 7;
public static final byte KEYWORD3 = 8;
public static final byte FUNCTION = 9;
public static final byte MARKUP = 10;
public static final byte OPERATOR = 11;
public static final byte DIGIT = 12;
public static final byte INVALID = 13; }}}

public static final byte ID_COUNT = 14;

//}} Special:
public static final byte WHITESPACE = 125;
public static final byte TAB = 126;
public static final byte END = 127;

//{{{( Instance variables
/**
 * The id of this token.
 */
public byte id;

/**
 * The start offset of this token.
 */
public int offset;

/**
 * The length of this token.
 */
public int length;

/**
 * The rule set of this token.
 */
public ParserRuleSet rules;

/**
 * The next token in the linked list.
 */
public Token next;
//}}}

//{{{( Token constructor
/**
 * Creates a new token.
 * @param id The id of the token
 * @param offset The start offset of the token
 * @param length The length of the token
 * @param rules The parser rule set that generated this token
 */
public Token(byte id, int offset, int length, ParserRuleSet rules) {
    this.id = id;
```java
    this.offset = offset;
    this.length = length;
    this.rules = rules;
} //}))

//{{{( toString() method
/**
 * Returns a string representation of this token.
 */
public String toString()
{
    return "[id=" + id + ",offset=" + offset + ",length=" + length + "]";
} //))))
```
package org.gjt.sp.jedit.syntax;

/**
 * Token markers send tokens to implementations of this interface.
 * @author Slava Pestov
 * @version $Id: TokenHandler.java,v 1.4 2003/02/07 21:57:43 spestov Exp $
 * @since jEdit 4.1pre1
 */

public interface TokenHandler
{
    /**
     * Called by the token marker when a syntax token has been parsed.
     * @param id The token type (one of the constants in the
     *            Token class).
     * @param offset The start offset of the token
     * @param length The number of characters in the token
     * @param context The line context
     * @since jEdit 4.1pre1
     */
    public void handleToken(byte id, int offset, int length,
                            TokenMarker.LineContext context);
}
/* TokenMarker.java - Tokenizes lines of text
 * :tabSize=8:indentSize=8:noTabs=false:
 * :folding=explicit:collapseFolds=1:
 * Copyright (C) 1998, 2002 Slava Pestov
 * Copyright (C) 1999, 2000 mike dillon
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or any later version.
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 */

package org.gjt.sp.jedit.syntax;

//{{{ Imports
import gnu.regexp.*;
import javax.swing.text.Segment;
import java.util.*;
import org.gjt.sp.jedit.search.CharIndexedSegment;
import org.gjt.sp.jedit.*;
import org.gjt.sp.util.Log;
//}}}

/**
 * A token marker splits lines of text into tokens. Each token carries
 * a length field and an identification tag that can be mapped to a color
 * or font style for painting that token.
 * @author Slava Pestov, mike dillon
 * @version $Id: TokenMarker.java,v 1.47 2003/01/31 04:49:31 spestov Exp $
 * @see org.gjt.sp.jedit.syntax.Token
 * @see org.gjt.sp.jedit.syntax.TokenHandler
 */

public class TokenMarker
{
    //{{}
    public TokenMarker()
    {
        ruleSets = new Hashtable(64);
    } //}}}

    //{{
    public String getName()
    {
        return name;
    } //}}}

    //{{
    public void setName(String name)
    {
        if (name == null)
            throw new NullPointerException();

        this.name = name;
        rulePfx = name.concat("::");
    } //}}}
}
public void addRuleSet(String setName, ParserRuleSet rules)
{
    if (rules == null)
        return;

    if (setName == null)
        setName = "MAIN";

    ruleSets.put(rulePfx.concat(setName), rules);

    if (setName.equals("MAIN"))
    
        mainRuleSet = rules;
}

public ParserRuleSet getMainRuleSet()
{
    return mainRuleSet;
}

public ParserRuleSet getRuleSet(String setName)
{
    ParserRuleSet rules;

    rules = (ParserRuleSet) ruleSets.get(setName);

    if (rules == null) {
        Log.log(Log.ERROR, this, "Unresolved delegate target: " + setName);
        return ParserRuleSet.getStandardRuleSet(Token.INVALID);
    }

    else
    
        // store external ParserRuleSet in the local hashtable
        // for faster lookups later
        ruleSets.put(setName, rules);

    if (rules == null)
        
        Log.log(Log.ERROR, this, "Unresolved delegate target: " + setName);

        return ParserRuleSet.getStandardRuleSet(Token.INVALID);
    }

else
public LineContext markTokens(LineContext prevContext, TokenHandler tokenHandler, Segment line) {
    //{{{ Set up some instance variables
    // this is to avoid having to pass around lots and lots of
    // parameters.
    this.tokenHandler = tokenHandler;
    this.line = line;

    lastOffset = line.offset;
    lineLength = line.count + line.offset;

    context = new LineContext();

    if(prevContext == null)
        context.rules = getMainRuleSet();
    else
        {
            context.parent = prevContext.parent;
            context.inRule = prevContext.inRule;
            context.rules = prevContext.rules;
        }

    keywords = context.rules.getKeywords();
    escaped = false;

    seenWhitespaceEnd = false;
    whitespaceEnd = line.offset;
    //}}}

    //{{{ Main parser loop
    ParserRule rule;
    int terminateChar = context.rules.getTerminateChar();
    boolean terminated = false;

    main_loop: for(pos = line.offset; pos < lineLength; pos++) {
        //{{{ check if we have to stop parsing
        if(terminateChar >= 0 && pos - line.offset >= terminateChar
        && !terminated)
            {
                terminated = true;
                context = new LineContext(ParserRuleSet.
                    getStandardRuleSet(context.rules
                    .getDefault()), context);
                keywords = context.rules.getKeywords();
            }
        //}}}

        //{{{ check for end of delegate
        if(context.parent != null)
            {
                rule = context.parent.inRule;
                if(rule != null)
                    {
                        if(checkDelegateEnd(rule))
                            {
                            seenWhitespaceEnd = true;
                            continue main_loop;
                        }
                    }
            }
        //}}

    return rules;
} //}})}}
```java
} })

// check every rule
char ch = line.array[pos];
rule = context.rules.getRules(ch);
while(rule != null)
{
  // stop checking rules if there was a match
  if (handleRule(rule, false))
  {
    seenWhitespaceEnd = true;
    continue main_loop;
  }

  rule = rule.next;
} //}}})

// check if current character is a word separator
if (Character.isWhitespace(ch))
{
  if (!seenWhitespaceEnd)
    whitespaceEnd = pos + 1;

  if (context.inRule != null)
    handleRule(context.inRule, true);

  handleNoWordBreak();

  markKeyword(false);

  if (lastOffset != pos)
  {
    tokenHandler.handleToken(
      context.rules.getDefault(),
      lastOffset - line.offset,
      pos - lastOffset,
      context);
  }

  tokenHandler.handleToken(
    (ch == '\t' ? Token.TAB :
     Token.WHITESPACE), pos - line.offset, 1,
    context);

  escaped = false;
}
else
{
  if (keywords != null || context.rules.getRuleCount() != 0)
  {
    String noWordSep = context.rules.getNoWordSep();

    if (!Character.isLetterOrDigit(ch)
      && noWordSep.indexOf(ch) == -1)
    {
      if (context.inRule != null)
        handleRule(context.inRule, true);

      handleNoWordBreak();

      markKeyword(true);

      tokenHandler.handleToken(
        context.rules.getDefault(),
        lastOffset - line.offset, 1,
```
context;
  lastOffset = pos + 1;
})
)

  seenWhitespaceEnd = true;
  escaped = false;
}) //})

  // Mark all remaining characters
  pos = lineLength;
  if (context.inRule != null)
    handleRule(context.inRule, true);  
    handleNoWordBreak();
  markKeyword(true);

  if (context.parent != null) {
    rule = context.parent.inRule;
    if (rule != null && (context.parent.inRule.action
      & ParserRule.NO_LINE_BREAK) == ParserRule.NO_LINE_BREAK)
      || terminated)
      context = context.parent;
      keywords = context.rules.getKeywords();
      context.inRule = null;
  } //})

  tokenHandler.handleToken(Token.END, pos - line.offset, 0, context);

  return context.intern();  
}) //})

  // Instance variables
  private Hashtable ruleSets;
  private String name;
  private String rulePfx;
  private ParserRuleSet mainRuleSet;

  // Instead of passing these around to each method, we just store them
  // as instance variables. Note that this is not thread-safe.
  private TokenHandler tokenHandler;
  private Segment line;
  private LineContext context;
  private KeywordMap keywords;
  private Segment pattern = new Segment();
  private int lastOffset;
  private int lineLength;
  private int pos;
  private boolean escaped;

  private int whitespaceEnd;
  private boolean seenWhitespaceEnd;
})//})

  // checkDelegateEnd() method
  private boolean checkDelegateEnd(ParserRule rule) {
    if (rule.end == null)
      return false;
LineContext tempContext = context;
context = context.parent;
keywords = context.rules.getKeywords();
boolean tempEscaped = escaped;
boolean b = handleRule(rule, true);
context = tempContext;
keywords = context.rules.getKeywords();

if(b && !tempEscaped)
{
    if(context.inRule != null)
        handleRule(context.inRule, true);

    markKeyword(true);
    context = (LineContext) context.parent.clone();

tokenHandler.handleToken(
    (context.inRule.action & ParserRule.EXCLUDE_MATCH)
    == ParserRule.EXCLUDE_MATCH
    ? context.rules.getDefault()
    : context.inRule.token,
    pos - line.offset, pattern.count, context);

    keywords = context.rules.getKeywords();
    context.inRule = null;
    lastOffset = pos + pattern.count;

    // move pos to last character of match sequence
    pos += (pattern.count - 1);

    return true;
}

    // check escape rule of parent
    rule = context.parent.rules.getEscapeRule();
    if(rule != null && handleRule(rule, false))
        return true;

    return false;
}  }}

//{{ handleRule() method
/**
 * Checks if the rule matches the line at the current position
 * and handles the rule if it does match
 */
private boolean handleRule(ParserRule checkRule, boolean end)
{
      //{{ Some rules can only match in certain locations
      if(!end)
      {
          if(Character.toUpperCase(checkRule.hashChar)
              != Character.toUpperCase(line.array[pos]))
              {
              return false;
          }

          if((checkRule.action & ParserRule.AT_LINE_START)
              == ParserRule.AT_LINE_START)
          {
              if(((checkRule.action & ParserRule.MARK_PREVIOUS) != 0) ?
                  lastOffset : pos) != line.offset)
                  return false;
          }
          else if((checkRule.action & ParserRule.AT_WHITESPACE_END)
              == ParserRule.AT_WHITESPACE_END)
if(((checkRule.action & ParserRule.MARK_PREVIOUS) != 0) ? lastOffset : pos) != whitespaceEnd)
    return false;
else if((checkRule.action & ParserRule.AT_WORD_START) == ParserRule.AT_WORD_START)
{
    if(((checkRule.action & ParserRule.MARK_PREVIOUS) != 0) ? lastOffset : pos) != lastOffset)
        return false;
}
int matchedChars = 1;

// See if the rule's start or end sequence matches here
if(!end || (checkRule.action & ParserRule.MARK_FOLLOWING) == 0)
{
    // the end cannot be a regular expression
    if((checkRule.action & ParserRule.REXP) == 0 || end)
    {
        pattern.array = (end ? checkRule.end : checkRule.start);
        pattern.offset = 0;
        pattern.count = pattern.array.length;
        matchedChars = pattern.count;

        if(!TextUtilities.regionMatches(context.rules.getIgnoreCase(), line, pos, pattern.array))
        {
            return false;
        }
    }
} else
{
    // note that all regexps start with \A so they only
    // match the start of the string
    int matchStart = pos - line.offset;
    REMatch match = checkRule.startRegexp.getMatch(
        new CharIndexedSegment(line, matchStart),
        0, RE.REG_ANCHORINDEX);
    if(match == null)
        return false;
    else if(match.getStartIndex() != 0)
        throw new InternalError("Can't happen");
    else
    matchedChars = match.getEndIndex();
}
}

// Check for an escape sequence
if((checkRule.action & ParserRule.IS_ESCAPE) == ParserRule.IS_ESCAPE)
{
    if(context.inRule != null)
        handleRule(context.inRule, true);
    escaped = !escaped;
    pos += pattern.count - 1;
} else if(escaped)
{
    escaped = false;
    pos += pattern.count - 1;
} else if(!end)
{
if (context.inRule != null)
    handleRule(context.inRule, true);

markKeyword((checkRule.action & ParserRule.MARK_PREVIOUS) != ParserRule.MARK_PREVIOUS);

switch (checkRule.action & ParserRule.MAJOR_ACTIONS) {
    //{{{{ SEQ
    case ParserRule.SEQ:
        if ((checkRule.action & ParserRule.REGEXP) != 0) {
            handleTokenWithTabs(tokenHandler, checkRule.token, pos - line.offset, matchedChars, context);
        } else {
            tokenHandler.handleToken(checkRule.token, pos - line.offset, matchedChars, context);
        }
        // a DELEGATE attribute on a SEQ changes the ruleset from the end of the SEQ onwards
        ParserRuleSet delegateSet = checkRule.getDelegateRuleSet(this);
        if (delegateSet != null) {
            context = new LineContext(delegateSet, context.parent);
            keywords = context.rules.getKeywords();
        }
        break;
    //}}}
    //{{{{ SPAN, EOL_SPAN
    case ParserRule.SPAN:
    case ParserRule.EOL_SPAN:
        context.inRule = checkRule;
        delegateSet = checkRule.getDelegateRuleSet(this);

        byte tokenType = ((checkRule.action & ParserRule.EXCLUDE_MATCH) == ParserRule.EXCLUDE_MATCH ? context.rules.getDefault() : checkRule.token);

        if ((checkRule.action & ParserRule.REGEXP) != 0) {
            handleTokenWithTabs(tokenHandler, tokenType, pos - line.offset, matchedChars, context);
        } else {
            tokenHandler.handleToken(tokenType, pos - line.offset, matchedChars, context);
        }
        // XXX
        /* String spanEndSubst = null;
        if ((checkRule.action & ParserRule.REGEXP) == ParserRule.REGEXP) spanEndSubst = checkRule.startRegexp... */
        context = new LineContext(delegateSet, context);
        keywords = context.rules.getKeywords();
}
break;

MarkFollowing

case ParserRule.MARK_FOLLOWING:
    tokenHandler.handleToken((checkRule.action
 & ParserRule.EXCLUDE_MATCH)
 == ParserRule.EXCLUDE_MATCH ?
    context.rules.getDefault()
 : checkRule.token, pos - line.offset,
    pattern.count, context);

    context.inRule = checkRule;
    break;

MarkPrevious

case ParserRule.MARK_PREVIOUS:
    if ((checkRule.action & ParserRule.EXCLUDE_MATCH)
 == ParserRule.EXCLUDE_MATCH)
    {
        if (pos != lastOffset)
        {
            tokenHandler.handleToken(
                checkRule.token,
                lastOffset - line.offset,
                pos - lastOffset,
                context);
        }

        tokenHandler.handleToken(
            context.rules.getDefault(),
            pos - line.offset, pattern.count,
            context);
    }
    else
    {
        tokenHandler.handleToken(checkRule.token,
            lastOffset - line.offset,
            pos - lastOffset + pattern.count,
            context);
    }

    break;

default:
    throw new InternalError("Unhandled major action");

// move pos to last character of match sequence
pos += (matchedChars - 1);
lastOffset = pos + 1;

// break out of inner for loop to check next char
}

Handle end of MARK_FOLLOWING

else if((context.inRule.action & ParserRule.MARK_FOLLOWING) != 0)
{
    if (pos != lastOffset)
    {
        tokenHandler.handleToken(
            context.inRule.token,
            lastOffset - line.offset,
            pos - lastOffset, context);
    }

    lastOffset = pos;
    context.inRule = null;
}
private void handleNoWordBreak()
{
    if (context.parent != null)
    {
        ParserRule rule = context.parent.inRule;
        if (rule != null && (context.parent.inRule.action & ParserRule.NO_WORD_BREAK) != 0)
        {
            if (pos != lastOffset)
            {
                tokenHandler.handleToken(rule.token, lastOffset - line.offset, pos - lastOffset, context);
            }

            lastOffset = pos;
            context = context.parent;
            keywords = context.rules.getKeywords();
            context.inRule = null;
        }
    }
}

private void handleTokenWithTabs(TokenHandler tokenHandler, byte tokenType, int start, int len, LineContext context)
{
    int last = start;
    int end = start + len;

    for (int i = start; i < end; i++)
    {
        if (line.array[i] == 't')
        {
            if (last != i)
            {
                tokenHandler.handleToken(tokenType, last, i - last, context);
                tokenHandler.handleToken(tokenType, i, 1, context);
                last = i + 1;
            }
        }
    }

    if (last != end)
    {
        tokenHandler.handleToken(tokenType, last, end - last, context);
    }
}

private void markKeyword(boolean addRemaining)
{
    int len = pos - lastOffset;
    if (len == 0)
    {
        return;
    }

    if (context.rules.getHighlightDigits())
    {
        boolean digit = false;
        boolean mixed = false;

        for (int i = lastOffset; i < pos; i++)
        {
            char ch = line.array[i];
            if (Character.isDigit(ch))
digit = true;
else
mixed = true;
}

if(mixed)
{
    RE digitRE = context.rules.getDigitRegexp();
    // only match against regexp if its not all digits; if all digits, no point matching
    if(digit)
    {
        if(digitRE == null)
        {
            // mixed digit/alpha keyword, and no regexp... don't highlight as DIGIT
            digit = false;
        }
        else
        {
            CharIndexedSegment seg = new CharIndexedSegment(
                line, false);
            int oldCount = line.count;
            int oldOffset = line.offset;
            line.offset = lastOffset;
            line.count = len;
            if(!digitRE.isMatch(seg))
            {
                digit = false;
                line.offset = oldOffset;
                line.count = oldCount;
            }
        }
    }
}

if(digit)
{
    tokenHandler.handleToken(Token.DIGIT,
        lastOffset - line.offset,
        len, context);
    lastOffset = pos;
    return;
}

if(keywords != null)
{
    byte id = keywords.lookup(line, lastOffset, len);
    if(id != Token.NULL)
    {
        tokenHandler.handleToken(id,
            lastOffset - line.offset,
            len, context);
        lastOffset = pos;
        return;
    }
}

if(addRemaining)
{
    tokenHandler.handleToken(context.rules.getDefault(),
        lastOffset - line.offset, len, context);
}
public static class LineContext
{
    private static Hashtable intern = new Hashtable();

    public LineContext parent;
    public ParserRule inRule;
    public ParserRuleSet rules;
    // used for SPAN_REGEXP rules; otherwise null
    public String spanEndSubst;

    public LineContext(ParserRule r, ParserRuleSet rs, String spanEndSubst)
    {
        inRule = r;
        rules = rs;
        this.spanEndSubst = spanEndSubst;
    }

    public LineContext(ParserRuleSet rs, LineContext lc)
    {
        rules = rs;
        parent = (lc == null ? null : (LineContext)lc.clone());
    }

    public LineContext()
    {
    }

    public LineContext intern()
    {
        Object obj = intern.get(this);
        if (obj == null)
        {
            intern.put(this, this);
            return this;
        }
        else
        {
            return (LineContext)obj;
        }
    }

    public int hashCode()
    {
        if (inRule != null)
            return inRule.hashCode();
        else if (rules != null)
            return rules.hashCode();
        else
            return 0;
    }

    public boolean equals(Object obj)
if (obj instanceof LineContext)
{
    LineContext lc = (LineContext)obj;
    if (lc.parent == null)
    {
        if (parent != null)
            return false;
    }
    else //if(lc.parent != null)
    {
        if (parent == null)
            return false;
        else if (!lc.parent.equals(parent))
            return false;
    }

    if (lc.spanEndSubst == null)
    {
        if (spanEndSubst != null)
            return false;
    }
    else
    {
        if (spanEndSubst == null)
            return false;
        else if (!lc.spanEndSubst.equals(spanEndSubst))
            return false;
    }

    return lc.inRule == inRule & & lc.rules == rules;
}
else
    return false;
} //}}}

//{{{{ clone() method
public Object clone()
{
    LineContext lc = new LineContext();
    lc.inRule = inRule;
    lc.rules = rules;
    lc.parent = (parent == null) ? null : (LineContext) parent.clone();
    lc.spanEndSubst = spanEndSubst;

    return lc;
} //}}}
} //}}}}
package org.gjt.sp.jedit.syntax;

//{{{ Imports
import com.microstar.xml.*;
import gnu.regexp.*;
import java.io.*;
import java.util.*;
import org.gjt.sp.jedit.search.RESearchMatcher;
import org.gjt.sp.jedit.*;
import org.gjt.sp.util.Log;
//}}}

/**
 * XML handler for mode definition files.
 */
public class XModeHandler extends HandlerBase
{
    //{{{ XModeHandler constructor
    public XModeHandler (XmlParser parser, String modeName, String path)
    {
        this.modeName = modeName;
        this.parser = parser;
        this.path = path;
        stateStack = new Stack();

        // default value
        lastNoWordSep = "_";
    } //}}}

    //{{{{ resolveEntity() method
    public Object resolveEntity(String publicId, String systemId)
    {
        if("xmode.dtd".equals(systemId))
        {
            // this will result in a slight speed up, since we
            // don't need to read the DTD anyway, as AElfred is
            // non-validating
            return new StringReader("<!-- -->");
        }
        /* try
        {
            return new BufferedReader(new InputStreamReader(
                getClass().getResourceAsStream( 
                    "/org/gjt/sp/jedit/syntax/xmode.dtd")));
        } catch (IOException e) {
            Log.printStackTrace(e);
        } */
        return null;
    } //}}}
}
public void attribute(String aname, String value, boolean isSpecified) {
    aname = (aname == null) ? null : aname.intern();

    if (aname == "NAME")
        propName = value;
    else if (aname == "VALUE")
        propValue = value;
    else if (aname == "TYPE")
        lastTokenID = Token.stringToToken(value);
        if (lastTokenID == -1)
            error("token-invalid",value);
    else if (aname == "AT_LINE_START")
        lastAtLineStart = (isSpecified) ? (value.equals("TRUE")) : false;
    else if (aname == "AT_WHITESPACE_END")
        lastAtWhitespaceEnd = (isSpecified) ? (value.equals("TRUE")) : false;
    else if (aname == "AT_WORD_START")
        lastAtWordStart = (isSpecified) ? (value.equals("TRUE")) : false;
    else if (aname == "NO_LINE_BREAK")
        lastNoLineBreak = (isSpecified) ? (value.equals("TRUE")) : false;
    else if (aname == "NO_WORD_BREAK")
        lastNoWordBreak = (isSpecified) ? (value.equals("TRUE")) : false;
    else if (aname == "EXCLUDE_MATCH")
        lastExcludeMatch = (isSpecified) ? (value.equals("TRUE")) : false;
    else if (aname == "IGNORE_CASE")
        lastIgnoreCase = (isSpecified) ? (value.equals("TRUE")) : true;
    else if (aname == "HIGHLIGHT_DIGITS")
        lastHighlightDigits = (isSpecified) ? (value.equals("TRUE")) :
false;
else if (aname == "DIGIT_RE")
{
    lastDigitRE = value;
}
else if (aname == "NO_WORD_SEP")
{
    if(isSpecified)
        lastNoWordSep = value;
}
else if (aname == "AT_CHAR")
{
    try
    {
        if (isSpecified) termChar =
            Integer.parseInt(value);
    }
    catch (NumberFormatException e)
    {
        error("termchar-invalid",value);
        termChar = -1;
    }
}
else if (aname == "ESCAPE")
{
    lastEscape = value;
}
else if (aname == "SET")
{
    lastSetName = value;
}
else if (aname == "DELEGATE")
{
    lastDelegateSet = value;
    if (lastDelegateSet != null
        && lastDelegateSet.indexOf("::") == -1)
    {
        lastDelegateSet = modeName + "::" + lastDelegateSet;
    }
}
else if (aname == "DEFAULT")
{
    lastDefaultID = Token.stringToToken(value);
    if(lastDefaultID == -1)
    {
        error("token-invalid",value);
        lastDefaultID = Token.NULL;
    }
}
else if (aname == "HASH_CHAR")
{
    if(value.length() != 1)
    {
        error("hash-char-invalid",value);
        lastDefaultID = Token.NULL;
    }
    else
    {
        lastHashChar = value.charAt(0);
    }
}
}}}

//{{doctypeDecl() method
public void doctypeDecl(String name, String publicId,
        String systemId) throws Exception
{
    if ("MODE".equalsIgnoreCase(name)) return;
error("doctype-invalid", name);
}

>{{{{ charData() method
public void charData(char[] c, int off, int len) {
    String tag = peekElement();
    String text = new String(c, off, len);
    if (tag == "EOL_SPAN" ||
        tag == "EOL_SPAN_REGEXP" ||
        tag == "MARK_PREVIOUS" ||
        tag == "MARK_FOLLOWING" ||
        tag == "SEQ" ||
        tag == "SEQ_REGEXP" ||
        tag == "BEGIN")
    {
        lastStart = text;
    }
    else if (tag == "END")
    {
        lastEnd = text;
    }
    else
    {
        lastKeyword = text;
    }
}

>{{{{ startElement() method
public void startElement (String tag) {
    tag = pushElement(tag);
    if (tag == "WHITESPACE")
    {
        Log.log(Log.WARNING, this, path + ": WHITESPACE rule " + "no longer needed");
    }
    else if (tag == "MODE")
    {
        mode = jEdit.getMode(modeName);
        if (mode == null)
        {
            mode = new Mode(modeName);
            jEdit.addMode(mode);
        }
    }
    else if (tag == "KEYWORDS")
    {
        keywords = new KeywordMap(rules.getIgnoreCase());
    }
    else if (tag == "RULES")
    {
        rules = new ParserRuleSet(lastSetName, mode);
        rules.setIgnoreCase(lastIgnoreCase);
        rules.setHighlightDigits(lastHighlightDigits);
        if (lastDigitRE != null)
        {
            try
            {
                rules.setDigitRegexp(new RE(lastDigitRE, lastIgnoreCase ? RE.REG_ICASE : 0, RESearchMatcher.RE_SYNTAX_JEDIT));
            }
            catch (Exception e)
            {
                Log.log(Log.ERROR, this, path + ": Error setting digit regular expression: ", e.getMessage());
            }
        }
    }
}
try {
    error("regexp",e);
}

if (lastEscape != null)
    rules.setEscapeRule(ParserRule.createEscapeRule(lastEscape));
rules.setDefault(lastDefaultID);
rules.setNoWordSep(lastNoWordSep);
}
} //}}}

public void endElement(String name) {
    if (name == null) return;
    String tag = popElement();
    if (name.equalsIgnoreCase(tag)) {
        //{{{ MODE
        if (tag == "MODE") {
            // no need for this anymore
            //mode.init();
        } //}}}
        //{{{ PROPERTY
        else if (tag == "PROPERTY") {
            props.put(propName, propValue);
        } //}}}
        //{{{ PROPS
        else if (tag == "PROPS") {
            if (peekElement().equals("RULES"))
                rules.setProperties(props);
            else
                mode.setProperties(props);

            props = new Hashtable();
        } //}}}
        //{{{ RULES
        else if (tag == "RULES") {
            rules.setKeywords(keywords);
            marker.addRuleSet(lastSetName, rules);
            keywords = null;
            lastSetName = null;
            lastEscape = null;
            lastIgnoreCase = true;
            lastHighlightDigits = false;
            lastDigitRE = null;
            lastDefaultID = Token.NULL;
            lastNoWordSep = ";";
            rules = null;
        } //}}}
        //{{{ TERMINATE
        else if (tag == "TERMINATE") {
            rules.setTerminateChar(termChar);
            termChar = -1;
        } //}}}
        //{{{ SEQ
else if (tag == "SEQ")
{
    if(lastStart == null)
    {
        error("empty-tag","SEQ");
        return;
    }

    rules.addRule(ParserRule.createSequenceRule(
        lastStart, lastDelegateSet, lastTokenID,
        lastAtLineStart, lastAtWhitespaceEnd,
        lastAtWordStart));

    lastStart = null;
    lastEnd = null;
    lastDelegateSet = null;
    lastTokenID = Token.NULL;
    lastAtLineStart = false;
    lastAtWordStart = false;
    lastAtWhitespaceEnd = false;
}

else if (tag == "SEQ_REGEXP")
{
    if(lastStart == null)
    {
        error("empty-tag","SEQ_REGEXP");
        return;
    }

    try
    {
        rules.addRule(ParserRule.createRegexpSequenceRule(
            lastHashChar, 
            lastStart, lastDelegateSet, lastTokenID, 
            lastAtLineStart, lastAtWhitespaceEnd, 
            lastAtWordStart, lastIgnoreCase));
    }

    catch(REException re)
    {
        error("regexp",re);
    }

    lastHashChar = '\0';
    lastStart = null;
    lastEnd = null;
    lastDelegateSet = null;
    lastTokenID = Token.NULL;
    lastAtLineStart = false;
    lastAtWordStart = false;
    lastAtWhitespaceEnd = false;
}

else if (tag == "SPAN")
{
    if(lastStart == null)
    {
        error("empty-tag","START");
        return;
    }

    if(lastEnd == null)
    {
        error("empty-tag","END");
        return;
    }

    rules.addRule(ParserRule
.createSpanRule(
lastStart, lastEnd,
lastDelegateSet,
lastTokenID, lastNoLineBreak,
lastAtLineStart,
lastAtWhitespaceEnd,
lastAtWordStart,
lastExcludeMatch,
lastNoWordBreak));

lastStart = null;
lastEnd = null;
lastTokenID = Token.NULL;
lastAtLineStart = false;
lastAtWordStart = false;
lastNoLineBreak = false;
lastAtWhitespaceEnd = false;
lastExcludeMatch = false;
lastNoWordBreak = false;
lastDelegateSet = null;
} ///})
//{{ SPAN_REGEXP
else if (tag == "SPAN_REGEXP")
{
if (lastStart == null)
{
error("empty-tag","START");
return;
}

if (lastEnd == null)
{
error("empty-tag","END");
return;
}

try
{
rules.addRule(ParserRule
 .createRegexpSpanRule(
lastHashChar,
lastStart, lastEnd,
lastDelegateSet,
lastTokenID, lastNoLineBreak,
lastAtLineStart,
lastAtWhitespaceEnd,
lastAtWordStart,
lastExcludeMatch,
lastNoWordBreak,
lastIgnoreCase));
}
catch(REException re)
{
error("regexp",re);
}

lastHashChar = '\0';
lastStart = null;
lastEnd = null;
lastTokenID = Token.NULL;
lastAtLineStart = false;
lastAtWordStart = false;
lastNoLineBreak = false;
lastAtWhitespaceEnd = false;
lastExcludeMatch = false;
lastNoWordBreak = false;
lastDelegateSet = null;
```java
else if (tag == "EOL_SPAN")
{
    if (lastStart == null)
    {
        error("empty-tag", "EOL_SPAN");
        return;
    }

    rules.addRule(ParserRule.createEOLSpanRule(
        lastStart, lastDelegateSet, lastTokenID, 
        lastAtLineStart, lastAtWhitespaceEnd, 
        lastAtWordStart, lastExcludeMatch));

    lastStart = null;
    lastEnd = null;
    lastDelegateSet = null;
    lastTokenID = Token.NULL;
    lastAtLineStart = false;
    lastAtWordStart = false;
    lastAtWhitespaceEnd = false;
    lastExcludeMatch = false;
}
```
lastTokenID, lastAtLineStart,  
lastAtWhitespaceEnd, lastAtWordStart,  
lastExcludeMatch));
lastStart = null;
lastEnd = null;
lastTokenID = Token.NULL;
lastAtLineStart = false;
lastAtWordStart = false;
lastAtWhitespaceEnd = false;
lastExcludeMatch = false;
} //}))
///{{( MARK_PREVIOUS
else if (tag == "MARK_PREVIOUS")
{
  if(lastStart == null)
  {
    error("empty-tag","MARK_PREVIOUS");
    return;
  }

  rules.addRule(ParserRule  
    .createMarkPreviousRule(lastStart,  
      lastTokenID, lastAtLineStart,  
      lastAtWhitespaceEnd, lastAtWordStart,  
      lastExcludeMatch));
lastStart = null;
lastEnd = null;
lastTokenID = Token.NULL;
lastAtLineStart = false;
lastAtWordStart = false;
lastAtWhitespaceEnd = false;
lastExcludeMatch = false;
} //}))
///{{( Keywords
else
{
  byte token = Token.stringToToken(tag);
  if(token != -1)
    addKeyword(lastKeyword,token);
} //))}
else
{
  // can't happen
  throw new InternalError();
}
} //))

///{{( startDocument() method
public void startDocument()
{
  marker = new TokenMarker();
  marker.setName(modeName);
  props = new Hashtable();

  pushElement(null);
} //))
///{{( Private members
///{{( Instance variables
private XmlParser parser;
private String modeName;
private String path;

private TokenMarker marker;
private KeywordMap keywords;
private Mode mode;
private Stack stateStack;
private String propName;
private String propValue;
private Hashtable props;
private String lastStart;
private String lastEnd;
private String lastKeyword;
private String lastSetName;
private String lastEscape;
private String lastDelegateSet;
private String lastNoWordSep;
private ParserRuleSet rules;
private byte lastDefaultID = Token.NULL;
private byte lastTokenID;
private int termChar = -1;
private boolean lastNoLineBreak;
private boolean lastNoWordBreak;
private boolean lastAtLineStart;
private boolean lastAtWhiteSpaceEnd;
private boolean lastAtWordStart;
private boolean lastExcludeMatch;
private boolean lastIgnoreCase = true;
private boolean lastHighlightDigits;
private String lastDigitRE;
private char lastHashChar;

//}}}

//{{{ addKeyword() method
private void addKeyword(String k, byte id)
{
  if (k == null)
  {
    error("empty-keyword");
    return;
  }
  if (keywords == null) return;
  keywords.add(k, id);
}
//}}}

//{{{ pushElement() method
private String pushElement(String name)
{
  name = (name == null) ? null : name.intern();
  stateStack.push(name);
  return name;
}
//}}}

//{{{ peekElement() method
private String peekElement()
{
  return (String) stateStack.peek();
}
//}}}

//{{{ popElement() method
private String popElement()
{
  return (String) stateStack.pop();
}
//}}}

//{{{ error() method
private void error(String msg)
{
  _error(jEdit.getProperty("xmode-error." + msg));
}
private void error(String msg, String subst)
{
    _error(jEdit.getProperty("xmode-error." + msg, new String[] { subst }));
}

private void error(String msg, Throwable t)
{
    _error(jEdit.getProperty("xmode-error." + msg, new String[] { t.toString() }));
    Log.log(Log.ERROR, this, t);
}

private void _error(String msg)
{
    Object[] args = { path, new Integer(parser.getLineNumber()),
                      new Integer(parser.getColumnNumber()), msg };

    GUIUtilities.error(null, "xmode-error", args);
}

//}))
//}}}}
//}}}}}
Appendix B: Checklist for class diagram inspection

Packages
Are package names exactly the same as the original?
Are there any missing packages?
Are there any unnecessary packages?

Classes & Interfaces
Are class & interface names exactly the same as the original?
Are there any classes that should be interfaces?
Are there any interfaces that should be classes?
Are there any missing classes or interfaces?
Are there any unnecessary classes or interfaces?
Are classes & interfaces placed in the right package?
Are classes & interfaces of the right visibility?
Are class & interface modifiers accurate?

Dependencies & associations
Are there any missing dependencies?
Are there any unnecessary dependencies?
Are associations of the right type?
Are associations correctly navigated?
Are association descriptions correct according to association type?
Are association descriptions correct according to original code?

Methods
Are method names exactly the same as the original?
Are methods of the right type?
Are methods of the right visibility?
Are method modifiers accurate?
Are parameter names exactly the same as the original?
Are parameters of the right type?
Are there any missing parameters?
Are there any unnecessary parameters?
Are there any missing methods?
Are there any unnecessary methods?

Variables
Are variable names exactly the same as the original?
Are variables of the right type?
Are variables of the right visibility?
Are variable modifiers accurate?
Are there any missing variables?
Are there any unnecessary variables?
Are initial values correct?
Are there any missing initial values?
Are there any unnecessary initial values?
Appendix C: Checklist for sequence diagram inspection

Classes & objects
Are class & object names exactly the same as the original?
Are classes displayed as classes i.e. not underlined?
Are objects displayed as objects i.e. underlined?
Are object types visible?

Messages (method calls & return statements)
Are method names exactly the same as the original?
Are parameter names exactly the same as the original?
Are there any missing parameters?
Are there any unnecessary parameters?
Are all method calls modelled?
Are all method calls aimed to the right object or class?
Are all return statements modelled?
Are variable names exactly the same as the original?
Are variables of the right type?
Is conditional information correct?
Are method calls that create new objects connected directly to the object?
Are method calls to existing objects connected to the lifeline activity column?
Appendix D: Checklist for code generation inspection

**Class diagram**
Has each class been assigned a java file?
Have inner classes been added to the same file as their owner?
Has the package definition been added to the file?
Have the import statements been added to the file?
Has the class definition, including visibility and modifiers, been added?
Has class inheritance been added?
Have all attributes been added?
Have all operations been added?
Is the operation signature of each operation complete?

**Sequence diagrams**
Have all messages been added to the right operation body?
Are all messages in the same order as in the models?
Are operation calls aimed at the correct objects or classes?
Are return statements written as such?
Appendix E: Checklist for code comparison

**Packages**
Is the package name correct?
Is the package definition missing?

**Import statements**
Are import statements spelled correctly?
Are the right classes or packages imported?
Are there any missing import statements?
Are there any unnecessary import statements?

**Class definitions**
Is the class definition present?
Is the class name exactly the same as the original?
Is there any class that should be an interface?
Is there any interface that should be a class?
Is the class of the right visibility?
Are class modifiers accurate?
Is class inheritance information correct?
Are inner classes present?

**Instance variables**
Are variable names exactly the same as the original?
Are variables of the right type?
Are variables of the right visibility?
Are variable modifiers accurate?
Are there any missing variables?
Are there any unnecessary variables?
Are initial values correct?
Are there any missing initial values?
Are there any unnecessary initial values?

**Method definitions**
Are method names exactly the same as the original?
Are methods of the right type?
Are methods of the right visibility?
Are method modifiers accurate?
Are parameter names exactly the same as the original?
Are parameters of the right type?
Are there any missing parameters?
Are there any unnecessary parameters?
Are there any missing methods?
Are there any unnecessary methods?

**Method bodies**
Is all original content present?
Is there any extra content?
Is present content logically the same as the original?
Is present content functionally the same as the original?

**General**
Is all content in the same order as in the original code?
Is there any content outside of operations that is not present?
Appendix G: syntax package class details

**Chunk**

```
+ DEBUG : static boolean = false
+ accessible : boolean
+ visible : boolean
+ monospaced : boolean
+ charWidth : float
+ style : SyntaxStyle
+ width : float
+ gv : GlyphVector
- positions : float[]
```

```
+ offsetToX(chunks : Chunk, offset : int) : static float
+ xToOffset(chunks : Chunk, x : float, round : boolean) : static int
+ Chunk(width : float, offset : int, rules : ParserRuleSet)
+ getPositions() : final float[]
+ offsetToX(offset : int) : final float
+ xToOffset(x : float, round : boolean) : final int
+ init(seg : Segment, expander : TabExpander, x : float, styles : SyntaxStyle[], fontRenderContext : FontRenderContext, defaultID : byte, charWidth : float) : void
```

**DefaultTokenHandler**

```
DefaultTokenHandler
(from org.gjt.sp.jedit.syntax)
```

```
# firstToken : Token
# lastToken : Token

+ init() : void
+ getTokens() : Token
+ handleToken(id : byte, offset : int, length : int, context : TokenMarker.LineContext) : void
# getParserRuleSet(context : TokenMarker.LineContext) : ParserRuleSet
# createToken(id : byte, offset : int, length : int, context : TokenMarker.LineContext) : Token
# addToken(token : Token, context : TokenMarker.LineContext) : void
```

**DisplayTokenHandler**

```
DisplayTokenHandler
(from org.gjt.sp.jedit.syntax)
```

```
# seg : Segment
# styles : SyntaxStyle[]
# fontRenderContext : FontRenderContext
# expander : TabExpander
# x : float
# charWidth : float

+ init(seg : Segment, styles : SyntaxStyle[], fontRenderContext : FontRenderContext, expander : TabExpander) : void
+ setMonospacedCharWidth(charWidth : float) : void
+ getChunks() : Chunk
# createToken(id : byte, offset : int, length : int, context : TokenMarker.LineContext) : Token
```
### DummyTokenHandler

```java
+ INSTANCE : static final DummyTokenHandler = new DummyTokenHandler()
+ handleToken(id : byte, offset : int, length : int, context : TokenMarker.LineContext) : void
```

### KeywordMap

```java
- mapLength : int
- map : Keyword[]
- ignoreCase : boolean
- noWordSep : StringBuffer
- noWordSepStr : String
+ KeywordMap(ignoreCase : boolean)
+ KeywordMap(ignoreCase : boolean, mapLength : int)
+ lookup(text : Segment, offset : int, length : int) : byte
+ add(keyword : String, id : byte) : void
+ getNonAlphaNumericChars() : String
+ getKeywords() : String[]
+ getIgnoreCase() : boolean
+ setIgnoreCase(ignoreCase : boolean) : void
- getStringMapKey(s : String) : int
# getSegmentMapKey(s : Segment, offset : int, length : int) : int
```

### Keyword

(inner class of KeywordMap)

```java
+ keyword : char[]
+ id : byte
+ next : Keyword
+ Keyword(keyword : char[], id : byte, next : Keyword)
```
ParserRule

Operation signature shown in separate note

ParserRule
(from org.gjt.sp.jedit.syntax)
+ getDelegateRuleSet(tokenMarker : TokenMarker) : ParserRuleSet
+ createSequenceRule(seq : String, delegate : String, id : byte, atLineStart : boolean, atWhitespaceEnd : boolean, atWordStart : boolean) : static final ParserRule
+ createRegexpSequenceRule(hashChar : char, seq : String, delegate : String, id : byte, atLineStart : boolean, atWhitespaceEnd : boolean, atWordStart : boolean, ignoreCase : boolean) : static final ParserRule {throws REException}
+ createMarkFollowingRule(seq : String, id : byte, atLineStart : boolean, atWhitespaceEnd : boolean, atWordStart : boolean, excludeMatch : boolean) : static final ParserRule
+ createMarkPreviousRule(seq : String, id : byte, atLineStart : boolean, atWhitespaceEnd : boolean, atWordStart : boolean, excludeMatch : boolean) : static final ParserRule
+ createEscapeRule(seq : String) : static final ParserRule
- ParserRule()
### ParserRuleSet

#### Static Members
- `standard : static ParserRuleSet[]`
- `RULE_BUCKET_COUNT : static final int = 128`

#### Instance Members
- `name : String`
- `mode : Mode`
- `props : Hashtable`
- `keywords : KeywordMap`
- `ruleCount : int`
- `ruleMapFirst : ParserRule[]`
- `ruleMapLast : ParserRule[]`
- `terminateChar : int = -1`
- `ignoreCase : boolean = true`
- `defaultToken : Byte`
- `escapeRule : ParserRule`
- `highlightDigits : boolean`
- `digitRE : RE`
- `_noWordSep : String`
- `noWordSep : String`

#### Public Methods

- `+ getStandardRuleSet(id : byte) : static ParserRuleSet`
- `+ ParserRuleSet(name : String, mode : Mode)`
- `+ getName() : String`
- `+ getMode() : Mode`
- `+ getProperties() : Hashtable`
- `+ setProperties(props : Hashtable) : void`
- `+ addRule(r : ParserRule) : void`
- `+ getRules(ch : char) : ParserRule`
- `+ getRuleCount() : int`
- `+ getTerminateChar() : int`
- `+ setTerminateChar(atChar : int) : void`
- `+ getIgnoreCase() : boolean`
- `+ setIgnoreCase(b : boolean) : void`
- `+ getKeywords() : KeywordMap`
- `+ setKeywords(km : KeywordMap) : void`
- `+ getHighlightDigits() : boolean`
- `+ setHighlightDigits(highlightDigits : boolean) : void`
- `+ getDigitRegexp() : RE`
- `+ setDigitRegexp(digitRE : RE) : void`
- `+ getEscapeRule() : ParserRule`
- `+ setEscapeRule(escapeRule : ParserRule) : void`
- `+ getDefault() : byte`
- `+ setDefault(def : byte) : void`
- `+ getNoWordSep() : String`
- `+ setNoWordSep(noWordSep : String) : void`
- `+ toString() : String`
### SoftWrapTokenHandler

- out : List
- wrapMargin : float
- endX : float
- end : Token
- seenNonWhitespace : boolean
- addedNonWhitespace : boolean
- endOfWhitespace : float
- initialSize : int

```java
+ init(seg : Segment, styles : SyntaxStyle[], fontRenderContext : FontRenderContext, expander : TabExpander, out : List, wrapMargin : float) : void
+ getChunkList() : List
+ handleToken(id : byte, offset : int, length : int, context : TokenMarker.LineContext) : void
```

### SyntaxStyle

- fgColor : Color
- bgColor : Color
- font : Font

```java
+ SyntaxStyle(fgColor : Color, bgColor : Color, font : Font)
+ getForegroundColor() : Color
+ getBackgroundColor() : Color
+ getFont() : Font
```
### Token

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>Static final byte = 0</td>
</tr>
<tr>
<td>COMMENT1</td>
<td>Static final byte = 1</td>
</tr>
<tr>
<td>COMMENT2</td>
<td>Static final byte = 2</td>
</tr>
<tr>
<td>LITERAL1</td>
<td>Static final byte = 3</td>
</tr>
<tr>
<td>LITERAL2</td>
<td>Static final byte = 4</td>
</tr>
<tr>
<td>LABEL</td>
<td>Static final byte = 5</td>
</tr>
<tr>
<td>KEYWORD1</td>
<td>Static final byte = 6</td>
</tr>
<tr>
<td>KEYWORD2</td>
<td>Static final byte = 7</td>
</tr>
<tr>
<td>KEYWORD3</td>
<td>Static final byte = 8</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>Static final byte = 9</td>
</tr>
<tr>
<td>MARKUP</td>
<td>Static final byte = 10</td>
</tr>
<tr>
<td>OPERATOR</td>
<td>Static final byte = 11</td>
</tr>
<tr>
<td>DIGIT</td>
<td>Static final byte = 12</td>
</tr>
<tr>
<td>INVALID</td>
<td>Static final byte = 13</td>
</tr>
<tr>
<td>ID_COUNT</td>
<td>Static final byte = 14</td>
</tr>
<tr>
<td>WHITESPACE</td>
<td>Static final byte = 125</td>
</tr>
<tr>
<td>TAB</td>
<td>Static final byte = 126</td>
</tr>
<tr>
<td>END</td>
<td>Static final byte = 127</td>
</tr>
<tr>
<td>id</td>
<td>Byte</td>
</tr>
<tr>
<td>offset</td>
<td>Int</td>
</tr>
<tr>
<td>length</td>
<td>Int</td>
</tr>
<tr>
<td>rules</td>
<td>ParserRuleSet</td>
</tr>
<tr>
<td>next</td>
<td>Token</td>
</tr>
</tbody>
</table>

+ stringToToken(value : String) : static byte
+ Token(id : byte, offset : int, length : int, rules : ParserRuleSet) : static byte
+ toString() : String
TokenHandler

- handleToken(id: byte, offset: int, length: int, context: TokenMarker.LineContext): void

TokenMarker

- ruleSets: Hashtable
- name: String
- rulePfx: String
- mainRuleSet: ParserRuleSet
- tokenHandler: TokenHandler
- line: Segment
- context: LineContext
- keywords: KeywordMap
- pattern: Segment = new Segment()
- lastOffset: int
- lineLength: int
- pos: int
- escaped: boolean
- whitespaceEnd: int
- seenWhitespaceEnd: boolean

+ TokenMarker()
+ getName(): String
+ setName(name: String): void
+ addRuleSet(setName: String, rules: ParserRuleSet): void
+ getMainRuleSet(): ParserRuleSet
+ getRuleContext(setName: String): ParserRuleSet
- checkDelegateEnd(rule: ParserRule): boolean
- handleRule(checkRule: ParserRule, end: boolean): boolean
- handleNoWordBreak(): void
- handleTokenWithTabs(tokenHandler: TokenHandler, tokenType: byte, start: int, len: int, context: LineContext): void
- markKeyword(addRemaining: boolean): void

(from org.gjt.sp.jedit.syntax)
**LineContext**
(inner class of TokenMarker)

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>- intern : static Hashtable = new Hashtable()</td>
</tr>
<tr>
<td>+ parent : LineContext</td>
</tr>
<tr>
<td>+ inRule : ParserRule</td>
</tr>
<tr>
<td>+ rules : ParserRuleSet</td>
</tr>
<tr>
<td>+ spanEndSubst : String</td>
</tr>
<tr>
<td>+ LineContext(r : ParserRule, rs : ParserRuleSet, spanEndSubst : String)</td>
</tr>
<tr>
<td>+ LineContext(rs : ParserRuleSet, lc : LineContext)</td>
</tr>
<tr>
<td>+ LineContext()</td>
</tr>
<tr>
<td>+ intern() : LineContext</td>
</tr>
<tr>
<td>+ hashCode() : int</td>
</tr>
<tr>
<td>+ equals(obj : Object) : boolean</td>
</tr>
<tr>
<td>+ clone() : Object</td>
</tr>
</tbody>
</table>
XModeHandler

<table>
<thead>
<tr>
<th>XModeHandler(parser : XmlParser, modeName : String, path : String)</th>
</tr>
</thead>
<tbody>
<tr>
<td>resolveEntity(publicId : String, systemId : String) : Object</td>
</tr>
<tr>
<td>attribute(aname : String, value : String, isSpecified : boolean) : void</td>
</tr>
<tr>
<td>doctypeDecl(name : String, publicId : String, systemId : String) : void {throws Exception}</td>
</tr>
<tr>
<td>charData(c : char[], off : int, len : int) : void</td>
</tr>
<tr>
<td>startElement(tag : String) : void</td>
</tr>
<tr>
<td>endElement(name : String) : void</td>
</tr>
<tr>
<td>startDocument() : void</td>
</tr>
<tr>
<td>addKeyWord(k : String, id : byte) : void</td>
</tr>
<tr>
<td>peekElement(name : String) : String</td>
</tr>
<tr>
<td>peekElement() : String</td>
</tr>
<tr>
<td>popElement() : String</td>
</tr>
<tr>
<td>error(msg : String) : void</td>
</tr>
<tr>
<td>error(msg : String, subst : String) : void</td>
</tr>
<tr>
<td>error(msg : String, t : Throwable) : void</td>
</tr>
<tr>
<td>_error(msg : String) : void</td>
</tr>
</tbody>
</table>
Appendix H: syntax package sequence diagrams
init(Segment, TabExpander, float, SyntaxStyle[], FontRenderContext, byte, float)

[length == 1 && seg.array[seg.offset + offset] = 't'] newX := nextTabStop(x,offset + length) : float

str := String(seg.array,seg.offset + offset,length) : String
<<create>>

str : String

style : SyntaxStyle

gv := getFont().createGlyphVector(fontRenderContext,str)

gv : GlyphVector

width := (float)gv.getLogicalBounds().getWidth()
DefaultTokenHandler

getTokens()

firstToken

handleToken(byte, int, int, LineContext)

token := createToken(id, offset, length, context) : Token

(token != null) addToken(token, context)

getParserRuleSet(LineContext)

[context.rules.getMode() != null] context.rules

while(context != null)

null

createToken(byte, int, int, LineContext)

new Token(id, offset, length, getParserRuleSet(context))
KeywordMap page 2

KeywordMap

getIgnoreCase()

ignoreCase

getStringMapKey(String)

(Character.toUpperCase(s.charAt(0)) + Character.toUpperCase(s.charAt(s.length()-1))) % mapLength

getSegmentMapKey(Segment, int, int)

(Character.toUpperCase(s.array[off]) + Character.toUpperCase(s.array[off + len - 1])) % mapLength
SoftWrapTokenHandler

: SoftWrapTokenHandler
  init(Segment, SyntaxStyle[], FontRenderContext, TabExpander, List, float)
  getChunkList()
  handleToken(byte, int, int, LineContext)

: DisplayTokenHandler
  out : List
  init(seg, styles, fontRenderContext, expander)
  initialSize := size()

: DefaultTokenHandler
  token := createToken(id, offset, length, context) : Token
  [token != null] addToken(token, context)
  [token != null] addToken(token, context)
  [[token != null] && (x > wrapMargin && end != null && addedNonWhitespace)] add(blankSpace)

BlankSpace : Chunk

[75x255]: SoftWrapTokenHandler
[95x485]: DisplayTokenHandler
[131x438]initialSize := size()
[149x241]getChunkList()
[167x262]out
[203x314]token := createToken(id, offset, length, context) : Token
[113x334]init(seg,styles,fontRenderContext,expander)
[239x446][token != null] addToken(token, context)
[285x759]blankSpace := Chunk(endOfWhitespace, end.offset + end.length, getParserRuleSet(context)) : Chunk
[221x445][token != null] addToken(token, context)
[281x307][token != null] addToken(token, context)
[22x17]SoftWrapTokenHandler
[271x307][token != null] addToken(token, context)
[281x307][token != null] addToken(token, context)
Token

stringToToken(String)

[value = "NULL"] Token.NULL
[value = "COMMENT1"] Token.COMMENT1
[value = "COMMENT2"] Token.COMMENT2
[value = "LITERAL1"] Token.LITERAL1
[value = "LITERAL2"] Token.LITERAL2
[value = "LABEL"] Token.LABEL
[value = "KEYWORD1"] Token.KEYWORD1
[value = "KEYWORD2"] Token.KEYWORD2
[value = "KEYWORD3"] Token.KEYWORD3
[value = "FUNCTION"] Token.FUNCTION
[value = "MARKUP"] Token.MARKUP
[value = "OPERATOR"] Token.OPERATOR
[value = "DIGIT"] Token.DIGIT
[value = "INVALID"] Token.INVALID
-1

toString()

return "[id=": id + ",offset=" + offset + ",length=" + length + "]"
markTokens() continues from previous page

context : LineContext

checkDelegateEnd(ParserRule)

[b && !tempEscaped] false

[b && !tempEscaped] true

[b && !tempEscaped] handleRule(context.inRule,true)

[b && !tempEscaped] markKeyword(true)

[b && !tempEscaped] context := (LineContext)context.parent.clone()

[b && !tempEscaped] handleToken((context.inRule.action & ParserRule.EXCLUDE_MATCH)==ParserRule.EXCLUDE_MATCH? context.rules.getDefault(): context.inRule.token,pos - line.offset,pattern.count,context)

[b && !tempEscaped] keywords := rules.getKeywords()

[b && !tempEscaped] true

rule := parent.rules.getEscapeRule()

(!end) && (Character.toUpperCase(checkRule.hashChar)!=Character.toUpperCase(line.array[pos])) false


markTokens() continues from previous page

handleToken(Token.END,pos - line.offset,0,context)

keywords := rules.getKeywords()

b := handleRule(rule,true) : boolean

keywords := rules.getKeywords()

[b && !tempEscaped] handleRule(context.inRule,true)

[b && !tempEscaped] markKeyword(true)

[b && !tempEscaped] context := (LineContext)context.parent.clone()

[b && !tempEscaped] handleToken((context.inRule.action & ParserRule.EXCLUDE_MATCH)==ParserRule.EXCLUDE_MATCH? context.rules.getDefault(): context.inRule.token,pos - line.offset,pattern.count,context)

[b && !tempEscaped] keywords := rules.getKeywords()

[b && !tempEscaped] true

rule := parent.rules.getEscapeRule()

(!end) && (Character.toUpperCase(checkRule.hashChar)!=Character.toUpperCase(line.array[pos])) false

resolveEntity(String, String)

"xmode.dtd".equals(systemId) new StringReader("<!-- -->")

attribute(String, String, boolean)

aname : String

aname != null intern()

aname = "TYPE" lastTokenID := stringToToken(value)

aname = "AT_LINE_START" && lastAtLineStart = isSpecified equals("TRUE")

aname = "AT_WHITESPACE_END" && lastAtWhitespaceEnd = isSpecified equals("TRUE")

aname = "AT_WORD_START" && lastAtWordStart = isSpecified equals("TRUE")

aname = "NO_LINE_BREAK" && lastNoLineBreak = isSpecified equals("TRUE")

aname = "NO_WORD_BREAK" && lastNoWordBreak = isSpecified equals("TRUE")

aname = "EXCLUDE_MATCH" && lastExcludeMatch = isSpecified equals("TRUE")

aname = "IGNORE_CASE" && lastIgnoreCase = isSpecified equals("TRUE")

aname = "HIGHLIGHT_DIGITS" && lastHighlightDigits = isSpecified equals("TRUE")

termChar := parseInt(value)

aname = "AT_CHAR" && isSpecified error("termChar-invalid", value)

aname = "DEFAULT" lastDefaultID := stringToToken(value)

aname = "DEFAULT" && lastDefaultID = -1 error("token-invalid", value)

aname = "HASH_CHAR" && value.length() != -1 error("hash-char-invalid", value)

aname = "HASH_CHAR" && value.length() = 1 lastHashChar := charAt(0);
XModeHandler

endElement() continues from previous page

\[
\text{[name.equalsIgnoreCase(tag) && tag = "SPAN_REGEXP" && lastStart = null]} \\text{error("empty-tag","START")}
\]

\[
\text{[name.equalsIgnoreCase(tag) && tag = "SPAN_REGEXP" && lastEnd = null]} \\text{error("empty-tag","END")}
\]

addRule(ParserRule.createRegexpSpanRule(lastHashChar,lastStart,lastEnd,lastDelegateSet,lastTokenID,lastNoLineBreak,lastAtLineStart,lastAtWhitespaceEnd,lastAtWordStart,lastExcludeMatch,lastIgnoreCase))

\[
\text{[name.equalsIgnoreCase(tag) && tag = "EOL_SPAN" && lastStart = null]} \\text{error("empty-tag","EOL_SPAN")}
\]

addRule(ParserRule.createEOLSpanRule(lastStart,lastDelegateSet,lastTokenID,lastAtLineStart,lastAtWhitespaceEnd,lastExcludeMatch))

\[
\text{[name.equalsIgnoreCase(tag) && tag = "EOL_SPAN_REGEXP" && lastStart = null]} \\text{error("empty-tag","EOL_SPAN_REGEXP")}
\]

addRule(ParserRule.createRegexpEOLSpanRule(lastHashChar,lastStart,lastDelegateSet,lastTokenID,lastAtLineStart,lastAtWhitespaceEnd,lastExcludeMatch))

\[
\text{[name.equalsIgnoreCase(tag) && tag = "MARK_FOLLOWING" && lastStart = null]} \\text{error("empty-tag","MARK_FOLLOWING")}
\]

addRule(ParserRule.createMarkFollowingRule(lastStart,lastTokenID,lastAtLineStart,lastAtWhitespaceEnd,lastExcludeMatch))

\[
\text{[name.equalsIgnoreCase(tag) && tag = "MARK_PREVIOUS" && lastStart = null]} \\text{error("empty-tag","MARK_PREVIOUS")}
\]

addRule(ParserRule.createMarkPreviousRule(lastStart,lastTokenID,lastAtLineStart,lastAtWhitespaceEnd,lastExcludeMatch))

[name.equalsIgnoreCase(tag)]
\[
\text{[name.equalsIgnoreCase(tag)] token := stringToToken(tag) : byte}
\]

[name.equalsIgnoreCase(tag) && token != -1]
\[
\text{addKeyword(lastKeyword,token)}
\]

[name.equalsIgnoreCase(tag)]
\[
\text{new InternalError()}
\]
startDocument()

marker := TokenMarker()

setName(modeName)

<<create>>

props := Hashtable()

pushElement(null)

addKeyword(String, byte)

[k = null] error("empty-keyword")

keywords := KeywordMap

add(k, id)

pushElement(String)

[name != null] name := intern()

stateStack := Stack

push(name)

(name) stateStack.peek()

popElement()

(String) stateStack.pop()

error(String)

(String) stateStack.pop()

error(String)

(String, String)

error(String, Throwable)

_error(jEdit.getProperty("xmode-error." + msg))

_error(jEdit.getProperty("xmode-error." + msg,new String[] { subst }));

_error(jEdit.getProperty("xmode-error." + msg,new String[] { t.toString() }));

log(Log.ERROR,this,t)

GUIUtilities

error(null,"xmode-error",args)
Appendix I: Generated code

// Generated code

class Chunk extends Token {
    public static boolean DEBUG = false;
    public boolean accessible;
    public boolean visible;
    public boolean monospaced;
    public float charWidth;
    public SyntaxStyle style;
    public float width;
    public GlyphVector gv;
    private float[] positions;

    public static float paintChunkList(Segment lineText, Chunk chunks, Graphics2D gfx, float x, float y, Color background, boolean glyphVector) {
        FontMetrics forBackground = gfx.getFontMetrics();
        for (;;) {
            if (chunks == null) {
                return _x;
            }

            if (DEBUG) {
                gfx.draw(new Rectangle2D.Float(x + _x, y - 10, width, 10));
            }

            if (start.accessable) {
                Color bgColor = start.style.getBackgroundColor();
            }

            if ((start.accessable) && (bgColor != null)) {
                Graphics2D xorGfx = (Graphics2D) gfx.create();
                xorGfx.setXORMode(background);
                xorGfx.setColor(bgColor);
                xorGfx.fill(new Rectangle2D.Float(x + _x, y -
                                                  forBackground.getAscent(), _x + width - _x,
                                                  forBackground.getHeight()));
                xorGfx.dispose();
            }

            if ((start.accessable) && (start.visible)) {
                gfx.setFont(start.style.getFont());
                gfx.setColor(start.style.getForegroundColor());
            }

            if (((start.accessable) && (start.visible)) &&
                (glyphVector && start.gv != null &&
start.next == chunks))
{
    gfx.drawGlyphVector(start.gv,x + _x,y);
}

if(((start.accessable) && (start.visible)) && (!glyphVector && start.gv != null && start.next == chunks))
{
    gfx.drawText(lineText.array,lineText.offset + start.offset,length, (int)(x + _x),(int)y);
}
}

public static float offsetToX(Chunk chunks, int offset)
{
    if(chunks != null && offset < chunks.offset)
    {
        throw new ArrayIndexOutOfBoundsException(offset + " < " + chunks.offset);
    }
    while(chunks != null)
    {
        if(chunks.accessable && offset < chunks.offset + chunks.length)
        {
            return x + chunks.offsetToX(offset - chunks.offset);
        }
    }
    return x;
}

public static int xToOffset(Chunk chunks, float x, boolean round)
{
    while(chunks != null)
    {
        if(chunks.accessable && x < _x + chunks.width)
        {
            return chunks.xToOffset(x - _x,round);
        }
    }
}

public Chunk(float width, int offset, ParserRuleSet rules)
{
    Token(Token.NULL,offset,0,rules);
}

public Chunk(byte id, int offset, int length, ParserRuleSet rules)
{
    Token(id,offset,length,rules);
}

public final float[] getPositions()
{
    if(gv == null)
    {
        return null;
    }

    if(positions == null)
    {
        positions = gv.getGlyphPositions(0,length,null);
    }
}
return positions;
}

public final float offsetToX(int offset)
{
    if (!visible)
    {
        return 0.0f;
    }
    if (monospaced)
    {
        return offset * charWidth;
    }
    return getPositions()[offset * 2];
}

public final int xToOffset(float x, boolean round)
{
    if (((visible) && (round && width - x < x)))
    {
        return offset + length;
    }
    if (!visible)
    {
        return offset;
    }
    if (monospaced)
    {
        x = Math.max(0, x);
    }
    if (((monospaced) && (round && remainder > charWidth / 2))
    {
        return offset + i + 1;
    }
    if (((monospaced) && (!round && remainder > charWidth / 2)))
    {
        return offset + i;
    }

    float[] pos = getPositions();
    for (int i = 0; i < length; i++)
    {
        if ((nextX > x) && (!round || nextX - x > x - glyphX))
        {
            return offset + i;
        }
        if ((nextX > x) && (!round || nextX - x > x - glyphX))
        {
            return offset + i + 1;
        }
    }
    return -1
}

public void init(Segment seg, TabExpander expander, float x,
                SyntaxStyle[] styles, FontRenderContext fontRenderContext,
                byte defaultID, float charWidth)
if(length == 1 & seg.array[seg.offset + offset] == '\t')
{
    float newX = expander.nextTabStop(x,offset + length);
}

String str = new String(seg.array,seg.offset + offset,length);
gv = style.getFont().createGlyphVector(fontRenderContext,str);
width = (float)gv.getLogicalBounds().getWidth();
//Generated code

package org.gjt.sp.jedit.syntax;

class DefaultTokenHandler implements TokenHandler {
    protected Token firstToken;
    protected Token lastToken;

    public void init()
    {
    }

    public Token getTokens()
    {
        return firstToken;
    }

    public void handleToken(byte id, int offset, int length,
                            TokenMarker.LineContext context)
    {
        Token token = createToken(id, offset, length, context);
        if (token != null)
        {
            addToken(token, context);
        }
    }

    protected ParserRuleSet getParserRuleSet(TokenMarker.LineContext context)
    {
        while (context != null)
        {
            if (context.rules.getMode() != null)
            {
                return context.rules;
            }
        }
        return null;
    }

    protected Token createToken(byte id, int offset, int length,
                                 TokenMarker.LineContext context)
    {
        return new Token(id, offset, length, getParserRuleSet(context));
    }

    protected void addToken(Token token, TokenMarker.LineContext context)
    {
    }
}
package org.gjt.sp.jedit.syntax;

import java.awt.font.*;
import javax.swing.text.*;
import org.gjt.sp.jedit.syntax.*;

class DisplayTokenHandler extends DefaultTokenHandler {
    protected Segment seg;
    protected SyntaxStyle[] styles;
    protected FontRenderContext fontRenderContext;
    protected TabExpander expander;
    protected float x;
    protected float charWidth;

    public void init(Segment seg, SyntaxStyle[] styles,
                     FontRenderContext fontRenderContext, TabExpander expander)
    {
        DefaultTokenHandler.init();
    }

    public void setMonospacedCharWidth(float charWidth)
    {
    }

    public Chunk getChunks()
    {
        return (Chunk)firstToken;
    }

    protected Token createToken(byte id, int offset, int length,
                                 TokenMarker.LineContext context)
    {
        if (id == Token.END)
        {
            return null;
        }

        Chunk chunk = new Chunk(id,offset,length,
                                 getParserRuleSet(context));

        chunk.init(seg,expander,x,styles,fontRenderContext,
                    context.rules.getDefault(), charWidth);

        return chunk;
    }
}
package org.gjt.sp.jedit.syntax;

class DummyTokenHandler implements TokenHandler {
    public static final DummyTokenHandler INSTANCE = new DummyTokenHandler();

    public void handleToken(byte id, int offset, int length, TokenMarker.LineContext context) {
    }
}

package org.gjt.sp.jedit.syntax;

import java.util.Vector
import javax.swing.text.Segment;
import org.gjt.sp.jedit.TextUtilities;

class KeywordMap {
    private int mapLength;
    private Keyword[] map;
    private boolean ignoreCase;
    private StringBuffer noWordSep;
    private String noWordSepStr;

    public KeywordMap(boolean ignoreCase) {
        this(ignoreCase, 52);
        noWordSep = new StringBuffer();
    }

    public KeywordMap(boolean ignoreCase, int mapLength) {
        map = new Keyword[mapLength];
    }

    public byte lookup(Segment text, int offset, int length) {
        if (length == 0) {
            return Token.NULL;
        }

        while (k != null) {
            if (TextUtilities.regionMatches(ignoreCase, text, offset, k.keyword)) {
                return k.id;
            }
        }

        return Token.NULL;
    }

    public void add(String keyword, byte id) {
        int key = getStringMapKey(keyword);
        char[] chars = keyword.toCharArray();

        for (int i = 0; i < chars.length; i++) {
            if (!Character.isLetterOrDigit(ch)) {
                noWordSep.append(ch);
            }
        }

        map[key] = new Keyword(chars, id, map[key]);
    }

    public String getNonAlphaNumericChars() {
        return noWordSep.toString();
    }
}
public String[] getKeywords()
{
    Vector vector = new Vector(100);

    for(int i = 0; i < map.length; i++)
    {
        while(keyword != null)
        {
            vector.addElement(new String(keyword.keyword));
        }
    }

    String[] retVal = new String[vector.size()];
    vector.copyInto(retVal);

    return retVal;
}

public boolean getIgnoreCase()
{
    return ignoreCase;
}

public void setIgnoreCase(boolean ignoreCase)
{
}

private int getStringMapKey(String s)
{
    return (Character.toUpperCase(s.charAt(0)) +
            Character.toUpperCase(s.charAt(s.length()-1))) % mapLength;
}

protected int getSegmentMapKey(Segment s, int off, int len)
{
    return (Character.toUpperCase(s.array[off]) +
            Character.toUpperCase(s.array[off + len - 1])) % mapLength
}

class Keyword
{
    public char[] keyword;
    public byte id;
    public Keyword next;

    public Keyword(char[] keyword, byte id, Keyword next)
    {
    }
}
}
package org.gjt.sp.jedit.syntax;

import gnu.regexp.*;
import org.gjt.sp.jedit.search.RESearchMatcher;

class ParserRule {
    public static final int MAJOR_ACTIONS = 0x000000FF;
    public static final int SEQ = 0;
    public static final int SPAN = 1 << 1;
    public static final int MARK_PREVIOUS = 1 << 2;
    public static final int MARK_FOLLOWING = 1 << 3;
    public static final int EOL_SPAN = 1 << 4;
    public static final int ACTION_HINTS = 0x0000FF00;
    public static final int EXCLUDE_MATCH = 1 << 8;
    public static final int AT_LINE_START = 1 << 9;
    public static final int AT_WHITESPACE_END = 1 << 10;
    public static final int AT_WORD_START = 1 << 11;
    public static final int NO_LINE_BREAK = 1 << 12;
    public static final int NO_WORD_BREAK = 1 << 13;
    public static final int IS_ESCAPE = 1 << 14;
    public static final int REGEXP = 1 << 15;
    public final char hashChar;
    public final char[] start;
    public final RE startRegexp;
    public final char[] end;
    public final int action;
    public final byte token;
    public ParserRule next;
    private String delegate;

    public ParserRuleSet getDelegateRuleSet(Tokenizer tokenMarker)
    {
        if( (delegate == null) && ((action & MAJOR_ACTIONS) == SEQ))
        {
            return null;
        }

        if( (delegate == null) && (!(action & MAJOR_ACTIONS) == SEQ))
        {
            return ParserRuleSet.getStandardRuleSet(token);
        }

        if(delegate != null)
        {
            ParserRuleSet delegateSet = tokenMarker.getRuleSet(delegate);
        }

        if( (delegate != null) && (delegateSet == null))
        {
            return ParserRuleSet.getStandardRuleSet(Token.NULL);
        }

        if( (delegate != null) && (delegateSet != null))
        {
            return delegateSet;
        }
    }

    public static final ParserRule createSequenceRule(String seq,
            String delegate, byte id, boolean atLineStart,
            boolean atWhitespaceEnd, boolean atWordStart)
    {
        return new ParserRule(ruleAction, seq.charAt(0),
                seq.toCharArray(), null, null, delegate, id);
    }
public static final ParserRule createRegexpSequenceRule(char hashChar, String seq, String delegate, byte id, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean ignoreCase) throws REException
{
    return new ParserRule(ruleAction, hashChar, null,
    new RE("\A" + seq, (ignoreCase ? RE.REG_ICASE : 0),
    RESearchMatcher.RE_SYNTAX_JEDIT),null, delegate, id);
}

public static final ParserRule createSpanRule(String begin, String end, String delegate, byte id, boolean noLineBreak, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean excludeMatch, boolean noWordBreak)
{
    return new ParserRule(ruleAction, begin.charAt(0), begin.toCharArray(), null, end.toCharArray(), delegate, id);
}

public static final ParserRule createRegexpSpanRule(char hashChar, String begin, String delegate, byte id, boolean noLineBreak, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean excludeMatch, boolean noWordBreak, boolean ignoreCase) throws REException
{
    return new ParserRule(ruleAction, hashChar, null,
    new RE("\A" + begin, (ignoreCase ? RE.REG_ICASE : 0),
    RESearchMatcher.RE_SYNTAX_JEDIT), end.toCharArray(), delegate, id);
}

public static final ParserRule createEOLSpanRule(String seq, String delegate, byte id, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean excludeMatch)
{
    return new ParserRule(ruleAction, seq.charAt(0), seq.toCharArray(), null, null, delegate, id);
}

public static final ParserRule createRegexpEOLSpanRule(char hashChar, String seq, String delegate, byte id, boolean noLineBreak, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean excludeMatch, boolean ignoreCase) throws REException
{
    return new ParserRule(ruleAction, hashChar, null,
    new RE("\A" + seq, (ignoreCase ? RE.REG_ICASE : 0),
    RESearchMatcher.RE_SYNTAX_JEDIT),null,delegate,id);
}

public static final ParserRule createMarkFollowingRule(String seq, byte id, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean excludeMatch)
{
    return new ParserRule(ruleAction, seq.charAt(0), seq.toCharArray(), null, null, null, id);
}

public static final ParserRule createMarkPreviousRule(String seq, byte id, boolean atLineStart, boolean atWhitespaceEnd, boolean atWordStart, boolean excludeMatch)
{
    return new ParserRule(ruleAction, seq.charAt(0), seq.toCharArray(), null, null, null, id);
}
public static final ParserRule createEscapeRule(String seq)
{
    return new ParserRule(ruleAction, seq.charAt(0), seq.toCharArray(), null, null, null, Token.NULL);
}

private ParserRule(int action, char hashChar, char[] start, RE startRegexp, char[] end, String delegate, byte token)
{
}
package org.gjt.sp.jedit.syntax;

import gnu.regexp.RE;
import java.util.*;
import org.gjt.sp.jedit.Mode;

class ParserRuleSet {
    private static ParserRuleSet[] standard;
    private static final int RULE_BUCKET_COUNT = 128;
    private String name;
    private Mode mode;
    private Hashtable props;
    private KeywordMap keywords;
    private int ruleCount;
    private ParserRule[] ruleMapFirst;
    private ParserRule[] ruleMapLast;
    private int terminateChar = -1;
    private boolean ignoreCase = true;
    private byte defaultToken;
    private ParserRule escapeRule;
    private boolean highlightDigits;
    private RE digitRE;
    private String _noWordSep;
    private String noWordSep;

    public static ParserRuleSet getStandardRuleSet(byte id) {
        return standard[id];
    }

    public ParserRuleSet(String name, Mode mode) {
        ruleMapFirst = new ParserRule[RULE_BUCKET_COUNT];
        ruleMapLast = new ParserRule[RULE_BUCKET_COUNT];
    }

    public String getName() {
        return name;
    }

    public Mode getMode() {
        return mode;
    }

    public Hashtable getProperties() {
        return props;
    }

    public void setProperties(Hashtable props) {
    }

    public void addRule(ParserRule r) {
        int key = Character.toUpperCase(r.hashChar) % RULE_BUCKET_COUNT;
    }

    public ParserRule getRules(char ch) {
        int key = Character.toUpperCase(ch) % RULE_BUCKET_COUNT;
    }
    return ruleMapFirst[key];
}

public int getRuleCount()
{
    return ruleCount;
}

public int getTerminateChar()
{
    return terminateChar;
}

public void setTerminateChar(int atChar)
{
}

public boolean getIgnoreCase()
{
    return ignoreCase;
}

public void setIgnoreCase(boolean b)
{
}

public KeywordMap getKeywords()
{
    return keywords;
}

public void setKeywords(KeywordMap km)
{
}

public boolean getHighlightDigits()
{
    return highlightDigits;
}

public void setHighlightDigits(boolean highlightDigits)
{
}

public RE getDigitRegexp()
{
    return digitRE;
}

public void setDigitRegexp(RE digitRE)
{
}

public ParserRule getEscapeRule()
{
    return escapeRule;
}

public void setEscapeRule(ParserRule escapeRule)
{
    addRule(escapeRule);
}

public byte getDefault()
{
    return defaultToken;
}
public void setDefault(byte def) {
}

public String getNoWordSep() {
    if (_noWordSep == null && (keywords != null)) {
        noWordSep = noWordSep + keywords.getNonAlphaNumericChars();
    }
    return noWordSep;
}

public void setNoWordSep(String noWordSep) {
}

public String toString() {
    return getClass().getName() + "[" + (mode == null ? "" : mode.getName()) + ":" + name + "]";
}

standard = new ParserRuleSet[Token.ID_COUNT];
for (byte i = 0; i < standard.length; i++) {
    standard[i] = new ParserRuleSet(null, null);
    standard[i].setDefault(i);
}
package org.gjt.sp.jedit.syntax;

import java.awt.font.*;
import java.util.List;
import javax.swing.text.*;
import org.gjt.sp.jedit.syntax.*;

class SoftWrapTokenHandler extends DisplayTokenHandler {
    private List out;
    private float wrapMargin;
    private float endX;
    private Token end;
    private boolean seenNonWhitespace;
    private boolean addedNonWhitespace;
    private float endOfWhitespace;
    private int initialSize;

    public void init(Segment seg, SyntaxStyle[] styles, FontRenderContext fontRenderContext, TabExpander expander, List out, float wrapMargin)
    {
        DisplayTokenHandler.init(seg, styles, fontRenderContext, expander);

        initialSize = out.size();
    }

    public List getChunkList()
    {
        return out;
    }

    public void handleToken(byte id, int offset, int length, TokenMarker.LineContext context)
    {
        Token token = DisplayTokenHandler.createToken(id,offset, length,context);

        if(token != null)
        {
            addToken(token,context);
        }

        if((token != null) && (out.size() == initialSize))
        {
            out.add(firstToken);
        }

        if((token != null) && (x > wrapMargin && end != null && addedNonWhitespace))
        {
            Chunk blankSpace = new Chunk(endOfWhitespace,end.offset + end.length,getParserRuleSet(context));

            out.add(blankSpace);
        }
    }
}
// Generated code

package org.gjt.sp.jedit.syntax;

import java.awt.Color;
import java.awt.Font;

class SyntaxStyle {
    private Color fgColor;
    private Color bgColor;
    private Font font;

    public SyntaxStyle(Color fgColor, Color bgColor, Font font) {
    }

    public Color getForegroundColor() {
        return fgColor;
    }

    public Color getBackgroundColor() {
        return bgColor;
    }

    public Font getFont() {
        return font;
    }
}
package org.gjt.sp.jedit.syntax;

class Token {
    public static final byte NULL = 0;
    public static final byte COMMENT1 = 1;
    public static final byte COMMENT2 = 2;
    public static final byte LITERAL1 = 3;
    public static final byte LITERAL2 = 4;
    public static final byte LABEL = 5;
    public static final byte KEYWORD1 = 6;
    public static final byte KEYWORD2 = 7;
    public static final byte KEYWORD3 = 8;
    public static final byte FUNCTION = 9;
    public static final byte MARKUP = 10;
    public static final byte OPERATOR = 11;
    public static final byte INVALID = 13;
    public static final byte DIGIT = 12;
    public static final byte ID_COUNT = 14;
    public static final byte WHITESPACE = 125;
    public static final byte TAB = 126;
    public static final byte END = 127;
    public byte id;
    public int offset;
    public int length;
    public ParserRuleSet rules;
    public Token next;

    public static byte stringToToken(String value) {
        value = value.intern();

        if (value == "NULL") {
            return Token.NULL;
        }

        if (value == "COMMENT1") {
            return Token.COMMENT1;
        }

        if (value == "COMMENT2") {
            return Token.COMMENT2;
        }

        if (value == "LITERAL1") {
            return Token.LITERAL1;
        }

        if (value == "LITERAL2") {
            return Token.LITERAL2;
        }

        if (value == "LABEL") {
            return Token.LABEL;
        }

        if (value == "KEYWORD1") {
            return Token.KEYWORD1;
        }

        return Token.INVALID;
    }
}
if (value == "KEYWORD2")
{
    return Token.KEYWORD2;
}

if (value == "KEYWORD3")
{
    return Token.KEYWORD3;
}

if (value == "FUNCTION")
{
    return Token.FUNCTION;
}

if (value == "MARKUP")
{
    return Token.MARKUP;
}

if (value == "OPERATOR")
{
    return Token.OPERATOR;
}

if (value == "DIGIT")
{
    return Token.DIGIT;
}

if (value == "INVALID")
{
    return Token.INVALID;
}

return -1;

public Token(byte id, int offset, int length, ParserRuleSet rules)
{
}

public String toString()
{
    return "[id=" + id + ",offset=" + offset + ",
    length=" + length + "]";
}
package org.gjt.sp.jedit.syntax;

interface TokenHandler
{
    public void handleToken(byte id, int offset, int length,
                           TokenMarker.LineContext context);
}
package org.gjt.sp.jedit.syntax;

import gnu.regexp./*;
import java.util./*;
import javax.swing.text.Segment;
import org.gjt.sp.jedit;/*;
import org.gjt.sp.jedit.search.CharIndexedSegment;
import org.gjt.sp.util.Log;

class TokenMarker {
  private Hashtable ruleSets;
  private String name;
  private String rulePfx;
  private ParserRuleSet mainRuleSet;
  private TokenHandler tokenHandler;
  private Segment line;
  private LineContext context;
  private KeywordMap keywords;
  private Segment pattern = new Segment();
  private int lastOffset;
  private int lineLength;
  private int pos;
  private boolean escaped;
  private int whitespaceEnd;
  private boolean seenWhitespaceEnd;

  public TokenMarker() {
    ruleSets = new Hashtable(64);
  }

  public String getName() {
    return name;
  }

  public void setName(String name) {
    if (name == null) {
      throw new NullPointerException();
    }
    rulePfx = name.concat("::");
  }

  public void addRuleSet(String setName, ParserRuleSet rules) {
    if (rules == null) {
      return;
    }
    ruleSets.put(rulePfx.concat(setName), rules);
  }

  public ParserRuleSet getMainRuleSet() {
    return mainRuleSet;
  }

  public ParserRuleSet getRuleSet(String setName) {
    ParserRuleSet rules = (ParserRuleSet) ruleSets.get(setName);
  }
if(rules == null && !setName.startsWith(rulePfx))
{
    int delim = setName.indexOf("::");
}

if((rules == null && !setName.startsWith(rulePfx)) &&
   (delim == -1))
{
    bytes id = Token.stringToToken(setName);
    rules = ParserRuleSet.getStandardRuleSet(id);
}

if((rules == null && !setName.startsWith(rulePfx)) &&
   (delim != -1))
{
    String modeName = setName.substring(0, delim);
    Mode mode = jEdit.getMode(modeName);
}

if((rules == null && !setName.startsWith(rulePfx)) &&
   (delim != -1) && (mode == null))
{
    Log.log(Log.ERROR, TokenMarker.class, "Unknown edit mode: " + modeName);
}

if((rules == null && !setName.startsWith(rulePfx)) &&
   (delim != -1) && (mode != null))
{
    TokenMarker marker = mode.getTokenMarker();
    rules = marker.getRuleSet(setName);
}

if(rules == null && !setName.startsWith(rulePfx))
{
    ruleSets.put(setName, rules);
}

if(rules == null)
{
    Log.log(Log.ERROR, this, "Unresolved delegate target: " + setName);
    return ParserRuleSet.getStandardRuleSet(Token.INVALID);
}

if(rules != null)
{
    return rules;
}

public LineContext markTokens(LineContext prevContext, TokenHandler tokenHandler, Segment line)
{
    context = new LineContext();
    if(prevContext == null)
    {
        context.rules = getMainRuleSet();
    }
    keywords = context.rules.getKeywords();
    int terminateChar = context.rules.getTerminateChar();

    for(pos = line.offset; pos < lineLength; pos++)
\[
{\begin{align*}
&\text{if} (\text{terminateChar} \geq 0 \&\& \text{pos} - \text{line.offset} \geq \text{terminateChar} \&\& \neg \text{terminated}) \\
&\quad \text{context} = \text{new LineContext(} \text{ParserRuleSet.getStandardRuleSet(context.rules.getDefault(), context)}; \\
&\quad \text{keywords} = \text{context.rules.getKeywords}(); \\
&\end{align*}}
\]

ParserRule rule = context.rules.getRules(ch);

\[
{\begin{align*}
&\text{if} ((\text{Character.isWhitespace}(ch)) \&\& (\text{context.inRule} \neq \text{null})) \\
&\quad \text{handleRule}(\text{context.inRule, true}); \\
&\end{align*}}
\]

\[
{\begin{align*}
&\text{if} (\text{Character.isWhitespace}(ch)) \\
&\quad \text{handleNoWordBreak}(); \\
&\quad \text{markKeyword(false);} \\
&\end{align*}}
\]

\[
{\begin{align*}
&\text{if} ((\text{Character.isWhitespace}(ch)) \&\& (\text{lastOffset} \neq \text{pos})) \\
&\quad \text{tokenHandler.handleToken(} \text{context.rules.getDefault()};, \\
&\quad \quad \text{lastOffset} - \text{line.offset}, \text{pos} - \text{lastOffset}, \text{context}); \\
&\end{align*}}
\]

\[
{\begin{align*}
&\text{if} (\text{Character.isWhitespace}(ch)) \\
&\quad \text{tokenHandler.handleToken((ch == '\t' ?} \\
&\quad \quad \text{Token.TAB : Token.WHITESPACE),} \\
&\quad \quad \text{pos} - \text{line.offset}, 1, \text{context}); \\
&\end{align*}}
\]

\[
{\begin{align*}
&\text{if} (((\text{Character.isWhitespace}(ch)) \&\& \text{keywords} \neq \text{null}) \&\& \text{context.rules.getRuleCount()} \neq 0)) \\
&\quad \text{String} \text{noWordSep} = \text{context.rules.getNoWordSep}(); \\
&\quad \text{if} (((\text{Character.isWhitespace}(ch)) \&\& \text{keywords} \neq \text{null}) \&\& \text{context.rules.getRuleCount()} \neq 0) \&\& \\
&\quad \quad \text{Character.isLetterOrDigit(ch) \&\&} \\
&\quad \quad \text{noWordSep.indexOf(ch) == -1}) \&\& (\text{context.inRule} \neq \text{null}) \\
&\quad \quad \text{handleRule(} \text{context.inRule, true}); \\
&\end{align*}}
\]

\[
{\begin{align*}
&\text{if} (((\text{Character.isWhitespace}(ch)) \&\& \text{keywords} \neq \text{null}) \&\& \text{context.rules.getRuleCount()} \neq 0) \&\& \\
&\quad \quad \text{Character.isLetterOrDigit(ch) \&\&} \\
&\quad \quad \text{noWordSep.indexOf(ch) == -1}) \\
&\quad \quad \text{handleNoWordBreak}(); \\
&\quad \quad \text{markKeyword(true);} \\
&\quad \quad \text{tokenHandler.handleToken(} \text{context.rules.getDefault()};, \\
&\quad \quad \quad \text{lastOffset} - \text{line.offset}, 1, \text{context}); \\
&\end{align*}}
\]

\[
{\begin{align*}
&\text{if} (\text{context.inRule} \neq \text{null}) \\
&\quad \text{handleRule(} \text{context.inRule, true}); \\
&\end{align*}}
\]
handleNoWordBreak();
markKeyword(true);

if((context.parent != null) && ((rule != null &&
  (context.parent.inRule.action & ParserRule.NO_LINE_BREAK) ==
  ParserRule.NO_LINE_BREAK) || terminated)) {
  keywords = context.rules.getKeywords();
}

tokenHandler.handleToken(Token.END, pos - line.offset, 0, context);
return context.intern();
}

private boolean checkDelegateEnd(ParserRule rule) {
  if(rule.end == null) {
    return false;
  }

  keywords = context.rules.getKeywords();
  boolean b = handleRule(rule, true);
  keywords = context.rules.getKeywords();
  if((b && !tempEscaped) && (context.inRule != null)) {
    handleRule(context.inRule, true);
  }

  if(b && !tempEscaped) {
    markKeyword(true);
    context = (LineContext)context.parent.clone();
    tokenHandler.handleToken((context.inRule.action &
      ParserRule.EXCLUDE_MATCH) == ParserRule.EXCLUDE_MATCH ?
    context.rules.getDefault() : context.inRule.token, pos - line.offset, pattern.count, context);
    keywords = context.rules.getKeywords();
    return true;
  }

  rule = context.parent.rules.getEscapeRule();
  if(rule != null && handleRule(rule, false)) {
    return true;
  }

  return false;
}

private boolean handleRule(ParserRule checkRule, boolean end) {
  if(!(end) && (Character.toUpperCase(checkRule.hashChar) != Character.toUpperCase(line.array[pos]))) {
    return false;
  }

  if(!(end) && (checkRule.action & ParserRule.AT_LINE_START)
== ParserRule.AT_LINE_START) && (((checkRule.action & ParserRule.MARK_PREVIOUS) != 0) ? lastOffset : pos) != line.offset))
{
    return false;
}

if(((end) && ((checkRule.action & ParserRule.AT_WHITESPACE_END)
    == ParserRule.AT_WHITESPACE_END)) && (((checkRule.action & ParserRule.MARK_PREVIOUS) != 0) ? lastOffset : pos) != whitespaceEnd)
{
    return false;
}

if(((end) && ((checkRule.action & ParserRule.AT_WORD_START)
    == ParserRule.AT_WORD_START) && !((checkRule.action & ParserRule.MARK_PREVIOUS) != 0) ? lastOffset : pos) != lastOffset)
{
    return false;
}

if(((end || (checkRule.action & ParserRule.MARK_FOLLOWING) == 0) && ((checkRule.action & ParserRule.REEXP) == 0 || end)) &&
    !TextUtilities.regionMatches(context.rules.getIgnoreCase(), line, pos, pattern.array))
{
    return false;
}

if(((end || (checkRule.action & ParserRule.MARK_FOLLOWING) == 0) && !((checkRule.action & ParserRule.REEXP) == 0 || end)) &&
    (match == null))
{
    return false;
}

if(((end || (checkRule.action & ParserRule.MARK_FOLLOWING) == 0) && !((checkRule.action & ParserRule.REEXP) == 0 || end)) &&
    (match.getStartIndex() != 0))
{
    throw new InternalError("Can't happen");
}

if(((end || (checkRule.action & ParserRule.MARK_FOLLOWING) == 0) && !((checkRule.action & ParserRule.REEXP) == 0 || end))
{
    matchedChars = match.getEndIndex();
}

if(((checkRule.action & ParserRule.IS_ESCAPE) == ParserRule.IS_ESCAPE) && (context.inRule != null))
{
    handleRule(context.inRule, true);
}

if(!(end) && (context.inRule != null))
{
    handleRule(context.inRule, true);
if(!end)
{
markKeyword((checkRule.action & ParserRule.MARK_PREVIOUS) != ParserRule.MARK_PREVIOUS);
}

if(!(end) && (checkRule.action & ParserRule.MAJOR_ACTIONS == ParserRule.SEQ) && ((checkRule.action & ParserRule.REGEXP) != 0))
{
  handleTokenWithTabs(tokenHandler, checkRule.token, pos - line.offset, matchedChars, context);
}

if(!(end) && (checkRule.action & ParserRule.MAJOR_ACTIONS == ParserRule.SEQ) && (checkRule.action & ParserRule.REGEXP) == 0)
{
  tokenHandler.handleToken(checkRule.token, pos - line.offset, matchedChars, context);
}

if(!(end) && (checkRule.action & ParserRule.MAJOR_ACTIONS == ParserRule.SEQ))
{
  ParserRuleSet delegateSet = checkRule.getDelegateRuleSet(this);
}

if(!(end) && (checkRule.action & ParserRule.MAJOR_ACTIONS == ParserRule.SEQ) && (delegateSet != null))
{
  context = new LineContext(delegateSet, context.parent);
  keywords = context.rules.getKeywords();
}

if(!(end) && (checkRule.action & ParserRule.MAJOR_ACTIONS == ParserRule.EOL_SPAN))
{
  delegateSet = checkRule.getDelegateRuleSet(this);
}

if(!(end) && (checkRule.action & ParserRule.MAJOR_ACTIONS == ParserRule.EOL_SPAN) && ((checkRule.action & ParserRule.REGEXP) != 0))
{
  handleTokenWithTabs(tokenHandler, tokenType, pos - line.offset, matchedChars, context);
}

if(!(end) && (checkRule.action & ParserRule.MAJOR_ACTIONS == ParserRule.EOL_SPAN) && (checkRule.action & ParserRule.REGEXP) == 0)
{
  tokenHandler.handleToken(tokenType, pos - line.offset, matchedChars, context);
}

if(!(end) && (checkRule.action & ParserRule.MAJOR_ACTIONS == ParserRule.EOL_SPAN))
{
  context = new LineContext(delegateSet, context);
  keywords = context.rules.getKeywords();
}

if(!(end) && (checkRule.action & ParserRule.MAJOR_ACTIONS == ParserRule.MARK_FOLLOWING))
private void handleNoWordBreak()
{
    if (context.parent != null)
    {
        ParserRule rule = context.parent.inRule;
    }

    if (context.inRule.action & ParserRule.NO_WORD_BREAK) != 0) && (pos != lastOffset)
    
    keywords = context.rules.getKeywords();
}
private void handleTokenWithTabs(TokenHandler tokenHandler, byte tokenType, int start, int len, LineContext context) {
    for (int i = start; i < end; i++)
    {
        if ((line.array[i] == '\t') && (last != i))
        {
            tokenHandler.handleToken(tokenType, last, i - last, context);
            tokenHandler.handleToken(tokenType, i, 1, context);
        }
    }
    if (last != end)
    {
        tokenHandler.handleToken(tokenType, last, end - last, context);
    }
}

private void markKeyword(boolean addRemaining)
{
    if (len == 0)
    {
        return;
    }
    if ((context.rules.getHighlightDigits()) && (mixed))
    {
        RE digitRE = context.rules.getDigitRegexp();
    }
    if ((context.rules.getHighlightDigits()) && (mixed) && ((digit) && (digitRE != null)))
    {
        CharIndexedSegment seg = new CharIndexedSegment(line, false);
    }
    if ((context.rules.getHighlightDigits()) && (digit))
    {
        tokenHandler.handleToken(Token.DIGIT, lastOffset - line.offset, len, context); lastOffset = pos;
        return;
    }
    if (keywords != null)
    {
        byte id = keywords.lookup(line, lastOffset, len);
    }
    if ((keywords != null) && (id != Token.NULL))
    {
        tokenHandler.handleToken(id, lastOffset - line.offset, len, context);
        return;
    }
    if (addRemaining)
    {
        tokenHandler.handleToken(context.rules.getDefault(), lastOffset - line.offset, len, context);
    }
}
class LineContext {
    private static Hashtable intern = new Hashtable();
    public LineContext parent;
    public ParserRule inRule;
    public ParserRuleSet rules;
    public String spanEndSubst;

    public LineContext(ParserRule r, ParserRuleSet rs, String spanEndSubst) {
    }

    public LineContext(ParserRuleSet rs, LineContext lc) {
        if (lc != null) {
            parent = (LineContext)lc.clone();
        }
    }

    public LineContext() {
    }

    public LineContext intern() {
        Object obj = intern.get(this);

        if (obj == null) {
            intern.put(this, this);
            return this;
        }

        if (obj != null) {
            return (LineContext)obj;
        }
    }

    public int hashCode() {
        if (inRule != null) {
            return inRule.hashCode();
        }

        if (rules != null) {
            return rules.hashCode();
        }

        return 0;
    }

    public boolean equals(Object obj) {
        if (((obj instanceof LineContext) && (lc.parent == null) && (parent != null))) {
            return false;
        }

        if (((obj instanceof LineContext) && (lc.parent != null)) && (parent == null))
```java
{
    return false;
}

if(((obj instanceof LineContext) && (lc.parent != null)) &&
    (!lc.parent.equals(parent)))
{
    return false;
}

if(((obj instanceof LineContext) && (lc.spanEndSubst == null)) &&
    (spanEndSubst != null))
{
    return false;
}

if(((obj instanceof LineContext) && (lc.spanEndSubst != null)) &&
    (spanEndSubst == null))
{
    return false;
}

if(((obj instanceof LineContext) && (lc.spanEndSubst != null)) &&
    (!lc.spanEndSubst.equals(spanEndSubst)))
{
    return false;
}

if(obj instanceof LineContext)
{
    return lc.inRule == inRule && lc.rules == rules;
}

if(!(obj instanceof LineContext))
{
    return false;
}

public Object clone()
{
    LineContext lc = new LineContext();

    if(parent != null)
    {
        lc.parent = (LineContext) parent.clone();
    }

    return lc;
}
```
package org.gjt.sp.jedit.syntax;

import com.microstar.xml.*;
import gnu.regexp.*;
import java.io.*;
import java.util.*;
import org.gjt.sp.jedit.*;
import org.gjt.sp.jedit.search.RESearchMatcher;
import org.gjt.sp.util.Log;

class XModeHandler extends HandlerBase {
    private XmlParser parser;
    private String modeName;
    private String path;
    private TokenMarker marker;
    private KeywordMap keywords;
    private Mode mode;
    private Stack stateStack;
    private String propName;
    private String propValue;
    private Hashtable props;
    private String lastStart;
    private String lastEnd;
    private String lastKeyword;
    private String lastSetName;
    private String lastEscape;
    private String lastDelegateSet;
    private String lastNoWordSep;
    private ParserRuleSet rules;
    private byte lastDefaultID = Token.NULL;
    private byte lastTokenID;
    private int termChar = -1;
    private boolean lastNoLineBreak;
    private boolean lastNoWordBreak;
    private boolean lastAtLineStart;
    private boolean lastAtWhiteSpaceEnd;
    private boolean lastAtWordStart;
    private boolean lastExcludeMatch;
    private boolean lastIgnoreCase = true;
    private boolean lastHighlightDigits;
    private String lastDigitRE;
    private char lastHashChar;

    public XModeHandler(XmlParser parser, String modeName, String path) {
        stateStack = new Stack();
    }

    public Object resolveEntity(String publicId, String systemId) {
        if("xmode.dtd".equals(systemId)) {
            return new StringReader("<!-- -->");
        }
        return null;
    }

    public void attribute(String aname, String value, boolean isSpecified) {
        if(aname != null) {
            aname = aname.intern();
        }
    }
}
if (aname == "TYPE")
{
    lastTokenID = Token.stringToToken(value);
}

if ((aname == "TYPE") && (lastTokenID == -1))
{
    error("token-invalid",value);
}

if ((aname == "AT_LINE_START") && (lastAtLineStart == (isSpecified)))
{
    value.equals("TRUE");
}

if ((aname == "AT_WHITESPACE_END") && (lastAtWhitespaceEnd == (isSpecified)))
{
    value.equals("TRUE");
}

if ((aname == "AT_WORD_START") && (lastAtWordStart == (isSpecified)))
{
    value.equals("TRUE");
}

if ((aname == "NO_LINE_BREAK") && (lastNoLineBreak == (isSpecified)))
{
    value.equals("TRUE");
}

if ((aname == "NO_WORD_BREAK") && (lastNoWordBreak == (isSpecified)))
{
    value.equals("TRUE");
}

if ((aname == "EXCLUDE_MATCH") && (lastExcludeMatch == (isSpecified)))
{
    value.equals("TRUE");
}

if ((aname == "IGNORE_CASE") && (lastIgnoreCase == (isSpecified)))
{
    value.equals("TRUE");
}

if ((aname == "HIGHLIGHT_DIGITS") && (lastHighlightDigits == (isSpecified)))
{
    value.equals("TRUE");
}

if ((aname == "AT_CHAR") && (isSpecified))
{
    termChar = Integer.parseInt(value);
}

if (aname == "AT_CHAR")
{
    catch (NumberFormatException e)
    {
    }
}

if (aname == "AT_CHAR" && (e))
{
error("termchar-invalid",value);
}
if(aname == "DEFAULT")
{
  lastDefaultID = Token.stringToToken(value);
}
if((aname == "DEFAULT" && (lastDefaultID == -1))
{
  error("token-invalid",value);
}
if((aname == "HASH_CHAR" && (value.length() != 1))
{
  error("hash-char-invalid",value);
}
if((aname == "HASH_CHAR" && (value.length() == 1))
{
  lastHashChar = value.charAt(0);
}
}
public void doctypeDecl(String name, String publicId, String systemId)
throws Exception
{
  if("MODE".equalsIgnoreCase(name))
  {
    return;
  }
  error("doctype-invalid",name);
}
public void charData(char[] c, int off, int len)
{
  String tag = peekElement();
  String text = new String(c, off, len);
}
public void startElement(String tag)
{
  tag = pushElement(tag);
  if(tag == "WHITESPACE")
  {
    Log.log(Log.WARNING, this, path + ": WHITESPACE rule " +
    "no longer needed");
  }
  if(tag == "MODE")
  {
    mode = jEdit.getMode(modeName);
  }
  if((tag == "MODE" && (mode == null))
  {
    mode = new Mode(modeName);
    jEdit.addMode(mode);
  }
  if(tag == "KEYWORDS")
  {
    keywords = new KeywordMap(rules.getIgnoreCase());
  }
if(tag == "RULES")
{
    rules = new ParserRuleSet(lastSetName,mode);
    rules.setIgnoreCase(lastIgnoreCase);
    rules.setHighlightDigits(lastHighlightDigits);
}

if((tag == "RULES") && (lastDigitRE != null))
{
    rules.setDigitRegexp(new RE(lastDigitRE, lastIgnoreCase
    ? RE.REG_ICASE : 0, RESearchMatcher.RE_SYNTAX_JEDIT));
}

if(((tag == "RULES") && (lastDigitRE != null)) && (REException e))
{
    error("regexp", e);
}

if((tag == "RULES") && (lastEscape != null))
{
    rules.setEscapeRule(ParserRule.createEscapeRule(lastEscape));
}

if(tag == "RULES")
{
    rules.setDefault(lastDefaultID);
    rules.setNoWordSep(lastNoWordSep);
}

public void endElement(String name)
{

    if(name == null)
    {
        return;
    }

    String tag = popElement();

    if((name.equalsIgnoreCase(tag)) && (tag == "MODE"))
    {
        mode.setTokenMarker(marker);
    }

    if((name.equalsIgnoreCase(tag)) && (tag == "PROPERTY"))
    {
        props.put(propName,propValue);
    }

    if((name.equalsIgnoreCase(tag)) && (tag == "PROPS")
        && (peekElement().equals("RULES")))
    {
        rules.setProperties(props);
    }

    if((name.equalsIgnoreCase(tag)) && (tag == "PROPS")
        && (!peekElement().equals("RULES")))
    {
        mode.setProperties(props);
    }

    if((name.equalsIgnoreCase(tag)) && (tag == "PROPS"))
    {
        props = new Hashtable();
    }
}
if ((name.equalsIgnoreCase(tag)) && (tag == "RULES")) {
    rules.setKeywords(keywords);
    marker.addRuleSet(lastSetName, rules);
}

if ((name.equalsIgnoreCase(tag)) && (tag == "TERMINATE")) {
    rules.setTerminateChar(termChar);
}

if ((name.equalsIgnoreCase(tag)) && (tag == "SEQ")) && (lastStart == null) {
    error("empty-tag","SEQ");
    return;
}

if ((name.equalsIgnoreCase(tag)) && (tag == "SEQ") && (lastStart == null)) {
    error("empty-tag","SEQ_REGEXP");
    return;
}

if ((name.equalsIgnoreCase(tag)) && (tag == "SEQ_REGEXP")) {
    rules.addRule(ParserRule.createSequenceRule(lastStart, lastDelegateSet, lastTokenID, lastAtLineStart, lastAtWhitespaceEnd, lastAtWordStart));
}

if ((name.equalsIgnoreCase(tag)) && (tag == "SEQ_REGEXP") && lastStart == null) {
    error("empty-tag","SEQ_REGEXP");
    return;
}

if ((name.equalsIgnoreCase(tag)) && (tag == "SEQ_REGEXP")) {
    rules.addRule(ParserRule.createRegexpSequenceRule(lastHashChar, lastStart, lastDelegateSet, lastTokenID, lastAtLineStart, lastAtWhitespaceEnd, lastAtWordStart, lastIgnoreCase));
}

if ((name.equalsIgnoreCase(tag)) && (tag == "SEQ_REGEXP")) && (REException re) {
    error("regexp",re);
}

if ((name.equalsIgnoreCase(tag)) && (tag == "SPAN")) && (lastBegin == null) {
    error("empty-tag","START");
    return;
}

if ((name.equalsIgnoreCase(tag)) && (tag == "SPAN")) && (lastEnd == null) {
    error("empty-tag","END");
    return;
}

if ((name.equalsIgnoreCase(tag)) && (tag == "SPAN")) {
    rules.addRule(ParserRule.createSpanRule(lastStart, lastEnd, lastDelegateSet, lastTokenID, lastNoLineBreak, lastAtLineStart,}
lastAtWhitespaceEnd, lastAtWordStart, lastExcludeMatch, lastNoWordBreak);}

if((name.equalsIgnoreCase(tag)) && (tag == "SPAN_REGEXP") &&
(lastStart == null)) {
    error("empty-tag", "START");
    return;
}

if((name.equalsIgnoreCase(tag)) && (tag == "SPAN_REGEXP") &&
(lastEnd == null)) {
    error("empty-tag", "END");
    return;
}

if((name.equalsIgnoreCase(tag)) && (tag == "SPAN_REGEXP")
    && (REException re)) {
    error("regexp", re);
}

if((name.equalsIgnoreCase(tag)) && (tag == "EOL_SPAN") &&
(lastStart == null)) {
    error("empty-tag", "EOL_SPAN");
    return;
}

if((name.equalsIgnoreCase(tag)) && (tag == "EOL_SPAN")
    && (lastStart == null)) {
    rules.addRule(ParserRule.createEOLSpanRule(lastStart, lastDelegateSet, lastTokenID, lastAtLineStart, lastAtWhitespaceEnd, lastAtWordStart, lastExcludeMatch, lastNoWordBreak, lastIgnoreCase));
}

if((name.equalsIgnoreCase(tag)) && (tag == "EOL_SPAN_REGEXP") &&
(lastStart == null)) {
    error("empty-tag", "EOL_SPAN_REGEXP");
    return;
}

if((name.equalsIgnoreCase(tag)) && (tag == "EOL_SPAN_REGEXP")
    && (REException re))
```java
if ((name.equalsIgnoreCase(tag) && (tag == "MARK_FOLLOWING")) && (lastStart == null))
    error("empty-tag","MARK_FOLLOWING");
    return;
}
if ((name.equalsIgnoreCase(tag) && (tag == "MARK_FOLLOWING"))
    rules.addRule(ParserRule.createMarkFollowingRule(lastStart,
        lastTokenID,lastAtLineStart,lastAtWhitespaceEnd,lastAtWordStart,
        lastExcludeMatch));
}
if ((name.equalsIgnoreCase(tag) && (tag == "MARK_PREVIOUS")) && (lastStart == null))
    error("empty-tag","MARK_PREVIOUS");
    return;
}
if ((name.equalsIgnoreCase(tag) && (tag == "MARK_PREVIOUS"))
    rules.addRule(ParserRule.createMarkPreviousRule(lastStart,
        lastTokenID,lastAtLineStart,lastAtWhitespaceEnd,
        lastAtWordStart,lastExcludeMatch));
}
if (name.equalsIgnoreCase(tag))
    byte token = Token.stringToToken(tag);
}
if ((name.equalsIgnoreCase(tag) && (token != -1))
    addKeyword(lastKeyword,token);
}
if (!name.equalsIgnoreCase(tag))
    throw new InternalError();
}

public void startDocument()
    marker = new TokenMarker();
    marker.setName(modeName);
    props = new Hashtable();
    pushElement(null);
}
private void addKeyword(String k, byte id)
    if (k == null)
        error("empty-keyword");
        return;
```
if (keywords == null)
{
    return;
}

keywords.add(k, id);

private String pushElement(String name)
{
    if (name != null)
    {
        name = name.intern();
    }

    stateStack.push(name);

    return name;
}

private String peekElement()
{
    return (String) stateStack.peek();
}

private String popElement()
{
    return (String) stateStack.pop();
}

private void error(String msg)
{
    _error(jEdit.getProperty("xmode-error." + msg));
}

private void error(String msg, String subst)
{
    _error(jEdit.getProperty("xmode-error." + msg, new String[] { subst }));
}

private void error(String msg, Throwable t)
{
    _error(jEdit.getProperty("xmode-error." + msg, new String[] { t.toString() }));

    Log.log(Log.ERROR, this, t);
}

private void _error(String msg)
{
    GUIUtilities.error(null,"xmode-error",args);
}
}