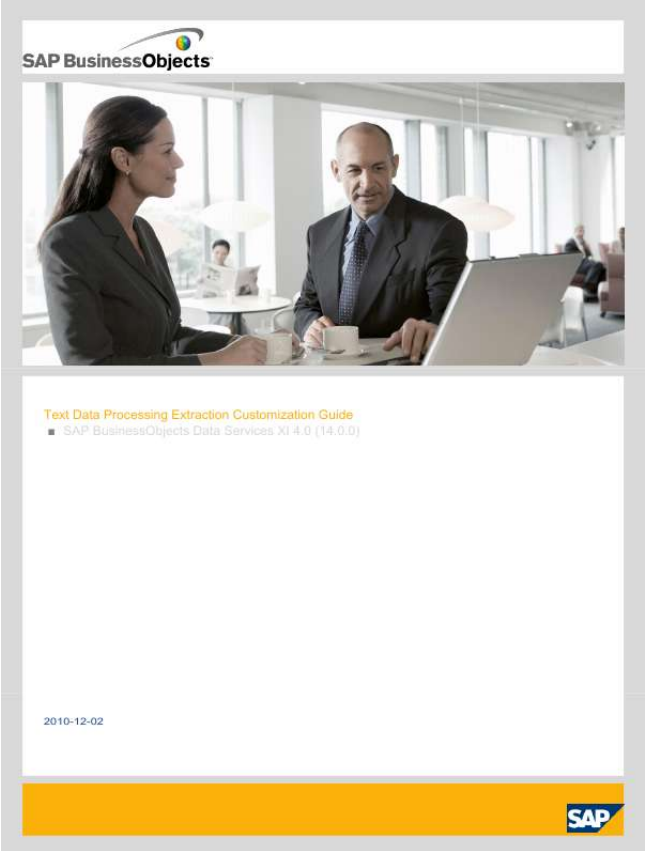

Manual abstract:

@@Data contained in this document serves informational purposes only. National product specifications may vary.These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty. 2010-12-02 Contents Chapter 1 Introduction....

.....
.....
.....

.....
.....
.....
.....

.....
.....
.....
.....

.....
.....
.....
.....

.....
.7 Welcome to SAP BusinessObjects Data Services....
.....
.....

.....
.....
.....
.....

.....
.....
...7 Welcome..
.....

.....
.....
.....
.....

.....
.....
.....
.....

.....
.....
.....
.....

.....
.....
.....
.....7 Documentation set for SAP BusinessObjects Data Services...

.....
.....
.....
.....

.....

.....
.....
.....
.....

.....
..83 Example: Extracting Names Starting with Z...
.....
.....

.....
.....
.....
.....

.....
.....
.....
.....84 Example: Extracting Names of Persons and Awards they Won..

.....
.....
.....
.....

.....
.....
85 Testing Dictionaries and Extraction Rules.....
.....

.....
.....
.....
.....

.....
.....
.....
.....

.87 89 4.1 4.2 4.3 4.3.1 4.3.2 4.3.
3 Chapter 5 Index 5 2010-12-02 Contents 6 2010-12-02 Introduction Introduction 1.1 Welcome to SAP BusinessObjects Data Services 1.1.1 Welcome SAP BusinessObjects Data Services delivers a single enterprise-class solution for data integration, data quality, data profiling, and text data processing that allows you to integrate, transform, improve, and deliver trusted data to critical business processes. It provides one development UI, metadata repository, data connectivity layer, run-time environment, and management console--enabling IT organizations to lower total cost of ownership and accelerate time to value. With SAP BusinessObjects Data Services, IT organizations can maximize operational efficiency with a single solution to improve data quality and gain access to heterogeneous sources and applications. 1.1.2 Documentation set for SAP BusinessObjects Data Services You should become familiar with all the pieces of documentation that relate to your SAP BusinessObjects Data Services product. Document What this document provides Administrator's Guide Customer Issues Fixed Designer Guide Documentation Map Information about administrative tasks such as monitoring, lifecation platform services Installation Guide for UNIX Information for administrators who are responsible for configuring, managing, and maintaining an Information platform services installation. Installation procedures for SAP BusinessObjects Information platform services on a UNIX environment. 9 2010-12-02 Introduction Document What this document provides Information platform services Installation Guide for Windows Installation procedures for SAP BusinessObjects Information platform services on a Windows environment. 1.1.3 Accessing documentation You can access the complete documentation set for SAP BusinessObjects Data Services in several places. 1.1.3.1 Accessing documentation on Windows After you install SAP BusinessObjects Data Services, you can access the documentation from the Start menu. 1.

Choose Start > Programs > SAP BusinessObjects Data Services XI 4.0 > Data Services Documentation. Note: Only a subset of the documentation is available from the Start menu. The documentation set for this release is available in <LINK_DIR>\Doc\Books\en. 2. Click the appropriate shortcut for the document that you want to view. 1.1.3.2 Accessing documentation on UNIX After you install SAP BusinessObjects Data Services, you can access the online documentation by going to the directory where the printable PDF files were installed.
1. Go to <LINK_DIR>/doc/book/en/. 2. Using Adobe Reader, open the PDF file of the document that you want to view. 1.
1.3.3 Accessing documentation from the Web 10 2010-12-02 Introduction You can access the complete documentation set for SAP BusinessObjects Data Services from the SAP BusinessObjects Business Users Support site. 1. Go to http://help.
sap.com. 2. Click SAP BusinessObjects at the top of the page. 3. Click All Products in the navigation pane on the left. You can view the PDFs online or save them to your computer. 1.1.4 SAP BusinessObjects information resources A global network of SAP BusinessObjects technology experts provides customer support, education, and consulting to ensure maximum information management benefit to your business.

Useful addresses at a glance: 11 2010-12-02 Introduction Address Content Customer Support, Consulting, and Education services http://service.sap.com/ Information about SAP Business User Support programs, as well as links to technical articles, downloads, and online forums.

Consulting services can provide you with information about how SAP BusinessObjects can help maximize your information management investment. Education services can provide information about training options and modules. From traditional classroom learning to targeted e-learning seminars, SAP BusinessObjects can offer a training package to suit your learning needs and preferred learning style. SAP BusinessObjects Data Services Community Get online and timely information about SAP BusinessObjects Data Services, including tips and tricks, http://www.sdn.sap.com/irj/sdn/ds additional downloads, samples, and much more.

All content is to and from the community, so feel free to join in and contact us if you have a submission. Forums on SCN (SAP Community Network) http://forums.sdn.sap.com/forum.

jspa?foru mID=305 Search the SAP BusinessObjects forums on the SAP Community Network to learn from other SAP BusinessObjects Data Services users and start posting questions or share your knowledge with the community. Blueprints for you to download and modify to fit your needs. Each blueprint contains the necessary SAP BusinessObjects Data Services project, jobs, data flows, file formats, sample data, template tables, and custom functions to run the data flows in your environment with only a few modifications. SAP BusinessObjects product documentation. Blueprints http://www. sdn.sap.com/irj/boc/blueprints Product documentation http://help.sap.com/businessobjects/ Supported Platforms (Product Availability Matrix) https://service.sap.com/PAM Get information about supported platforms for SAP BusinessObjects Data Services. Use the search function to search for Data Services. Click the link for the version of Data Services you are searching for. 1.

2 Overview of This Guide 12 2010-12-02 Introduction Welcome to the Extraction Customization Guide. SAP BusinessObjects Data Services text data processing software enables you to perform extraction processing and various types of natural language processing on unstructured text. The two major features of the software are linguistic analysis and extraction. Linguistic analysis includes natural-language processing (NLP) capabilities, such as segmentation, stemming, and tagging, among other things. Extraction processing analyzes unstructured text, in multiple languages and from any text data source, and automatically identifies and extracts key entity types, including people, dates, places, organizations, or other information, from the text. It enables the detection and extraction of activities, events and relationships between entities and gives users a competitive edge with relevant information for their business needs. 1.2.1 Who Should Read This Guide This guide is written for dictionary and extraction rule writers. Users of this guide should understand extraction concepts and have familiarity with linguistic concepts and with regular expressions.

This documentation assumes the following: · You understand your organization's text analysis extraction needs. 1.2.2 About This Guide This guide contains the following information: · · · · Overview and conceptual information about dictionaries and extraction rules. How to create, compile, and use dictionaries and extraction rules.

Examples of sample dictionaries and extraction rules. Best practices for writing extraction rules. 13 2010-12-02 Introduction 14 2010-12-02 Using Dictionaries Using Dictionaries A dictionary in the context of the extraction process is a user-defined repository of entities. It can store customized information about the entities your application must find. You can use a dictionary to store name variations in a structured way that is accessible through the extraction process.

A dictionary structure can also help standardize references to an entity. Dictionaries are language-independent. This means that you can use the same dictionary to store all your entities and that the same patterns are matched in documents of different languages. You can use a dictionary for: · · · name variation management disambiguation of unknown entities control over entity recognition 2.1 Entity Structure in Dictionaries This section examines the entity structure in dictionaries. A dictionary contains a number of user-defined entity types, each of which contains any number of entities. For each entity, the dictionary distinguishes between a standard form name and variant names: · Standard form nameThe most complete or precise form for a given entity. For example, United States of America might be the standard form name for that country. A standard form name can have one or more variant names (also known as source form) embedded under it. Variant nameLess standard or complete than a standard form name, and it can include abbreviations, different spellings, nicknames, and so on.

For example, United States, USA and US could be variant names for the same country. In addition, a dictionary lets you assign variant names to a type. For example, you might define a variant type ABBREV for abbreviations. The following figure shows a graphical representation of the dictionary hierarchy and structure of a dictionary entry for United Parcel Service of America, Inc: · 15 2010-12-02 Using Dictionaries The real-world entity, indicated by the circle in the diagram, is associated with a standard form name and an entity type ORGANIZATION and subtype COMMERCIAL. Under the standard form name are name variations, one of which has its own type specified. The dictionary lookup lets you get the standard form and the variant names given any of the related forms. 2.1.1 Generating Predictable Variants The variants United Parcel Service and United Parcel Service of America, Inc. are predictable, and more predictable variants can be generated by the dictionary compiler for later use in the extraction process.

The dictionary compiler, using its variant generate feature, can programmatically generate certain predictable variants while compiling a dictionary. Variant generation works off of a list of designators for entities in the entity type ORGANIZATION in English. For instance, Corp. designates an organization. Variant generation in languages other than English covers the standard company designators, such as AG in German and SA in French.

The variant generation facility provides the following functionality: · · · Creates or expands abbreviations for specified designators. For example, the abbreviation Inc. is expanded to Incorporated, and Incorporated is abbreviated to Inc., and so on. Handles optional commas and periods.

Makes optional such company designators as Inc, Corp. and Ltd, as long as the organization name has more than one word in it. For example, variants for Microsoft Corporation can include: · · · Microsoft Corporation Microsoft Corp. Microsoft Corp 16 2010-12-02 Using Dictionaries Single word variant names like Microsoft are not automatically generated as variant organization names, since they are easily misidentified.

One-word variants need to be entered into the dictionary individually. Variants are not enumerated without the appropriate organization designators. Note: Variant generation is supported in English, French, German, and Spanish. Related Topics · Adding Standard Variant Types 2.1.2 Custom Variant Types You can also define custom variant types in a dictionary.

*Custom variant types can contain a list of variant name pre-modifiers and post-modifiers for a standard form name type. For any variant names of a standard form name to be generated, it must match at least one of the patterns defined for that custom variant type. A variant generation definition can have one or more patterns. For each pattern that matches, the defined generators are invoked. Patterns can contain the wildcards * and ?, that match zero-or-more and a single token respectively. Patterns can also contain one or more capture groups. These are sub-patterns that are enclosed in brackets. The contents of these capture groups after matching are copied into the generator output when referenced by its corresponding placeholder (if any). Capture groups are numbered left to right, starting at 1. A capture group placeholder consists of a backslash, followed by the capture group number.*
The pattern always matches the entire string of the standard form name and never only part of that string. For example, <define-variant_generation type="ENUM_TROOPS" > <pattern string="(?) forces" > <generate string="\1 troops" /> <generate string="\1 soldiers" /> <generate string="\1 Army" /> <generate string="\1 military" /> <generate string="\1 forces" /> </pattern> </define-variant_generation> In the above example this means that: · The pattern matches forces preceded by one token only. Thus, it matches Afghan forces, but not U.S. forces, as the latter contains more than one token.
To capture variant names with more than one token, use the pattern (*) forces. The single capture group is referenced in all generators by its index: \1. The generated variant names are Afghan troops, Afghan soldiers, Afghan Army, Afghan military, and Afghan forces. In principle you do not need the last generator, as the standard form name already matches those tokens. · 17 2010-12-02 Using Dictionaries The following example shows how to specify the variant generation within the dictionary source: <entity_name standard_form="Afghan forces"> <variant_generation type="ENUM_TROOPS" /> \ <variant name="Afghanistan's Army" /> </entity_name> Note: Standard variants include the base text in the generated variant names, while custom variants do not.
Related Topics · Adding Custom Variant Types 2.1.3 Entity Subtypes Dictionaries support the use of entity subtypes to enable the distinction between different varieties of the same entity type. For example, to distinguish leafy vegetables from starchy vegetables. To define an entity subtype in a dictionary entry, add an @ delimited extension to the category identifier, as in VEG@STARCHY. Subtyping is only one-level deep, so TYPE@SUBTYPE@SUBTYPE is not valid. Related Topics · Adding an Entity Subtype 2.1.4 Variant Types Variant names can optionally be associated with a type, meaning that you specify the type of variant name. For example, one specific type of variant name is an abbreviation, ABBREV.

Other examples of variant types that you could create are ACRONYM, NICKNAME, or PRODUCT-ID. 2.1.5 Wildcards in Entity Names Dictionary entries support entity names specified with wildcard pattern-matching elements. These are the Kleene star ("") and question mark ("?") characters, used to match against a portion of the input string. For example, either "* University" or "? University" might be used as the name of an entity belonging to a custom type UNIVERSITY. 18 2010-12-02 Using Dictionaries These wildcard elements must be restricted to match against only part of the input buffer. Consider a pattern "Company *" which matches at the beginning of a 500 KB document. If unlimited matching were allowed, the * wildcard would match against the document's remaining 499+ KB. Note: Using wildcards in a dictionary may affect the speed of entity extraction.*
Performance decreases proportionally with the number of wildcards in a dictionary. Use this functionality keeping potential performance degradations in mind. 2.1.5.
*1 Wildcard Definitions The * and ? wildcards are described as follows, given a sentence: · · * matches any number of tokens greater than or equal to zero within a sentence. ? matches only one token within a sentence. A token is an independent piece of a linguistic expression, such as a word or a punctuation. The wildcards match whole tokens only and not sub-parts of tokens. For both wildcards, any tokens are eligible to be matching elements, provided the literal (fixed) portion of the pattern is satisfied.*
2.1.5.2 Wildcard Usage Wildcard characters are used to specify a pattern, normally containing both literal and variable elements, as the name of an entity. For instance, consider this input: I once attended Stanford University, though I considered Carnegie Mellon University. Consider an entity belonging to the category UNIVERSITY with the variant name " University". The pattern will match any sentence ending with "University". If the pattern were "? University", it would only match a single token preceding "University" occurring as or as a part of a sentence. Then the entire string "Stanford University" would match as intended. However, for "Carnegie Mellon University", it is the substring "Mellon University" which would match: "Carnegie" would be disregarded, since the question mark matches one token at mostand this is probably not the intended result.*

If several patterns compete, the extraction process returns the match with the widest scope. Thus if a competing pattern " University" were available in the previous example, "Carnegie Mellon University" would be returned, and "Mellon University" would be ignored. Since * and ? are special characters, "escape" characters are required to treat the wildcards as literal elements of fixed patterns. The back slash "\" is the escape character. Thus "*" represents the literal asterisk as opposed to the Kleene star. A back slash can itself be made a literal by writing "\\". 19 2010-12-02 Using Dictionaries Note: Use wildcards when defining variant names of an entity instead of using them for defining a standard form name of an entity. Related Topics · Adding Wildcard Variants 2.2 Creating a Dictionary To create a dictionary, follow these steps: 1.*

Create an XML file containing your content, formatted according to the dictionary syntax.
2. Run the dictionary compiler on that file. Note: For large dictionary source files, make sure the memory available to the compiler is at least five times the size of the input file, in bytes. Related Topics · Dictionary XSD · Compiling a Dictionary 2.3 Dictionary Syntax 2.
3.1 Dictionary XSD The syntax of a dictionary conforms to the following XML Schema Definition (XSD). When creating your custom dictionary, format your content using the following syntax, making sure to specify the encoding if the file is not UTF-8. <?xml version="1.0" encoding="UTF-8"?> !-Copyright 2010 SAP AG.

By default, the standard form name is used as the starting point for variant generation. Need not be explicitly closed. [Optional] Specifies the language to use for standard variant generation, in lower case, for example, "english". If this option is not specified in the dictionary, the language specified with the compiler command is used, or it defaults to English when there is no language specified in either the dictionary or the compiler command. [Required] Types supported are standard or the name of a custom variant generation defined earlier in the dictionary. [Optional] Specifies text other than the standard form name to use as the starting point for the computation of variants. language variant_generation type base_text define-variant_genera tion pattern generate Specifies custom variant generation. Specifies the pattern that must be matched to generate custom variants. Specifies the exact pattern for custom variant generation within each generate tag.

Related Topics · Adding Custom Variant Types · Formatting Your Source 24 2010-12-02 Using Dictionaries 2.3.2 Guidelines for Naming Entities This section describes several guidelines for the format of standard form and variant names in a dictionary: · · · · You can use any part-of-speech (word class). Use only characters that are valid for the specified encoding. The symbols used for wildcard pattern matching, "?" and "", must be escaped using a back slash character ("\") .*

Any other special characters, such as quotation marks, ampersands, and apostrophes, can be escaped according to the XML specification. The following table shows some such character entities (also used in HTML), along with the correct syntax: Character < > & " ' Description Less than (<) sign Greater than (>) sign Ampersand (&) sign Quotation marks (") Apostrophe (') Dictionary Entry < > & " ' 2.3.3 Character Encoding in a Dictionary A dictionary supports all the character encodings supported by the Xerces-C XML parser. If you are creating a dictionary to be used for more than one language, use an encoding that supports all required languages, such as UTF-8.

For information on encodings supported by theXerces-C XML parser, see http://xerces.apache.org/xerces-c/faq-parse-3.html#faq-16. 25 2010-12-02 Using Dictionaries The default input encoding assumed by a dictionary is UTF-8. Dictionary input files that are not in UTF8 must specify their character encoding in an XML directive to enable proper operation of the configuration file parser, for example: <?xml version="1.0" encoding="UTF-16" ?>. If no encoding specification exists, UTF-8 is assumed. For best results, always specify the encoding. Note: CP-1252 must be specified as windows-1252 in the XML header element.

The encoding names should follow the IANA-CHARSETS recommendation. 2.3.4 Dictionary Sample File Here is a sample dictionary file. <?xml version="1.0" encoding="windows-1252"?> <dictionary> <entity_category name="ORGANIZATION@COMMERCIAL"> <entity_name standard_form="United Parcel Service of America, Incorporated"> <variant name="United Parcel Service" /> <variant name="U.P.S." type="ABBREV" /> <variant name="UPS" /> <variant_generation type="standard" language="english" /> </entity_name> </entity_category> </dictionary> Related Topics · Entity Structure in Dictionaries 2.3.

5 Formatting Your Source Format your source file according to the dictionary XSD. The source file must contain sufficient context to make the entry unambiguous. The required tags for a dictionary entry are: · entity_category · entity_name Others can be mentioned according to the desired operation. If tags are already in the target dictionary, they are augmented; if not, they are added. The add operation never removes tags, and the remove operation never adds them.

Related Topics · Dictionary XSD 26 2010-12-02 Using Dictionaries 2.3.6 Working with a Dictionary This section provides details on how to update your dictionary files to add or remove entries as well as update existing entries. 2.3.

6.1 Adding an Entity To add an entity to a dictionary: · Specify the entity's standard form under the relevant entity category, and optionally, its variants. The example below adds two new entities to the ORGANIZATION@COMMERCIAL category: <?xml version="1.0" encoding="windows-1252"?> <dictionary> <entity_category name="ORGANIZATION@COMMERCIAL"> <entity_name standard_form="Seventh Generation Incorporated"> <variant name="Seventh Generation"/> <variant name="SVNG"/> </entity_name> <entity_name standard_form="United Airlines, Incorporated"> <variant name="United Airlines, Inc."/> <variant name="United Airlines"/> <variant name="United"/> </entity_name> </entity_category> </dictionary> 2.3.6.2 Adding an Entity Type To add an entity type to a dictionary: · Include a new entity_category tag For example: <?xml version="1.0" encoding="windows-1252"?> <dictionary> <entity_category name="YOUR_ENTITY_TYPE"> ..

. </entity_category> </dictionary> 27 2010-12-02 Using Dictionaries 2.3.6.3 Adding an Entity Subtype To add an entity subtype to a dictionary: · include a new entity_category tag, using an @ delimited extension to specify the subtype. For example: <?xml version="1.0" encoding="windows-1252"?> <dictionary> <entity_category name="VEG@STARCHY"> ... </entity_category> </dictionary> 2.

3.6.4 Adding Variants and Variant Types To add variants for existing entities: · Include a variant tag under the entity's standard form name. Optionally, indicate the type of variant. For example: <?xml version="1.

0" encoding="windows-1252"?> <dictionary> <entity_category name="ORGANIZATION@COMMERCIAL"> <entity_name standard_form="Apple Computer, Inc."> <variant name="Job's Shop" type="NICKNAME" /> </entity_name> </entity_category> </dictionary> 2.3.6.5 Adding Standard Variant Types To add standard variant types, · Include a variant_generation tag in an entity's definition.

For example: <?xml version="1.0" encoding="windows-1252"?> <dictionary> <entity_category name="ORGANIZATION@COMMERCIAL"> <entity_name standard_form="Seventh Generation Inc"> <variant_generation type="standard" language="english" /> </entity_name> </entity_category> </dictionary> 28 2010-12-02 Using Dictionaries If you want variants generated for both standard form and variant names, use more than one variant_generation tag.

In the language attribute, specify the language for which variant generation applies; standard variant generations are language dependent. Variant generation is supported in English, French, German and Spanish. 2.3.6.6 Adding Custom Variant Types To add custom variant types, · Define a name with the list of variant generations. For example: <?xml version="1.0" encoding="windows-1252"?> <dictionary> <define-variant_generation type="ENUM_INC" > <pattern string="(*) Inc" > <generate string="\1 Inc" /> <generate string="\1, Inc" /> <generate string="\1 Incorporated" /> <generate string="\1, Incorporated" /> </pattern> </define-variant_generation> <entity_category name="ORGANIZATION@COMMERCIAL"> <entity_name standard_form="Seventh Generation Inc" <variant_generation type="ENUM_INC" /> <variant_generation type="ENUM_INC" base_text="7th Generation Inc" /> <variant_generation type="ENUM_INC" base_text="Seven Generation Inc" /> <variant name="7th Generation Inc" /> <variant name="Seven Generation Inc" /> </entity_name> </entity_category> </dictionary> The example should match the following expressions, with "Seventh Generation Inc" as the standard form name: · · · · · · · · · · Seventh Generation Inc Seventh Generation, Inc Seventh Generation Incorporated Seventh Generation, Incorporated 7th Generation Inc 7th Generation, Inc 7th Generation Incorporated 7th Generation, Incorporated Seven Generation Inc Seven Generation, Inc 29 2010-12-02 Using Dictionaries · · Seven Generation Incorporated Seven Generation, Incorporated The pattern string for the variant generation includes the following elements used specifically for custom variant generation types: · Pattern-This is the content specified in the <pattern> tag, within parenthesis, typically a token wildcard, as in the example above.

The content is applied on the standard form name of the entity, gets repeated in the variants, and can appear before and after the user-defined content, numbered left to right within the generate tag, as in the example below. Note: Custom variants generate patterns exactly as specified within each generate tag, therefore the static content itself is not generated unless you include a generate tag for that specific pattern, as indicated by the second pattern tag in the example below. · User-defined generate strings--This is the content that changes as specified in each generate tag, as shown in the examples. This is literal content that cannot contain wildcards. <?xml version="1.0" encoding="windows-1252"?> <dictionary> <entity_category name="ORGANIZATION@COMMERCIAL"> <entity_name standard_form="ABC Corporation of America"> <define-variant_generation type="ENUM_CORP" > <pattern string="(*) Corpo (*)" > <generate string="\1 Corp \2" /> <generate string="\1 Corp. \2" /> <generate string="\1 Corpo \2" /> <generate string="\1 Corporation \2" /> </pattern> <pattern string=""> <generate string="\1" /> </pattern> </define-variant_generation> </entity_name> </entity_category> </dictionary> Note: The variants pattern must cover the entire standard form name, not a substring of it. Related Topics · Dictionary XSD 2.3.6.*

*7 Adding Wildcard Variants To add wildcard variants: · Define a name with a wildcard in it. 30 2010-12-02 Using Dictionaries For example: <?xml version="1.0" encoding="windows-1252"?> <dictionary> <entity_category name="BANKS"> <entity_name standard_form="American banks"> <variant name="Bank of * America" /> </entity_name> </entity_category> ...*

</dictionary> This wildcard entry matches entities like Bank of America, Bank of Central America, Bank of South America, and so on. 2.4 Compiling a Dictionary You create a new dictionary or modify an existing one by composing an XML file containing expressions as per the dictionary syntax. To replace dictionary material, first delete the elements to be changed, then add replacements. When your source file is complete, you pass it as input to the dictionary compiler (tf-ncc).

The dictionary compiler compiles a dictionary binary file from your XML-compliant source text. Note: For large dictionary source files, make sure the memory available to the compiler is at least five times the size of the input file, in bytes. Related Topics · Dictionary XSD · Working with a Dictionary 2.4.1 Command-line Syntax for Compiling a Dictionary The command line syntax to invoke the dictionary compiler is: tf-ncc [options] <input filename> where, [options] are the following optional parameters. 31 2010-12-02 Using Dictionaries <input_file> specifies the dictionary source file to be compiled. This argument is mandatory. Syntax Description Specifies the directory where the language modules are stored. -d <language_module_directory> This is a mandatory option. You must specify this option along with the language directory location.

The default location for the language directory is, ../TextAnaly sis/languages relative to the LINK_DIR/bin directory. Requests that tf-ncc add the entities in the input file to an existing compiled dictionary. Note: If -o is specified, the <additions_file> remains unchanged and the output file contains the merged <additions_file> and the input file. If no output file is specified then the output is placed in the <addi tions_file> file. Equivalent to -a except that the elements in the input file will be removed from the existing compiled dictionary file. The path and filename of the resulting compiled dictionary. If none is supplied the file lxtf2.nc is created in the current directory.

Indicates verbose. Shows progress messages. Specifies the default language for standard variant generation. If no language is specified in the tag or on the command line, english will be used. -a <additions_file> -r <removals_file> -o <output filename> -v -l <language> Note: Encoding must be specified by a <?xml encoding=X> directive at the top of the source file or it is assumed to be utf-8.

-config_file <filename> 32 2010-12-02 Using Dictionaries Syntax Description Specifies the dictionary configuration file. The default configuration file tf.nc-config is located at ../TextAnalysis/languages relative to the LINK_DIR/bin directory.

Generates case-sensitive variants. -case_sensitive Note: If you include this command, you should include every variant of the word. Generates case-insensitive variants. Note: Use caution when compiling a dictionary in case-insensitive mode as spurious entries may result. For instance, if either of the proper nouns May or Apple were listed in a case-insensitive dictionary, then the verb may and the fruit apple would be matched. Displays the compiler version. Prints a help message. -case_insensitive -version -h, -help, --help Related Topics · Dictionary XSD 2.4.2 Adding Dictionary Entries To add entries to an existing dictionary, 1.

Go to the directory where the dictionary compiler is installed. This will be <LINK_DIR>/bin directory; where <LINK_DIR> is your Data Services installation directory. For example, C:/Program Files/SAP Business Objects/Data Services 2. Create an XML file <input file> containing entries to be added. 33 2010-12-02 Using Dictionaries 3. Invoke the dictionary compiler with the -a command for the add operation. tf-ncc -d ../TextAnalysis/languages -a english.nc additions.
xml where, english.nc is a compiled dictionary. additions.xml is the xml file that contains the new entries. Note: This command enables you to merge two dictionaries.
Related Topics · Command-line Syntax for Compiling a Dictionary 2.4.3 Removing Dictionary Entries Removing entries from a dictionary is similar to adding them. To remove dictionary entries, 1. Go to the directory where the dictionary compiler is installed.
This will be <LINK_DIR>/bin directory; where <LINK_DIR> is your Data Services installation directory. For example, C:/Program Files/SAP Business Objects/Data Services 2. Create an XML file <input file> containing material to be removed. 3. Invoke the dictionary compiler with the -r command for the remove operation. tf-ncc -d ../TextAnalysis/languages -r english.nc removals.xml Note: The remove operation applies to the most embedded level specified and anything embedded below it.

Related Topics · Command-line Syntax for Compiling a Dictionary 2.4.4 Removing Standard Form Names from a Dictionary To remove standard form names from a dictionary, 1. Create an XML file <input file> specifying the standard form names without variants. 34 2010-12-02 Using Dictionaries 2. Go to the directory where the dictionary compiler is installed. This will be <LINK_DIR>/bin directory; where <LINK_DIR> is your Data Services installation directory. For example, C:/Program Files/SAP Business Objects/Data Services 3. Invoke the dictionary compiler with the -r command for the remove operation. For example, if you invoked the dictionary compiler with the -r command for the following file, it would remove Acme, Inc.
and any variants from the specified dictionary. <?xml version="1.0" encoding="windows-1252"?> <dictionary> <entity_category name="ORGANIZATION"> <entity_name standard_form="Acme, Inc."> </entity_name> </entity_category> </dictionary> Related Topics · Command-line Syntax for Compiling a Dictionary 35 2010-12-02 Using Dictionaries 36 2010-12-02 Using Extraction Rules Using Extraction Rules Extraction rules (also referred to as CGUL rules) are written in a pattern-based language that enables you to perform pattern matching using character or token-based regular expressions combined with linguistic attributes to define custom entity types. You can create extraction rules to: · · · · · Extract complex facts based on relations between entities and predicates (verbs or adjectives).
Extract entities from new styles and formats of written communication. Associate entities such as times, dates, and locations, with other entities (entity-to-entity relations). Identify entities in unusual or industry-specific language. For example, use of the word crash in computer software versus insurance statistics. Capture facts expressed in new, popular vernacular.
For example, recognizing sick, epic, and fly as slang terms meaning good. 3.1 About Customizing Extraction The software provides tools you can use to customize extraction by defining extraction rules that are specific to your needs. To create extraction rules, you write patterns using regular expressions and linguistic attributes that define categories for the entities, relations, and events you need extracted. These patterns are written in CGUL (Custom Grouper User Language), a token-based pattern matching language. These patterns form rules that are compiled by the rule compiler (tf-cgc). The rule compiler checks CGUL syntax and logs any syntax errors. Extraction rules are processed in the same way as pre-defined entities. It is possible to define entity types that overlap with pre-defined entities. Once your rules are created, saved into a text file, and compiled into a binary (.

fsm) file, you can test them using the Entity Extraction transform in the Designer. 37 2010-12-02 Using Extraction Rules Following diagram describes a basic workflow for testing extraction rules: Related Topics · Compiling Extraction Rules · Designer Guide: Transforms, Text Data Processing transforms, To add a text data processing transform to a data flow 3.2 Understanding Extraction Rule Patterns With CGUL, you define extraction rules using character or token-based regular expressions combined with linguistic attributes. The extraction process does not extract patterns that span across paragraphs. Therefore, patterns expressed in CGUL represent patterns contained in one paragraph; not patterns that start in one paragraph and end in the next. Tokens are at the core of the CGUL language. The tokens used in the rules correspond with the tokens generated by the linguistic analysis. Tokens express a linguistic expression, such as a word or punctuation, combined with its linguistic attributes. In CGUL, this is represented by the use of literal strings or regular expressions, or both, along with the linguistic attributes: part-of-speech (POS) and STEM STEM is a base form a word or standard form that represents a set of morphologically related words. This set may be based on inflections or derivational morphology.
The linguistic attributes supported vary depending on the language you use. For information about the supported languages and about the linguistic attributes each language supports, refer to the Text Data Processing Language Reference Guide. 38 2010-12-02 Using Extraction Rules 3.2.1 CGUL Elements CGUL rules are composed of the elements described in the following table.
Each element is described in more detail within its own section. Element Description These directives define character classes (#define), subgroups (#subgroup), and facts (#group). For more information, see CGUL Directives . Tokens can include words in their literal form (cars) or regular expressions (car.

), their stem (car), their part-of-speech (Nn), or any of these elements combined: Tokens Tokens are delimited by angle brackets (< >) . <car., POS:Nn> <STEM:fly, POS:V> For more information, see Tokens. CGUL Directives 39 2010-12-02 Using Extraction Rules Element Description The following operators are used in building character patterns, tokens, and entities. These include the following quantifier operators: Iteration Operators +, *, ?, { m}, {n,m}.For more information, see Iteration Operators Supported in CGUL. These include the following: · Character wildcard (.) Operators · · · Standard Operators · · Alternation (\|) Escape character (\) Character and string negation (^ and ~) Subtraction (-) Character Classifier (\p{val ue}) For more information, see, Standard Operators Valid in CGUL. Grouping and Containment Operators 40 2010-12-02 Using Extraction Rules Element Description These include the following operators: [range] , (item), {expres sion}, where, · [range] defines a range of characters · (item) groups an expression together to form an item that is treated as a unit {expression} groups an expression together to form a single rule, enabling the rule writer to wrap expressions into multiple lines · For more information, see Grouping and Containment Operators Supported in CGUL. These include the following markers: [SN] Sentence [NP]Noun phrase [VP]Verb phrase [CL] and [CC]Clause and clause container Expression Markers [OD]Context [TE]Entity [UL]and [UC]Unordered list and contiguous unordered list [P]Paragraph For more information, see Expression Markers Supported in CGUL. 41 2010-12-02 Using Extraction Rules Element Description The following match filters can be used to specify whether a CGUL preceding token expression matches the longest or shortest pattern that applies.

Match filters include: · Longest match · Shortest match · List (returns all matches) For more information, see Match Filters Supported in CGUL. #include directives are used to include other CGUL source files and .pdc files. You must include CGUL source files and .pdc files before you use the extraction rules or predefined classes that are contained in the files you include. For more information, see Including Files in a Rule File. #lexicon directives are used to include the contents of a dictionary file that contains a list of single words, delimited by new lines. For more information, see Including a Dictionary in a Rule File. Comments are marked by an initial exclamation point (!). When the compiler encounters a ! it ignores the text that follows it on the same line.
Include Directive Lexicon Directive Comments 3.2.2 CGUL Conventions CGUL rules must follow the following conventions: · · Rules are case-sensitive by default. However, you can make rules or part of rules case insensitive by using character classifiers. A blank space is required after #define, #subgroup, #group, #include,#lexicon, and between multiple key-value pairs.
Otherwise, blank spaces are rejected unless preceded by the escape character. Names of CGUL directives (defined by #define, #subgroup, or #group) must be in alphanumeric ASCII characters with underscores allowed. You must define item names before using them in other statements. · · 42 2010-12-02 Using Extraction Rules Related Topics · Character Classifier (\p) 3.3 Including Files in a Rule File You can include CGUL source files and .
pdc files anywhere within your rules file. However, the entry point of an #include directive should always precede the first reference to any of its content. The syntax for the included files is checked and separate error messages are issued if necessary. Note: The names defined in included files cannot be redefined. Syntax #include <filename> #include "filename" where filename is the name of the CGUL rules file or .pdc file you want to include. You can use an absolute or relative path for the file name, but it is recommended that you use an absolute path. Note: The absolute path is required if: · · · The input file and the included file reside in different directories. The input file is not stored in the directory that holds the compiler executable. The file is not in the current directory of the input file.

3.3.1 Using Predefined Character Classes The extraction process provides predefined character and token classes for each language supported by the system. Both character classes and token classes are stored in the <language>.pdc files. To use these classes, use the#include statement to include the .pdc file in your rule file. 3.4 Including a Dictionary in a Rule File You can include a dictionary within your rules file. However, the entry point of a #lexicon directive should always precede the first reference to its content.
43 2010-12-02 Using Extraction Rules The dictionary file consists of single words separated by new lines. The compiler interprets the contents of a dictionary as a #define directive with a rule that consists of a bracketed alternation of the items listed in the file. Syntax #lexicon name "filename" where, name is the CGUL name of the dictionary. filename is the name of the file that contains the dictionary. Example #lexicon FRUITLIST "myfruits.
txt" ... #group FRUIT: <STEM:%(FRUITLIST)> Description In this example, the dictionary is compiled as a #define directive named FRUITLIST and is contained in a file called myfruits.txt.
Later in the rule file the dictionary is used in the FRUIT group. If myfruits.txt contains the following list: apple orange banana peach strawberry The compiler interprets the group as follows: #group FRUIT: <STEM:apple\orange\banana\peach\strawberry> Note: A dictionary cannot contain entries with multiple words. In the preceding example, if wild cherry was included in the list, it would not be matched correctly. 3.5 CGUL Directives CGUL directives define character classes (#define), tokens or group of tokens (#subgroup), and entity, event, and relation types (#group). The custom entities, events, and relations defined by the #group directive appear in the extraction output. The default scope for each of these directives is the sentence. The relationship between items defined by CGUL directives is as follows: · · · Character classes defined using the #define directive can be used within #group or #subgroup Tokens defined using the #subgroup directive can be used within #group or #subgroup Custom entity, event, and relation types defined using the #group directive can be used within #group or #subgroup 44 2010-12-02 Using Extraction Rules Related Topics · CGUL Conventions · Writing Directives · Using the #define Directive · Using the #subgroup Directive · Using the #group Directive · Using Items in a Group or Subgroup 3.

5.

1 Writing Directives You can write directives in one line, such as: #subgroup menu: (<mashed><potatoes>)|(<Peking><Duck>)|<tiramisu> or, you can span a directive over multiple lines, such as: #subgroup menu: { (<mashed><potatoes>)| (<Peking><Duck>)| <tiramisu> } To write a directive over multiple lines, enclose the directive in curly braces{}. 3.5.2 Using the #define Directive The #define directive is used to denote character expressions. At this level, tokens cannot be defined. These directives represent user-defined character classes. You can also use predefined character classes. Syntax #define name: expression where, name is the name you assign to the character class. colon (:) must follow the name. expression is the literal character or regular expression that represents the character class.

Example #define ALPHA: [A-Za-z] #define URLBEGIN: (www\.|http\:) 45 2010-12-02 Using Extraction Rules #define VERBMARK: (ed|ing) #define COLOR: (red|blue|white) Description ALPHA represents all uppercase and lowercase alphabetic characters. URLBEGIN represents either www. or http: for the beginning of a URL address VERBMARK represents either ed or ing in verb endings COLOR represents red, blue, or white Note: A #define directive cannot contain entries with multiple words. In the preceding example, if navy blue was included in the list, it would not be matched correctly.

Related Topics · Using Predefined Character Classes 3.5.3 Using the #subgroup Directive The #subgroup directive is used to define a group of one or more tokens. Unlike with #group directives, patterns matching a subgroup do not appear in the extraction output. Their purpose is to cover sub-patterns used within groups or other subgroups.

Subgroups make #group directives more readable. Using subgroups to define a group expression enables you to break down patterns into smaller, more precisely defined, manageable chunks, thus giving you more granular control of the patterns used for extraction. In #subgroup and #group statements alike, all tokens are automatically expanded to their full format: <literal, stem, POS> Note: A rule file can contain one or more subgroups. Also, you can embed a subgroup within another subgroup, or use it within a group. Syntax #subgroup name:<expression> where, name is the name you are assigning the token colon (:) must follow the name <expression> is the expression that constitutes the one or more tokens, surrounded by angle brackets <>, if the expression includes an item name, the item's syntax would be: %(item) 46 2010-12-02 Using Extraction Rules Example #subgroup Beer: <Stella>|<Jupiler>|<Rochefort> #subgroup BeerMod: %(Beer) (<Blonde>|<Trappist>) #group BestBeer: %(BeerMod) (<Premium>|<Special>) Description The Beer subgroup represents specific brands of beer (Stella, Jupiler, Rochefort). The BeerMod subgroup embeds Beer, thus it represents any of the beer brands defined by Beer, followed by the type of beer (Blonde or Trappist). @@@@Using the following input... @@@@@@@@Custom facts and entity types appear in the extraction output.

@@For example, to distinguish leafy vegetables from starchy vegetables. @@Note: A rule file can contain one or more groups. Also, you can embed a group within another group. @@@@#group A (para graph="[1]"): ... #group C (paragraph="[14]"): ... @@The value is either sentence or paragraph. @@@@@@#group Z (alert="Or ange"): ... @@@@@@@@INITCAP is then used within a group, MRPERSON. @@@@@@@@In this case, an error message is issued: #subgroup INITCAP: <%(UPPER)%(LOWER)+> #group MRPERSON: <Mr\.

> <%(INITCAP)> 50 2010-12-02 Using Extraction Rules 3.6 Tokens Tokens (also referred as syntactic units) are at the core of CGUL. They express an atomic linguistic expression, such as a word or punctuation, combined with its linguistic attributes: part-of-speech (POS) and STEM. CGUL uses tokens to represent the linguistic expressions to be matched. 3.

6.1 Building Tokens You can specify tokens that express a broad variety of patterns to help you define custom entity, event, and relation types and extract related entities. To build tokens, you use any of three optional fields: string, STEM, and POS (part-of-speech) tags. The string and STEM fields can use all valid CGUL operators to define patterns, while the POS field only accepts alternation and string negation. Syntax <string, STEM:stem, POS:"pos_tag"> <string, STEM:stem, POS:pos_tag> where, string can be a word, or a regular expression that represents a word pattern. stem can be a word stem or a regular expression that represents a stem pattern. pos_tag is a part-of-speech tag. Note: The part-of-speech tag can be expressed within or without quotation marks. The behavior is as follows: · · · "pos_tag" (within quotation marks) The POS value matches exactly. For example <POS:"Adj"> matches only Adj, and <POS: "Adj"|"Punct"> matches Adj or Punct.

Each part-of-speech value requiring an exact match must be surrounded by quotes. Hence an expression such as <POS:"Adj|Punct"> is syntactically invalid. pos_tag (no quotation marks) The POS value includes the umbrella part-of-speech and all its expansions. for example <POS:Adj> matches Adj, and Adj-Sg, Adj-Pl, and so on. Tokens conform to the following syntactic rules: · Tokens must be delimited by angle brackets <> #subgroup BADDOG2: bulldog This is literally a character sequence and the string is not expanded with a STEM and a POS, therefore, it does not match the token bulldog. The proper notation is: #subgroup DOG2: <bulldog> 51 2010-12-02 Using Extraction Rules · Tokens are composed of three optional fields: Literal, STEM, and POS For example, <activat.+, STEM:activat.+, POS:V> · The fields within the token are optional and are delimited by commas. Note: Fields that are not defined are expanded to the following defaults: .+ for literal, ANYSTEM for STEM, and ANYPOS for POS.

Hence, they are assumed to be any possible value for that specific field. For example, <STEM:be, POS:V> means any token that has a stem be and is a verb · · · · STEM and POS must be written in all uppercase, followed by a colon (:), and separated by a comma. POS can be any part-of-speech tag. Blank spaces are ignored either within or between tokens, thus <POS:V> and <POS: V> are the same, and <apple><tree> and <apple> <tree> are the same. Items that are already defined as tokens cannot be surrounded by angle brackets (<>) when used as part of another definition.

For example, the following #group statement is incorrect, and generates an error because COLOR is already defined as a token.