HRC Language Reference

12 September 2003

This version:
take5.beta2: 12 September 2003
(Available as HTML, PDF, DocBook)
Previous versions:
take5.beta1: 30 March 2003
take5.alpha3: 1 March 2003
take5.alpha2: 30 January 2003

Author:
Igor Ruskih `<cail at nm.ru>`

Abstract

This reference defines syntax and semantic of HRC language, used in Colorer-take5 Library to represent and describe syntax and lexical structure of target programming language. This description is used by library to parse and colourise text in editors or other systems.

Table of Contents

1. Introduction

HRC is a script language, describing parse process of text files to produce syntax highlighting. It is based on XML language, and defines its own XML vocabulary and structure. HRC language was developed to achieve most flexible and efficient process of describing programming language structure.

Started nearly in year 1999, it was a simple XML-like structure, describing some common language construction. But later it has grown into the much more complex and powerful language, describing complex relations between different languages, syntax contexts.

2. Core Syntax

HRC language allows describing and storing syntax rules for numerous languages. All language descriptions are divided into two parts: *Informal* part, used to describe this language properties, language choose rules (`prototype`), and *Formal* part, which defines syntax and semantic of target parsed language (`type`). Prototypes are used to determine, which type to apply to the currently opened file, they define some internal application-dependent properties and other useful information about language.

Structure. Each HRC file contains declaration of one or more prototypes or one language type. Root XML content starts with `hrc` element, which contains all other definitions.

Element Name: `hrc`, type: `hrc`

Root of the HRC file XML structure.

Attribute: `version`, type: `xs:NMTOKEN`

Specifies version of HRC language. For example, 'take5' for Colorer-take5.

Content:

Element: `annotation`, type: `annotation`

Defines formal documentation for HRC language elements.

Element: `prototype`, type: `prototype`

Defines prototype of single target programming language.

Element: `type`, type: `type`

Language container, used to store all parser specific information.

Each HRC language object is defined using XML elements and attributes. You can find definition of HRC XML Syntax in Appendix E, *XML Schema for HRC Language*. Each element in HRC can be documented with XML Schema-like standard elements:

Element Name: `annotation`, type: `annotation`

Defines formal documentation for HRC language elements.

Content:

Element: `appinfo`, type: `appinfo`

Element: `documentation`, type: `documentation`

Element: `contributors`, type: `contributors`

Annotation object can be used anywhere in HRC context to document and describe any of the HRC elements.

2.1. File Types

Each language prototype requires definition of language name and description. These properties are used to determine language in context of other language definitions and in inter-languages linkage.

2.1.1. Prototypes

Element Name: `prototype`, type: `prototype`

Defines prototype of single target programming language. This prototype must have name, equals to real type, defined in linked resource.

Attribute: `name`, type: `xs:NCName`

Common internal name of this language type. Must be valid XML non-qualified name.

Attribute: `description`, type: `xs:string`

User description, used to represent language in target IDE.

Attribute: `group`, type: `xs:Name`

Group of languages, this language belongs to.

Attribute: `targetNamespace`, type: `xs:anyURI`

Applicable to the XML group of languages. Specifies namespace, this HRC file describing. Allows automatically linking and combining different XML languages in HRC.

Content:

Element: `annotation`, type: `annotation`

Defines formal documentation for HRC language elements.

Element: `location`, type: `location`

Points to the location of HRC file with this language description. Link is well formed URI address of requested HRC file. This location can be relative to the location of parent type, or absolute (with any URI schema). If URI schema is absent, 'file://' is assumed.

Element: `filename`, type: `filename`

Defines Regular Expression, used to identify programming language by its file name.

Element: `firstline`, type: `firstline`

Defines Regular Expression, used to identify programming language by its starting content.

Element: `parameters`, type: `parameters`

Custom parameters, used to specify additional properties of this language type. These can include different language resources (icons, templates and so on).

Each language must be chosen by library before starting syntax highlighting process. This is made with help of `firstline` and `filename` parameters. Each matched instance of one of these parameters adds some additional weight to total language weight. This value is taken by default, or could be changed explicitly with `weight` attribute of these elements. When total weights of all types are evaluated, first language with maximum weight is selected to assign to opened file.

Element Name: `filename`, type: `filename`

Defines Regular Expression, used to identify programming language by its file name. This can include file's extension or some more complex dependencies.

Attribute: `weight`, type: `xs:decimal`, default: `2`

This attribute defines weight, added to the total language weight, when choosing one from a list of available.

Element Name: `firstline`, type: `firstline`

Defines Regular Expression, used to identify programming language by its starting content. First line could be used, or some small part of text. This entry has less default weight against filename one.

Attribute: `weight`, type: `xs:decimal`, default: `1`

This attribute defines weight, added to the total language weight, when choosing one from a list of available.

If any of these two operators is used more than one time, each its matched instance adds specified weight to the total language weight.

2.1.2. Types

Each prototype defines its linkage with real file type, describing information, specific for the syntax parsing process. This information is stored in basic units, called `type`s.

Element Name: `type`, type: `type`

Language container, used to store all parser specific information. These defines are used by parser to analyze and colorize target text data.

Attribute: `name`, type: `xs:NCName`

HRC Language type name.

Attribute: `access`, type: `access`, default: `private`

Deprecated???

Content:

Element: `annotation`, type: `annotation`

Defines formal documentation for HRC language elements.

Element: `import`, type: `import`

Import statement.

Element: `region`, type: `region`

Definition of basic syntax region - text range with assigned syntax meaning.

Element: `entity`, type: `entity`

HRC Entity definition.

Element: `scheme`, type: `scheme`

HRC Scheme is basic unit, which represents some fixed set of lexemes, tokens and syntax regions (lexical context).

Element Name: `region`, type: `region`

Definition of basic syntax region - text range with assigned syntax meaning. Later, these regions could be mapped into required color information and displayed on screen.

Attribute: `name`, type: `xs:NCName`

HRC Region name.

Attribute: `parent`, type: `QName`

Region's parent reference. If region has parent, its properties could be inherited from this one. Also region inheritance creates tree structure of HRC Regions.

Attribute: `description`, type: `xs:string`

Optional description, used to represent region's purpose and to show it to user in convient and friendly way.

Element Name: `entity`, type: `entity`

HRC Entity definition. Entities are some form of macro-definitions, they lately could be used in regular expressions syntax to make them more simple. Each entity consists of Entity name and Entity content, which would be substituted into regular expression, when parser finds entity reference. Each entity could be referenced with %entityname; syntax.

Attribute: `name`, type: `xs:NCName`

HRC Entity name.

Attribute: `value`, type: `xs:string`

HRC Entity value, used to substitute entity in RE string.

2.2. Schemas

Element Name: `scheme`, type: `scheme`

HRC Scheme is basic unit, which represents some fixed set of lexemes, tokens and syntax regions (lexical context). Each time at any position in the text only one schema is active. And its content is applied to the current text position. When starting text parse process, scheme with name, equals to its type name is used.

Attribute: `name`, type: `xs:NCName`

HRC Scheme name.

Attribute: `access`, type: `access`, default: `private`

Deprecated???

Content:

Element: `annotation`, type: `annotation`
Defines formal documentation for HRC language elements.

Element: `regexp`, type: `regexp`
Regular Expression token.

Element: `block`, type: `block`
Context switch operator.

Element: `keywords`, type: `keywords`
List of tokens with equal properties.

Element: `inherit`, type: `inherit`
Scheme inheritance construction.

2.3. Namespaces

xxxxxx

3. Scheme syntax

3.1. Keyword lists

Element Name: `keywords`, type: `keywords`
List of tokens with equal properties. Keywords, symbols and so on... These lists are used to make processing of many tokens faster, when it isn't required to use RE to define syntax tokens.

Attribute: `ignorecase`, default: `yes`
Match this list of tokens with case sensitive or no.

Attribute: `region`, type: `QName`
Region, assigned to this list of tokens. Each token can define its custom region.

Attribute: `priority`, type: `priority`, default: `low`
Priority of any token could be normal (default) and low.

Attribute: `worddiv`, type: `REworddiv`
Class of characters, used to search words edges.

Content:

Element: `word`, type: `word`
Keyword tokens - use specified word edges.

Element: `symb`, type: `symb`
Symbol tokens - ignores specified word edges.

Element Name: `word`, type: `word`

Keyword tokens - use specified word edges.

Attribute: `name`, type: `xs:string`

Attribute: `region`, type: `QName`

A pair of type name and valid XML name.

Element Name: `symb`, type: `symb`

Symbol tokens - ignores specified word edges.

Attribute: `name`, type: `xs:string`

Attribute: `region`, type: `QName`

A pair of type name and valid XML name.

3.2. Regular Expressions

Element Name: `regexp`, type: `regexp`

Regular Expression token.

Attribute: `match`, type: `REstring`

RE syntax

Attribute: `priority`, type: `priority`, default: `normal`

Priority of any token could be normal (default) and low.

3.3. Blocked context switch

Element Name: `block`, type: `block`

Context switch operator. Used to switch currently used context into the specified one. Context is switched, if RE pattern, placed in 'start' attribute, is matches. Switched context is closed, then parser finds match of the 'end' RE.

Attribute: `start`, type: `REstring`

Regular Expression

Attribute: `end`, type: `REstring`

Regular Expression

Attribute: `scheme`, type: `QName`

A pair of type name and valid XML name.

Attribute: `priority`, type: `priority`, default: `normal`

Priority of any token could be normal (default) and low.

Attribute: `content-priority`, type: `priority`, default: `normal`
Priority of any token could be normal (default) and low.

Attribute: `region`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region00`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region01`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region02`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region03`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region04`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region05`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region06`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region07`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region08`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region09`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region0a`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region0b`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region0c`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region0d`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region0e`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region0f`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region10`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region11`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region12`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region13`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region14`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region15`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region16`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region17`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region18`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region19`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region1a`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region1b`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region1c`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region1d`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region1e`, type: `QName`
A pair of type name and valid XML name.

Attribute: `region1f`, type: `QName`
A pair of type name and valid XML name.

Content:

Element: `start`, type: `blockInner`
Alternative style of RE definition.

Element: `end`, type: `blockInner`
Alternative style of RE definition.

Element Name: `blockInner`, type: `blockInner`

Alternative style of RE definition. Could be used, when RE is very complex and it is easier to use character (or CDATA) sections to define it.

3.4. Scheme boundaries and priority

4. Inter-scheme links

4.1. Inheritance

Element Name: `inherit`, type: `inherit`

Scheme inheritance construction. If one scheme is inherited in another, then the latter scheme takes all the definitions from the former, as it was included directly in place of inherit operator. One scheme can't inherit another, if that scheme is already makes inheritance (even indirect) of the first one.

Attribute: `scheme`, type: `QName`

Inherited scheme name.

Content:

Element: `virtual`, type: `virtual`

Inheritance substitution element.

Element Name: `virtual`, type: `virtual`

Inheritance substitution element. While inheriting one scheme in another, it is possible to redefine inner inherited schemes with some others. This could be used to change inherited language behavior.

Attribute: `scheme`, type: `QName`

Redefined scheme.

Attribute: `subst-scheme`, type: `QName`

Scheme to use instead redefined one.

4.2. Schemes substitutions

Regular Expressions syntax

1. Introduction

All work of the Colorer library and HRC language is based on the regular expressions (RE) usage. They allows you to create universal syntax rules of highlighting in HRC.

Regular expressions consist of the set of characters. Some of these are simple, and some are special metacharacters. All metacharacters (escapes) are divided into three categories: first - zerolength (words boundaries and so on); second - class metacharacters (\w, \s .); and the third class is operators. RE operators could be applied to single character, to block, enwrapped in brackets or into other operators. You can use brackets to group any sequence of characters. Regular expressions in HRC Language are like Perl regexp in their base syntax. There are some differences in extended operators.

2. Syntax

All regexps must be in slashes /.../ After the end slash there could be parameters:

- i - ignore symbol case
- x - ignore direct spaces and crlf (for comfort)
- s - suppose, that regexp is single line - it means, than '.' class should include \r\n symbols.

Each symbol in RE is linearly compared with the target string. Everything, that doesn't looks like metacharacters, means simple character.

3. Metacharacters

Table A.1. Metacharacters

^	Match the beginning of the line
$	Match the end of the line
.	Match any character (except \r\n)
[...]	Match characters in set
[^...]	Match characters not in set. Here all the operators are disabled, but you can use other metacharacters, and range operator: a-z means all chars from first to second (a - z)
\#	Next symbol '#' after slash (except a-z and 1-9)
\b	Start of word
\B	End of word
\xNN	NN - ASCII char (hex)
\n	0x10 (lf)

\r	0x13 (cr)
\t	0x09 (tab)
\s	tab/space/cr/lf
\S	Non-space
\w	Word symbol (chars, digits, _)
\W	Non-word symbol
\d	Digit
\D	Non-Digit
\u	Uppercase symbol
\l	Lowercase symbol

4. Extended metacharacter

These metacharacters are incompatible with Perl

Table A.2. Extended Metacharacters

\c	means 'non-word before'
\N	Link inside of regexp to one of its brackets. N - needed brackets pair. This operator works only with non-operator symbols in a bracket.

And these could be disabled during compilaton as highlight-dependent

Table A.3. Extended Metacharacters

~	Matches for the start of parent scheme (end of start start).
\m	Change start of regexp
\M	Change end of regexp
\yN	Link to the external regexp (in End to the Start param). N - requeried brackets pair.

5. Operators

Operators couldn't be used without some preceding character sequence. Each operator must be applied to the appropriate character, metacharacter, or block of their combina-

tion (enclosed with brackets).

Table A.4. Operators

`( )`	Group and remember characters to form one pattern.
`|`	Match previous or next pattern.
`*`	Match previous pattern 0 or more times.
`+`	Match previous pattern 1 or more times.
`?`	Match previous pattern 0 or 1 times.
`{n}`	Repeat n times.
`{n,}`	Repeat n or more times.
`{n,m}`	Repeat from n to m times.

If you'll add `?` after operator, it becomes nongreedy. For example `*` operator becomes nongreedy if placing `*?` Greedy operator tries to take as much in string, as it can. Non-greedy takes by minimum.

6. Extended operators

Table A.5. Extended Operators

`?#N`	Look-behind. N - symbol count.
`?~N`	Inverted Look-behind.
`?=`	Look-ahead.
`?!`	Inverted Look-ahead.

Note, that two last operators exist in Perl - in form of `(?=foobar)`. But colorer uses syntax `(foobar)?=`

7. Examples

Example A.1. RE examples

`/foobar/` will match "foobar", "foobar barfoo"

`/ FOO bar /ix`	will match "foobar" "FOOBAR" "foobar and two other foos"
`/(foo)?bar/`	will match "foobar", "bar"
`/^foobar$/`	will match _only_ with "foobar"
`/([\d\.])+/`	will match any number
`/(foo|bar)+/`	will match "foofoofoobarfoobar", "bar"
`/f[obar]+r/`	will match "foobar", "for", "far"

HRC Coding Recommendations

#######, ### ##### ####### ### #### ## ####### ########## ########## ####. ### ### ###### ###### ##### ## ####, ###### ##### ########## ## ##### #########. # ###### ### ### #### ####### #####. ##### #######, # ######## ####### #### ###### ##### ##### ###############, ####### ######## # ######## ## ########## #######. ## ## ##### ######### # ############ ######### # ########## ########.

#####, ###### ### ### ######### ############. ########## ##### #### ###### ### def, # ######, ##### ##### ##### ###### ## ####### ###### #####. ####### ######## ######## `scheme='c:StringCore'` ### ###### `scheme='StringCore'`.

changes:

########, `<import>` # ###### #####.

Format of catalog.xml file

Catalog of all Colorer Library resources is a convenient way to unify creation and management of all the Colorer features. This catalog is stored in `catalog.xml` file, and mapped into the ParserFactory class. Catalog supports storing of all installed HRC modules, management of error logging and listing of available HRD sets.

Element Name: `catalog`, type: `catalog`

Describes all available Colorer Library resources.

Content:

Element: `hrc-sets`, type: `hrc-sets`

Lists all installed root locations of HRC codes.

Element: `hrd-sets`, type: `hrd-sets`
Lists all available HRD sets.

Element Name: `hrc-sets`, type: `hrc-sets`

Lists all installed root locations of HRC codes. These locations are loaded when HRC bases are created.

Attribute: `log-location`, type: `xs:string`
Path to the default library log file. If missed, there is no logging.

Content:

Element: `location`, type: `location`
Single resource location.

Element Name: `hrd-sets`, type: `hrd-sets`

Lists all available HRD sets. Each HRD Entry describes single color scheme, used to represent colored text. Note, that one Entry

Content:

Element: `hrd`, type: `hrd-entry`
Describes one HRD properties set.

Element Name: `hrd-entry`, type: `hrd-entry`

Describes one HRD properties set.

Attribute: `class`, type: `xs:NMTOKEN`
HRD class. Currently available 'console', 'rgb' and 'text' classes.

Attribute: `name`, type: `xs:NMTOKEN`
Internal name of this set, used to referring from executable codes.

Attribute: `description`, type: `xs:string`
User-friendly description of this HRD set.

Content:

Element: `location`, type: `location`
Single resource location.

Element Name: `location`, type: `location`

Single resource location. Path can be relative to the catalog location, or absolute URI with or without URI schema specification.

Attribute: `link`, type: `xs:string`

```
<schema targetNamespace="http://colorer.sf.net/2003/catalog"
elementFormDefault="qualified"
  xmlns="http://www.w3.org/2001/XMLSchema"
```

```
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <element name="catalog" type="catalog"/>

  <complexType name="catalog">
    <sequence>
      <element name="hrc-sets" type="hrc-sets"/>
      <element name="hrd-sets" type="hrd-sets"/>
    </sequence>
  </complexType>

  <complexType name="hrc-sets">
    <sequence>
      <element name="location" type="location" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="log-location" type="xs:string">
    </attribute>
  </complexType>

  <complexType name="hrd-sets">
    <sequence>
      <element name="hrd" type="hrd-entry" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
  </complexType>

  <complexType name="hrd-entry">
    <sequence>
      <element name="location" type="location" maxOccurs="unbounded"/>
    </sequence>
    <attribute name="class" type="xs:NMTOKEN" use="required">
    </attribute>
    <attribute name="name" type="xs:NMTOKEN" use="required">
    </attribute>
    <attribute name="description" type="xs:string">
    </attribute>
  </complexType>

  <complexType name="location">
    <attribute name="link" type="xs:string" use="required"/>
  </complexType>
</schema>
```

Format of HRD color schemes

HRD storage is used to assign each HRC Region with some editor-specific properties. Commonly, these include color and style information. HRD file consists of the list of the entries, each of them describing one HRC Region.

Element Name: `hrd`, type: `hrd`

List of assigns between regions and their external properties. These properties commonly include text decoration parameters, such as color, style, font and so on... Global color layering model can be chosen by the target application, depending on its text presentation style, features and requirements. In general, all transparent colors must inherit color value

from its parent schema color. If the current schema is a top-level, default fore- and back-ground colors are used. Default Colors could be stored in HRD, using standard default region 'def:Text', or could be requested by application from the GUI environment.

Content:

Element: `assign`, type: `assign`

Single entry, describes region's properties.

Element Name: `assign`, type: `assign`

Single entry, describes region's properties. If entry is specified more than one time, then used the latest definition. This allows processing of several HRD files to complete color description of target HRC regions.

Attribute: `name`

Full qualified region name (a pair [type:name]). Note, that if region has no HRD properties associations, it inherits properties from its parent. If any of its ancestors has no assigned properties, region visualization must be skipped (it becomes fully transparent).

Attribute: `fore`, type: `xs:token`

Foreground color. If missed, transparent color assumed.

Attribute: `back`, type: `xs:token`

Background color. If missed, transparent color assumed.

Attribute: `style`, type: `xs:token`

Style bits (bold, italic, underline).

Attribute: `stext`, type: `xs:string`

Text prefix mapping (foreground).

Attribute: `etext`, type: `xs:string`

Text prefix mapping (background).

Attribute: `sback`, type: `xs:string`

Text Suffix mapping (foreground).

Attribute: `eback`, type: `xs:string`

Text Suffix mapping (background).

It is possible to maintain different HRD settings for different languages, or to compile them into one single HRD file. The former allows you to distribute recommended settings with each language, and the latter - to unify modifying and storing changed HRD settings with provided UI.

```
<schema targetNamespace="http://colorer.sf.net/2003/hrd"
elementFormDefault="qualified"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <element name="hrd" type="hrd"/>

  <complexType name="hrd">
```

```
    <sequence minOccurs="0" maxOccurs="unbounded">
      <element name="assign" type="assign"/>
    </sequence>
  </complexType>

  <complexType name="assign">
    <attribute name="name" use="required">
      <simpleType>
        <restriction base="xs:string">
          <pattern value="\i\c*\:\i\c*"/>
        </restriction>
      </simpleType>
    </attribute>
    <attribute name="fore" type="xs:token">
    </attribute>
    <attribute name="back" type="xs:token">
    </attribute>
    <attribute name="style" type="xs:token">
    </attribute>
    <attribute name="stext" type="xs:string">
    </attribute>
    <attribute name="etext" type="xs:string">
    </attribute>
    <attribute name="sback" type="xs:string">
    </attribute>
    <attribute name="eback" type="xs:string">
    </attribute>
  </complexType>
</schema>
```

XML Schema for HRC Language

This XML Schema instance was automatically generated from original hrc.xsd source, available at http://colorer.sf.net/2003/hrc.xsd. All comments and documentation tags were stripped to achieve more compact format. To use this schema in other, than informational purposes, use up-to-date version, available on the link above.

```
<schema targetNamespace="http://colorer.sf.net/2003/hrc"
elementFormDefault="qualified"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <simpleType name="REstring">
    <restriction base="xs:string">
      <whiteSpace value="collapse"/>
      <pattern value="/.*/[ix]*"/>
    </restriction>
  </simpleType>

  <simpleType name="REworddiv">
    <restriction base="xs:string">
      <whiteSpace value="collapse"/>
      <pattern value="\[.*\]|%.*;"/>
    </restriction>
  </simpleType>

  <simpleType name="REstring-or-null">
    <union memberTypes="REstring">
      <simpleType>
```

```
        <restriction base="xs:string">
          <enumeration value=""/>
        </restriction>
      </simpleType>
    </union>
  </simpleType>

  <simpleType name="QName">
    <restriction base="xs:string"/>
  </simpleType>

  <attributeGroup name="regionX">
    <attribute name="region" type="QName"/>
    <attribute name="region0" type="QName"/>
    <attribute name="region1" type="QName"/>
    <attribute name="region2" type="QName"/>
    <attribute name="region3" type="QName"/>
    <attribute name="region4" type="QName"/>
    <attribute name="region5" type="QName"/>
    <attribute name="region6" type="QName"/>
    <attribute name="region7" type="QName"/>
    <attribute name="region8" type="QName"/>
    <attribute name="region9" type="QName"/>
    <attribute name="regiona" type="QName"/>
    <attribute name="regionb" type="QName"/>
    <attribute name="regionc" type="QName"/>
    <attribute name="regiond" type="QName"/>
    <attribute name="regione" type="QName"/>
    <attribute name="regionf" type="QName"/>
  </attributeGroup>

  <element name="hrc" type="hrc"/>

  <complexType name="hrc">
    <sequence>
      <element name="annotation" type="annotation" minOccurs="0"/>
      <element name="prototype" type="prototype" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="type" type="type" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
    <attribute name="version" type="xs:NMTOKEN" use="required">
    </attribute>
  </complexType>

  <complexType name="annotation">
    <choice minOccurs="0" maxOccurs="unbounded">
      <element name="appinfo">
        <complexType mixed="true">
          <sequence minOccurs="0" maxOccurs="unbounded">
            <any namespace="##other" processContents="lax"/>
          </sequence>
        </complexType>
      </element>
      <element name="documentation">
        <complexType mixed="true">
          <sequence minOccurs="0" maxOccurs="unbounded">
            <any namespace="##other" processContents="skip"/>
          </sequence>
        </complexType>
      </element>
      <element name="contributors">
        <complexType mixed="true">
          <sequence minOccurs="0" maxOccurs="unbounded">
            <any namespace="##other" processContents="lax"/>
```

```
          </sequence>
        </complexType>
      </element>
    </choice>
  </complexType>

  <complexType name="prototype">
    <sequence>
      <element name="annotation" type="annotation" minOccurs="0"/>
      <element name="location" minOccurs="0">
        <complexType>
          <attribute name="link" type="xs:anyURI" use="required"/>
        </complexType>
      </element>
      <element name="filename" type="filename" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="firstline" type="firstline" minOccurs="0"
maxOccurs="unbounded"/>
      <element name="parameters" minOccurs="0">
        <complexType>
          <sequence minOccurs="0" maxOccurs="unbounded">
            <element name="param">
              <complexType>
                <attribute name="name" type="xs:string"
use="required"/>
                <attribute name="value" type="xs:string"
use="required"/>
              </complexType>
            </element>
          </sequence>
        </complexType>
      </element>
    </sequence>
    <attribute name="name" type="xs:NCName" use="required">
    </attribute>
    <attribute name="description" type="xs:string" use="required">
    </attribute>
    <attribute name="group" type="xs:Name">
    </attribute>
    <attribute name="targetNamespace" type="xs:anyURI">
    </attribute>
  </complexType>

  <complexType name="filename">
    <simpleContent>
      <extension base="REstring">
        <attribute name="weight" type="xs:decimal" default="2">
        </attribute>
      </extension>
    </simpleContent>
  </complexType>

  <complexType name="firstline">
    <simpleContent>
      <extension base="REstring">
        <attribute name="weight" type="xs:decimal" default="1">
        </attribute>
      </extension>
    </simpleContent>
  </complexType>

  <complexType name="type">
    <choice minOccurs="0" maxOccurs="unbounded">
      <element name="annotation" type="annotation"/>
      <element name="import" type="import"/>
```

```
    <element name="region" type="region"/>
    <element name="entity" type="entity"/>
    <element name="scheme" type="scheme"/>
  </choice>
  <attribute name="name" type="xs:NCName" use="required">
  </attribute>
  <attribute name="access" type="access" default="private"/>
</complexType>

<simpleType name="access">
  <restriction base="xs:string">
    <enumeration value="public"/>
    <enumeration value="private"/>
  </restriction>
</simpleType>

<complexType name="scheme">
  <sequence>
    <element name="annotation" type="annotation" minOccurs="0"/>
    <choice minOccurs="0" maxOccurs="unbounded">
      <element name="regexp" type="regexp"/>
      <element name="block" type="block"/>
      <element name="keywords" type="keywords"/>
      <element name="inherit" type="inherit"/>
    </choice>
  </sequence>
  <attribute name="name" type="xs:NCName" use="required">
  </attribute>
  <attribute name="access" type="access" default="private"/>
</complexType>

<complexType name="import">
  <attribute name="type" type="xs:NCName" use="required"/>
</complexType>

<complexType name="entity">
  <attribute name="name" type="xs:NCName" use="required">
  </attribute>
  <attribute name="value" type="xs:string" use="required">
  </attribute>
</complexType>

<complexType name="region">
  <attribute name="name" type="xs:NCName" use="required">
  </attribute>
  <attribute name="parent" type="QName">
  </attribute>
  <attribute name="description" type="xs:string">
  </attribute>
</complexType>

<complexType name="regexp">
  <simpleContent>
    <extension base="REstring-or-null">
      <attribute name="match" type="REstring">
      </attribute>
      <attribute name="priority" type="priority" default="normal"/>
      <attributeGroup ref="regionX"/>
    </extension>
  </simpleContent>
</complexType>

<simpleType name="priority">
  <restriction base="xs:string">
    <enumeration value="low"/>
```

```
      <enumeration value="normal"/>
    </restriction>
  </simpleType>

  <complexType name="block">
    <sequence minOccurs="0">
      <element name="start" type="blockInner"/>
      <element name="end" type="blockInner"/>
    </sequence>
    <attribute name="start" type="REstring"/>
    <attribute name="end" type="REstring"/>
    <attribute name="scheme" type="QName" use="required"/>
    <attribute name="priority" type="priority" default="normal"/>
    <attribute name="content-priority" type="priority"
default="normal"/>
    <attribute name="region" type="QName"/>
    <attribute name="region00" type="QName"/>
    <attribute name="region01" type="QName"/>
    <attribute name="region02" type="QName"/>
    <attribute name="region03" type="QName"/>
    <attribute name="region04" type="QName"/>
    <attribute name="region05" type="QName"/>
    <attribute name="region06" type="QName"/>
    <attribute name="region07" type="QName"/>
    <attribute name="region08" type="QName"/>
    <attribute name="region09" type="QName"/>
    <attribute name="region0a" type="QName"/>
    <attribute name="region0b" type="QName"/>
    <attribute name="region0c" type="QName"/>
    <attribute name="region0d" type="QName"/>
    <attribute name="region0e" type="QName"/>
    <attribute name="region0f" type="QName"/>
    <attribute name="region10" type="QName"/>
    <attribute name="region11" type="QName"/>
    <attribute name="region12" type="QName"/>
    <attribute name="region13" type="QName"/>
    <attribute name="region14" type="QName"/>
    <attribute name="region15" type="QName"/>
    <attribute name="region16" type="QName"/>
    <attribute name="region17" type="QName"/>
    <attribute name="region18" type="QName"/>
    <attribute name="region19" type="QName"/>
    <attribute name="region1a" type="QName"/>
    <attribute name="region1b" type="QName"/>
    <attribute name="region1c" type="QName"/>
    <attribute name="region1d" type="QName"/>
    <attribute name="region1e" type="QName"/>
    <attribute name="region1f" type="QName"/>
  </complexType>

  <complexType name="blockInner">
    <simpleContent>
      <extension base="REstring">
        <attributeGroup ref="regionX"/>
      </extension>
    </simpleContent>
  </complexType>

  <complexType name="inherit">
    <sequence>
      <element name="virtual" type="virtual" minOccurs="0"
maxOccurs="unbounded"/>
    </sequence>
    <attribute name="scheme" type="QName" use="required">
    </attribute>
```

```
  </complexType>

  <complexType name="virtual">
    <attribute name="scheme" type="QName" use="required">
    </attribute>
    <attribute name="subst-scheme" type="QName" use="required">
    </attribute>
  </complexType>

  <complexType name="keywords">
    <choice minOccurs="0" maxOccurs="unbounded">
      <element name="word" type="word"/>
      <element name="symb" type="symb"/>
    </choice>
    <attribute name="ignorecase" default="yes">
      <simpleType>
        <restriction base="xs:string">
          <enumeration value="yes"/>
          <enumeration value="no"/>
        </restriction>
      </simpleType>
    </attribute>
    <attribute name="region" type="QName">
    </attribute>
    <attribute name="priority" type="priority" default="low"/>
    <attribute name="worddiv" type="REworddiv">
    </attribute>
  </complexType>

  <complexType name="symb">
    <attribute name="name" type="xs:string" use="required"/>
    <attribute name="region" type="QName"/>
  </complexType>

  <complexType name="word">
    <attribute name="name" type="xs:string" use="required"/>
    <attribute name="region" type="QName"/>
  </complexType>
</schema>
```

References

[XML 1.0] Tim Bray, Jean Paoli, and C. M. Sperberg-McQueen, Eve Maler, editors. *Extensible Markup Language (XML) 1.0 Second Edition*. W3C (World Wide Web Consortium), 2000.

[XSLT 1.0] James Clark, editor. *XSL Transformations (XSLT) 1.0*. W3C (World Wide Web Consortium), 1999.

[W3C XML Schema Structures] Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, editors. *XML Schema Part 1: Structures*. W3C (World Wide Web Consortium), 2001.

[W3C XML Schema Datatypes] Paul V. Biron, Ashok Malhotra, editors. *XML Schema Part 2: Datatypes*. W3C (World Wide Web Consortium), 2001.