DOWNLOAD PDF SYNTAX AND PARSING

Chapter 1 : Syntax Parse Examples

Parsing Syntax This section describes syntax-parse, the syntax/parse library's facility for parsing syntax. Both syntax-parse and the specification facility, syntax classes, use a common language of syntax patterns, which is described in detail in Syntax Patterns.

If your document breaks any of these rules, it will be rejected by most, if not all, XML parsers. Well-Formedness The minimal requirement for an XML document is that it be well-formed, meaning that it adheres to a small number of syntax rules, 6 which are summarized in Table and explained in the following sections. However, a document can abide by all these rules and still be invalid. All attribute values must be quoted single or double quotes. White space in content, including line breaks, is significant by default. All start tags must have corresponding end tags exception: Elements must not overlap; they may be nested, however. This is also technically true for HTML. Each element except the root element must have exactly one parent element that contains it. Element and attribute names are case-sensitive: Legal XML Name Characters An XML Name sometimes called simply a Name is a token that begins with a letter, underscore, or colon but not other punctuation continues with letters, digits, hyphens, underscores, colons, or full stops [periods], known as name characters. Element and attribute names must be valid XML Names. Attribute values need not be. Therefore, authors should not use the colon in XML names except for namespace purposes e. Listing illustrates a number of legal XML Names, followed by three that should be avoided but may or may not be identified as illegal, depending on the XML parser you use, and four that are definitely illegal. This is file name-tests. Since the last three examples in the first group use a colon, they are assumed to be elements in the namespaces identified by the prefixes "kbs", "xlink", and "xsl". Of these, the last two refer to W3C-specified namespaces; xlink: See chapter 5 for a thorough discussion of namespaces. The three debatable examples are xml-price, xml: The four illegal cases are much more clear. The first three, 7price, -price, and. The fourth example is illegal because a space character cannot occur in an XML Name. Most parsers will think this is supposed to be the element named discounted and the attribute named price, minus a required equal sign and value. Therefore, the following elements are all unique and are in no way related to one another in XML: Be sure to remember this when doing string comparisons in code. Uppercase Keywords Since XML is case-sensitive, it should not be surprising that certain special words must appear in a particular case. In general, the keywords that relate to DTDs e. On the other hand, the various strings used in the XML declaration e. Case Conventions or Guidelines When creating your own XML vocabulary, it would be desirable if there were conventions to explain the use of uppercase, lowercase, mixed case, underscores, and hyphens. Unfortunately, no such conventions exist in XML 1. It is a good idea to adopt your own conventions and to apply them consistently, at least across your project, but ideally throughout your entire organization. In fact, the terms UpperCamelCase as I use for elements and lowerCamelCase as I use for attributes are often used to make this distinction more clear. It would be just as reasonable, however, to use all uppercase letters for elements, all lowercase for attributes, and a hyphen to separate multipart terms as in the following examples, or even to use all uppercase for elements and attributes. The most important thing is to pick a convention for your project or your company and to be consistent across developers and applications. Employee with its sex attribute, Address, PhoneNumbers, and so on. That is, all elements are nested within the root element. All descendants of the root, whether immediate children or not, represent the content of the root. We also noted that this document element must be the first element the parser encounters after the XML prolog, which does not contain elements. A somewhat surprising aspect, at least to this author, is that the XML Recommendation does not preclude a recursive root! In other words, it is possible for a root element to be defined in a DTD as containing itself. Although this is not common, it is worth noting. The DTD syntax shown here is formally described in chapter 4. Such empty elements convey information simply by their presence or possibly by their attributes, if any. Optional white space may be used before the two terminating characters. This author prefers to include a space to emphasize empty elements. If the term empty element seems strange to you when attributes are involved, just think in terms of the content of the element. Proper

DOWNLOAD PDF SYNTAX AND PARSING

Nesting of Start and End Tags No overlapping of start and end tags from different elements is permitted. Although this might seem like an obvious requirement, HTML as implemented by major browsers is considerably more forgiving and recovers from improper tag overlap. Correct nesting looks like this: The improper nesting example results in either one or two fatal errors, with a message similar to this depending on the parser: The terms ancestor and descendant are not used in the XML 1. An element is a child of exactly one parent, which is the element that contains it. A parent may have more than one child. Immediate children and also children of a child are descendants of the parent. An element is an ancestor of all its descendants. The root is the ancestor of all elements. Every element is a descendant of the root. Every element has exactly one parent, except the root, which has no parent. Generally, single-word values do not require quotes in HTML. For example, both of these are acceptable and equivalent in HTML: All attribute values must be quoted, even if there are no embedded spaces. Of course, if the attribute value contains double quotes, then you must use single quotes as the delimiter, and vice versa. In the XML 1. This means that the following two examples are not equivalent: The application that invokes the parser can either consider the white space important, ignore it i.

Chapter 2 : Parsing - Wikipedia

Parsing, syntax analysis, or syntactic analysis is the process of analysing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar.

Human languages[edit] It has been suggested that this section be split out into another article titled Natural language parsing. Discuss December Traditional methods[edit] The traditional grammatical exercise of parsing, sometimes known as clause analysis, involves breaking down a text into its component parts of speech with an explanation of the form, function, and syntactic relationship of each part. Techniques such as sentence diagrams are sometimes used to indicate relation between elements in the sentence. Parsing was formerly central to the teaching of grammar throughout the English-speaking world, and widely regarded as basic to the use and understanding of written language. However, the general teaching of such techniques is no longer current. This section needs additional citations for verification. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed. February Learn how and when to remove this template message In some machine translation and natural language processing systems, written texts in human languages are parsed by computer programs[clarification needed]. Human sentences are not easily parsed by programs, as there is substantial ambiguity in the structure of human language, whose usage is to convey meaning or semantics amongst a potentially unlimited range of possibilities but only some of which are germane to the particular case. It is difficult to prepare formal rules to describe informal behaviour even though it is clear that some rules are being followed. The choice of syntax is affected by both linguistic and computational concerns; for instance some parsing systems use lexical functional grammar , but in general, parsing for grammars of this type is known to be NP-complete. Head-driven phrase structure grammar is another linguistic formalism which has been popular in the parsing community, but other research efforts have focused on less complex formalisms such as the one used in the Penn Treebank. Shallow parsing aims to find only the boundaries of major constituents such as noun phrases. Another popular strategy for avoiding linguistic controversy is dependency grammar parsing. Most modern parsers are at least partly statistical ; that is, they rely on a corpus of training data which has already been annotated parsed by hand. This approach allows the system to gather information about the frequency with which various constructions occur in specific contexts. Approaches which have been used include straightforward PCFGs probabilistic context-free grammars , maximum entropy , and neural nets. Most of the more successful systems use lexical statistics that is, they consider the identities of the words involved, as well as their part of speech. However such systems are vulnerable to overfitting and require some kind of smoothing to be effective. As mentioned earlier some grammar formalisms are very difficult to parse computationally; in general, even if the desired structure is not context-free , some kind of context-free approximation to the grammar is used to perform a first pass. Algorithms which use context-free grammars often rely on some variant of the CYK algorithm , usually with some heuristic to prune away unlikely analyses to save time. However some systems trade speed for accuracy using, e. A somewhat recent development has been parse reranking in which the parser proposes some large number of analyses, and a more complex system selects the best option. Psycholinguistics[edit] In psycholinguistics , parsing involves not just the assignment of words to categories, but the evaluation of the meaning of a sentence according to the rules of syntax drawn by inferences made from each word in the sentence. This normally occurs as words are being heard or read. Consequently, psycholinguistic models of parsing are of necessity incremental, meaning that they build up an interpretation as the sentence is being processed, which is normally expressed in terms of a partial syntactic structure. Creation of initially wrong structures occurs when interpreting garden path sentences. This section does not cite any sources. Please help improve this section by adding citations to reliable sources. February Parser[edit] A parser is a software component that takes input data frequently text and builds a data structure â€” often some kind of parse tree , abstract syntax tree or other hierarchical structure, giving a structural representation of the input while checking for correct syntax. The parsing may be preceded or followed by other steps, or these may be combined into a single step. The parser is often preceded by a separate lexical

analyser , which creates tokens from the sequence of input characters; alternatively, these can be combined in scannerless parsing. Parsers may be programmed by hand or may be automatically or semi-automatically generated by a parser generator. Parsing is complementary to templating , which produces formatted output. The input to a parser is often text in some computer language , but may also be text in a natural language or less structured textual data, in which case generally only certain parts of the text are extracted, rather than a parse tree being constructed. An important class of simple parsing is done using regular expressions , in which a group of regular expressions defines a regular language and a regular expression engine automatically generating a parser for that language, allowing pattern matching and extraction of text. In other contexts regular expressions are instead used prior to parsing, as the lexing step whose output is then used by the parser. The use of parsers varies by input. In the case of data languages, a parser is often found as the file reading facility of a program, such as reading in HTML or XML text; these examples are markup languages. In the case of programming languages , a parser is a component of a compiler or interpreter , which parses the source code of a computer programming language to create some form of internal representation; the parser is a key step in the compiler frontend. Programming languages tend to be specified in terms of a deterministic context-free grammar because fast and efficient parsers can be written for them. For compilers, the parsing itself can be done in one pass or multiple passes â€" see one-pass compiler and multi-pass compiler. The implied disadvantages of a one-pass compiler can largely be overcome by adding fix-ups , where provision is made for code relocation during the forward pass, and the fix-ups are applied backwards when the current program segment has been recognized as having been completed. An example where such a fix-up mechanism would be useful would be a forward GOTO statement, where the target of the GOTO is unknown until the program segment is completed. In this case, the application of the fix-up would be delayed until the target of the GOTO was recognized. Conversely, a backward GOTO does not require a fix-up, as the location will already be known. Context-free grammars are limited in the extent to which they can express all of the requirements of a language. Informally, the reason is that the memory of such a language is limited. The grammar cannot remember the presence of a construct over an arbitrarily long input; this is necessary for a language in which, for example, a name must be declared before it may be referenced. More powerful grammars that can express this constraint, however, cannot be parsed efficiently. Thus, it is a common strategy to create a relaxed parser for a context-free grammar which accepts a superset of the desired language constructs that is, it accepts some invalid constructs ; later, the unwanted constructs can be filtered out at the semantic analysis contextual analysis step. For example, in Python the following is syntactically valid code: Overview of process[edit] The following example demonstrates the common case of parsing a computer language with two levels of grammar: The first stage is the token generation, or lexical analysis , by which the input character stream is split into meaningful symbols defined by a grammar of regular expressions. The next stage is parsing or syntactic analysis, which is checking that the tokens form an allowable expression. This is usually done with reference to a context-free grammar which recursively defines components that can make up an expression and the order in which they must appear. However, not all rules defining programming languages can be expressed by context-free grammars alone, for example type validity and proper declaration of identifiers. These rules can be formally expressed with attribute grammars. The final phase is semantic parsing or analysis, which is working out the implications of the expression just validated and taking the appropriate action. In the case of a calculator or interpreter, the action is to evaluate the expression or program, a compiler, on the other hand, would generate some kind of code. Attribute grammars can also be used to define these actions. Types of parsers[edit] The task of the parser is essentially to determine if and how the input can be derived from the start symbol of the grammar. This can be done in essentially two ways: Top-down parsing - Top-down parsing can be viewed as an attempt to find left-most derivations of an input-stream by searching for parse trees using a top-down expansion of the given formal grammar rules. Tokens are consumed from left to right. Inclusive choice is used to accommodate ambiguity by expanding all alternative right-hand-sides of grammar rules. Intuitively, the parser attempts to locate the most basic elements, then the elements containing these, and so on. LR parsers are examples of bottom-up parsers. Another term used for this type of parser is Shift-Reduce parsing. LL parsers and recursive-descent parser are

examples of top-down parsers which cannot accommodate left recursive production rules. Although it has been believed that simple implementations of top-down parsing cannot accommodate direct and indirect left-recursion and may require exponential time and space complexity while parsing ambiguous context-free grammars , more sophisticated algorithms for top-down parsing have been created by Frost, Hafiz, and Callaghan [6] [7] which accommodate ambiguity and left recursion in polynomial time and which generate polynomial-size representations of the potentially exponential number of parse trees. Their algorithm is able to produce both left-most and right-most derivations of an input with regard to a given context-free grammar. An important distinction with regard to parsers is whether a parser generates a leftmost derivation or a rightmost derivation see context-free grammar. LL parsers will generate a leftmost derivation and LR parsers will generate a rightmost derivation although usually in reverse. You can help by converting this article to prose, if appropriate. Editing help is available. January Some of the well known parser development tools include the following. Also see comparison of parser generators.

DOWNLOAD PDF SYNTAX AND PARSING

Chapter 3 : Syntax and Parsing - Paul Gorrell - Google Books

â€¢ Parse will go one way or the other, regardless of words â€¢ We addressed this in one way with unlexicalized grammars (how?) â€¢ Lexicalization allows us to be sensitive to specific words.

The interior nodes are labeled by non-terminal categories of the grammar, while the leaf nodes are labeled by terminal categories. The image below represents a constituency-based parse tree; it shows the syntactic structure of the English sentence John hit the ball: The parse tree is the entire structure, starting from S and ending in each of the leaf nodes John, hit, the, ball. The following abbreviations are used in the tree: S for sentence , the top-level structure in this example NP for noun phrase. The first leftmost NP, a single noun "John", serves as the subject of the sentence. The second one is the object of the sentence. VP for verb phrase , which serves as the predicate V for verb. D for determiner , in this instance the definite article "the" N for noun Each node in the tree is either a root node, a branch node, or a leaf node. Within a sentence, there is only ever one root node. A branch node is a mother node that connects to two or more daughter nodes. A leaf node, however, is a terminal node that does not dominate other nodes in the tree. The leaves are the lexical tokens of the sentence. In the example, S is a parent of both N and VP. A daughter node is one that has at least one node directly above it to which it is linked by a branch of a tree. From the example, hit is a daughter node of V. The terms parent and child are also sometimes used for this relationship. Dependency-based parse trees[edit] The dependency-based parse trees of dependency grammars [4] see all nodes as terminal, which means they do not acknowledge the distinction between terminal and non-terminal categories. They are simpler on average than constituency-based parse trees because they contain fewer nodes. The dependency-based parse tree for the example sentence above is as follows: This parse tree lacks the phrasal categories S, VP, and NP seen in the constituency-based counterpart above. Like the constituency-based tree, constituent structure is acknowledged. Any complete sub-tree of the tree is a constituent. Thus this dependency-based parse tree acknowledges the subject noun John and the object noun phrase the ball as constituents just like the constituency-based parse tree does. Whether the additional syntactic structure associated with constituency-based parse trees is necessary or beneficial is a matter of debate. Phrase markers[edit] Phrase markers, or P-markers, were introduced in early transformational generative grammar , as developed by Noam Chomsky and others. A phrase marker representing the deep structure of a sentence is generated by applying phrase structure rules. Then, this application may undergo further transformations. Phrase markers may be presented in the form of trees as in the above section on constituency-based parse trees , but are often given instead in the form of "bracketed expressions", which occupy less space in the memory. For example, a bracketed expression corresponding to the constituency-based tree given above may be something like:

Chapter 4 : Parse tree - Wikipedia

Deep Parsing Natural language parsing (also known as deep parsing) is a process of analyzing the complete syntactic structure of a sentence. This in In this tutorial, we will learn about what parsing is, its different types, and techniques to automatically infer the parse tree of sentences.

Chapter 5 : Difference between a DOM tree parsing and a syntax tree parsing? - Stack Overflow

Phrases: Syntax and Parsing Definitions. A token is a legal sequence of characters. A phrase is a potentially meaningful sequence of tokens. For example, the following token sequences are potentially meaningful.

Chapter 6 : Parsing: Understanding Text Syntax and Structures, Part 3 | Commonlounge

Parse error: syntax error, unexpected T_STRING, expecting ';' in racedaydvl.com on line Which lists the possible location of a syntax mistake. See the mentioned file name and line number.

DOWNLOAD PDF SYNTAX AND PARSING

Chapter 7 : Syntax and Parsing

*Parse Tree vs. Abstract Syntax Tree. Ambiguous Grammars A grammar can easily be ambiguous. Consider parsing 3 - 4 * 2 + 5 with the grammar e!e+eje ejeeje=e +-3 * 4 2*

Chapter 8 : Markup, Character Data, and Parsing | XML Syntax and Parsing Concepts | InformIT

Query Syntax and Parsing Solr supports several query parsers, offering search application designers great flexibility in controlling how queries are parsed. This section explains how to specify the query parser to be used.

Chapter 9 : XML Syntax Rules | XML Syntax and Parsing Concepts | InformIT

The tree can be a parse tree or an abstract syntax tree. They are both trees, but they differ in how closely they represent the actual code written and the intermediate elements defined by the parser.