RSTModules

User's Guide

Florian Zipser <saltnpepper@lists.hu-berlin.de>
Mario Frank <saltnpepper@lists.hu-berlin.de>
INRIA
SFB 632 Information Structure / D1 Linguistic Database
Humboldt-Universität zu Berlin
Universität Potsdam

RSTModules: User's Guide

by Florian Zipser, Mario Frank, , , , and

Version ${project.version}

Table of Contents

Foreword v
1. Overview 1
2. rstImporter 2
    Mapping to Salt 2
    Properties 3
        rstImporter.tokenize 3
        rstImporter.nodeKindName 3
        rstImporter.nodeTypeName 3
        rstImporter.relationTypeName 3
        rstImporter.relationTypeName 3

List of Tables

1.1. pepper modules contained in this project 1
2.1. properties to customize importer behaviour 3

Foreword

The intention of this document is first to give a guide to the user of how to use the here mentioned pepper modules and how to utilize a mapping performed by them. Second this document shall give a closer view in the details of such a mapping in a declarative way, to give the user a chance to understand how specific data will be mapped by the presented pepper modules.

Chapter 1. Overview

This project contains the pepper modules listed in Table 1.1, “pepper modules contained in this project”. A single module can be identified via its coordinates (module-name, format-name, format-version) also given in Table 1.1, “pepper modules contained in this project”. You can use these coordinates in a pepper workflow description file to identify the modules in a pepper conversion process. A description of how to model a workfloe description file can be found under https://korpling.german.hu-berlin.de/saltnpepper/.

RST stands for Rhetorical Structure Theory and descibes a linguistical analysis. Next to to the rethorical structure theory, a tool named RST Tool was developed see: http://www.wagsoft.com/RSTTool/. This tool produces rs3 files, which are imported by the here provided RSTImporter. The RSTImporter makes use of the rst-api (see: https://korpling.german.hu-berlin.de/p/projects/rst-api/wiki).

Table 1.1. pepper modules contained in this project

Name of pepper module	Type of pepper module	Format (if module is im- or exporter)
rstImporter	importer	1.0

Chapter 2. rstImporter

Rst is a theory on phrase-like constructs, which mean, that a token in rst is a phrase. When mapping these data to Salt, an SToken object represents such a phrase. Since in most cases this is not the wanted behaviour, the RSTImporter provides a mechanism to tokenize the phrases into word-like structures. Therefore it takes use of the tokenizer provided by Salt which copies the way of tokenization from the TreeTagger (see: www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/).

Rst in general forms a tree-like structure, which is mapped as this to a Salt model. For more information on the Rst model which is used, please take a look to the rst-api, where the RSTImporter is based on (see: https://korpling.german.hu-berlin.de/p/projects/rst-api/wiki).

Mapping to Salt

In rst, the primary data is represented by the textual values contained by `Segment` objects. The text of all `Segment` objects is mapped the sText value of exactly one `STextualDS` object. The text of a `Segment` object is concatenated to the sText prefixed by the blank seperator ' '. Imagine a text of segment1 'Is this example' and the text of segment2 'more complicated than it is supposed to be', than the sText value is 'Is this example more complicated than it is supposed to be'. The separator can be user defined or set to '' (empty) by taking use of the property rstImporter.relationTypeName.

A `Segment` object itself is mapped to `sStructure`. As already mentioned, the RSTImporter provides the mechanism to tokenize a text in word-like tokens. Therefore, the text covered by a `Segment` object is tokenized to `SToken` objects. These `SToken` objects than are be related to `SStructure` object representing the `Segment` object via a `SDominanceRelation`. In case of a `Segment` object covers the text 'Is this example', the tokenizer will create `SToken` objects covering the text 'Is', 'this' and 'example'. The three tokens are dominated by one `SStructure` object. To avoid the tokenization, take use of the property rstImporter.tokenize.

Each `Group` object is also mapped to a `SStructure` object. The tree-like structure given by `Group` objects and `Relation` objects related to `Group` or `Segment` objects is mapped to `SStructure` objects related via `SDominanceRelation` objects. Imagine a `Group` object 'grp1', containing another `Group` object 'grp2' and a `Segment` object 'seg1'. This will be mapped into a Salt model having three `SStructure` objects 'struct1' for 'grp1', 'struct2' for 'grp2' and 'struct3' for 'seg1'. Further, two `SDominanceRelation` objects 'dom1' and 'dom2' are created with 'struct1 -dom1-> struct2' and 'struct1 -dom2-> struct3'.

Since all `Group` and `Segment` objects are mapped to `SStructure` objects, we won't loose the information, what has been the source. Therefore, for the kind of the node 'group' or 'segment' a `SAnnotation` object is created and related to the `SStructure` object. The sName of this `SAnnotation` object is set to 'kind'. To change the sName, take use of the property rstImporter.nodeKindName.

Also for the type attribute of a `Group` or `Segment` object an `SAnnotation` object is created. Its sName is set to 'type'. To adopt the sName, take use of the property rstImporter.nodeTypeName

The name of a `Relation` object is mapped to a `SAnnotation` object having the sName 'name'. To avoid, that a bunch of `Relation` object get the same name, an artificial number is concatenated to the name (`SDominanceRelation.sName`='name'+ occurance). For instance there are two `Relation` objects haveing the name 'rel', than the first will get the sName 'rel1' and the second will get the sName 'rel2'.

The type of a `Relation` object (//relations/rel@type in rs3) is mapped to the `sType` of a created `SDominanceRelation` object.

Properties

The table Table 2.1, “properties to customize importer behaviour” contains an overview of all usable properties to customize the behaviour of this pepper module. The following section contains a close description to each single property and describes the resulting differences in the mapping to the salt model.

Table 2.1. properties to customize importer behaviour

Name of property	Type of property	optional/ mandatory	default value
rstImporter.tokenize	yes|no	optional	yes
rstImporter.nodeKindName	String	optional	--
rstImporter.nodeTypeName	String	optional	--
rstImporter.relationTypeName	String	optional	--
rstImporter.relationTypeName	String	optional	' ' (Blank)

rstImporter.tokenize

This parameter is an optional parameter and can be set to “yes” or “no”. If it is set to “yes”, the text being included in a segment will be tokenized. The tokens will be mapped to `SToken`-objects in Salt and attached to the `SDocumentGraph`-object. Further, an `STextualRelation` between a token and the text will be created and a dominance relation between the token and the segment. The default configuration of this parameter is true, if non tokenization is required, this parameter must explicitly set to false.

rstImporter.nodeKindName

Name of the property to specify the sName of the SAnnotattion to which the kind of a node (segment or group) is mapped.

rstImporter.nodeTypeName

Name of the property to specify the sName of the SAnnotation to which the type attribute of a node is mapped.

rstImporter.relationTypeName

Name of the property to specify the sName of the SAnnotation to which the name attribute of a relation is mapped to.

rstImporter.relationTypeName

A property to add a a separator like a blank between the text of segments, when it is concatenated to the primary text in STextualDS.For instance the segment text 'Is' of segment1 and the segment text 'this' of segment2 will be concatenated to an sText value 'is'SEPARATOR'this'.