

Datatypes

Datatypes play an important role in xReporter. For each parameter that is asked to the user, or for each value that is retrieved from a resultset, a datatype is required.

It is possible to create a catalog of datatypes. The datatypes defined in the catalog can then be reused across report definitions. It is also possible to define datatypes locally in the report definitions, or to create a new datatype based on an existing datatypes (sort of "extends" functionality).

A datatype is always based on one of the following built-in basetypes: string, long, bigdecimal, date, time or datetime. This basetype can then be extended with a prompt, a title, validation rules, formatting patterns, a default value and a search list.

The datatype catalog is an XML file. The name and location of this file can be freely chosen and must be configured in the config.xml.

A first datatype

Below is an example of a simple datatype catalog containing one datatype:

```
<datatypes>
  <datatype id="name">
    <prompt>prompt.name</prompt>
    <title>title.name</title>
    <type base="string"/>
  </datatype>
</datatypes>
```

The datatypes element is the document element of the datatype catalog. It contains a number of datatype elements. Each datatype element must have an id attribute, whose value must be a unique identification for that datatype. id's should always start with a letter, and can then contain letters, numbers, and the characters . (dot), - (dash) and _ (underscore).

The prompt element is used to specify the text shown to the user when it enters a value of this type. The prompt element does not really contain this text, instead it contains a resource bundle key. Such a key is used, together with the user's language, to lookup the text from a resource bundle. A resource bundle key can be chosen freely, but may not contain spaces, or the characters '=' and '!'.

The title element is similar to the prompt element, but specifies the text shown when outputting a value of this datatype. This will usually be used as column title in a table.

The type element specifies the basetype on which this datatype is based. The table below gives an overview of the possible values for the base attribute.

Overview of the basetypes

basetype	to be used for
string	text
long	whole numbers
bigdecimal	real numbers
date	dates (without time component)
time	time (without date component)
datetime	date and time

Formatting patterns

The value of the string-based datatypes is always displayed as-is. The values of numeric and time related datatypes however need to be formatted before it can be displayed. xReporter allows to specify specific formatting patterns according to the language and country of the user (also called the user's locale). If you don't do this, a default formatting will occur, which will also be locale-dependent.

Below is an example of how a formatting pattern can be specified.

```
<datatype id="sequence-number">
  <title>title.seqnr</title>
  <type base="long">
    <patterns>
      <pattern locale="nl-BE" value="nr 0000"/>
      <pattern locale="*-*" value="0000"/>
    </patterns>
  </type>
</datatype>
```

The special value `*-*` for the locale attribute means "all locales". This pattern will be used if no pattern is found for the locale of the user. It is not required to specify this pattern.

The syntax of the patterns is very similar to the ones used in other applications such as spreadsheets. The syntax for long and bigdecimal is described in [this document](#)¹, the syntax for date, time and datetime in [this document](#)².

The formatting patterns are not only used to display values, but are also used to recognize ("parse") the values entered by the user. For more on this see the section on validation.

Default value

In case the user has to enter a value, a default value can be specified. The default value is calculated using an expression. These expressions use xReporter's [expression language](#)³ which is also used for other purposes. Below is an example of a datatype with a default value.

```
<datatype id="SomeNumber">
  <default expression="Random()*100"/>
  <type base="long"/>
</datatype>
```

The function `Random()` returns a random value between 0 and 1. By multiplying with 100, a number between 0 and 100 is obtained. In expressions, all numbers are considered as decimal numbers. Because the basetype in this example is long, the decimal value will be automatically converted to a whole value.

Here are some more example, the explanation follows below.

```
1. <default expression="'hello'"/>
2. <default expression="Date(2002,12,25)"/>
3. <default expression="Now()"/>
4. <default expression="4.33"/>
```

-
1. <http://icu.sourceforge.net/apiref/icu4j/com/ibm/icu/text/DecimalFormat.html>
 2. <http://icu.sourceforge.net/apiref/icu4j/com/ibm/icu/text/SimpleDateFormat.html>
 3. [daisy:91-cd \(Expression Language\)](#)

Example 1 shows how a fixed string can be used as default value. Example 2 shows how the date 25 December 2002 can be defined as default value. Example 3 shows how the current point in time can be used as default. Example 4 shows how the decimal number 4.33 can be used as default.

Since the expression language is extensible with new functions (which can be implemented in Java), the flexibility is unlimited. If the datatype is used inside a report, it is also possible to use certain 'context' of the report. For example, to use the value of a parameter (from an interaction step) inside an expression, you can simply use the id of the parameter (as if it were a variable). Only parameters from previous interaction steps can be referred.

When specifying expressions, you should always make sure that the result type of the expression corresponds with the result type of the data type. For the basetypes long and bigdecimal the result of the expression has to be a numeric value. For date, time and datetime the result of the expression must be a point in time (date- or timecomponent will automatically be removed if not applicable).

Validation

When a user has entered a value, this value has to be validated. This happens in two steps. First the text entered by the user must be recognized as being a value of the basetype of the datatype. For example, if the basetype is long and the user entered "abc", then this will cause a validation error. This kind of validation is done automatically by xReporter. Next to that, a number of validation rules can be specified.

For the first part of the validation the same formatting patterns are used as previously discussed, or if not specified the locale-dependent defaults. For example, a decimal number in Belgium has to be specified as "5,3", while in the US this should be "5.3".

Below is an example using validation rules.

```
<datatype id="SomeOtherNumber">
  <type base="long">
    <validation>
      <minExclusive value="0"/>
      <maxInclusive value="100"/>
    </validation>
  </type>
</datatype>
```

The datatype above will only allow values from one up to and including hundred. The minExclusive and maxInclusive elements are two examples of validation rules. For long and bigdecimal, the following validation rules are available:

- minInclusive
- minExclusive
- maxInclusive
- maxExclusive

The purpose of these rules should be apparent from their name. For string, the following validation rules are available:

- length
- minLength
- maxLength
- pattern

The length rule checks that a string is exactly a certain number of characters long. The pattern rule checks that a string satisfies a regular expression. The format of the regular expressions is described in [this document](#)⁴. Below is an example datatype that only accepts one capital as input.

```
<datatype id="SomeLetter">
  <type base="string">
    <validation>
      <pattern value="[A-Z]" />
    </validation>
  </type>
</datatype>
```

For date, time and datetime there are currently no specific validation rules, aside from the rules set forth by `DateFormat.parse(String)`⁵. Should you wish for some reason to allow lenient parsing of dates (e.g., for backward-compatibility for existing report definitions), you may do so by setting an (optional) lenient attribute to 'true'.

```
<datatype id="SomeDate">
  <type base="date" lenient="true" />
  <title>Some Date</title>
  <prompt>Enter Some Date</prompt>
</datatype>
```

Finally there is one more validation rule that can be applied to all basetypes, namely expression. This validation rule uses xReporter's [expression language](#)⁶. To support validation there are three specific functions available in the expressions: `ValidationValue()`, `ValidationError(string)` and `ValidationOk()`. The result of the expression should always be either `ValidationError(string)` or `ValidationOk()`. The parameter of `ValidationError(string)` should be a resource bundle key representing the error message. `ValidationValue()` is a function that returns the value to be validated. For complex validation needs it is possible to create new functions for the expression language in Java.

Below is a simple example of validation using an expression, which checks if the entered value is anything but 50:

```
<datatype id="NumberNot50">
  <type base="long">
    <validation>
      <expression value="If(ValidationValue()=50,
        ValidationError("error.fifty-not-allowed"),ValidationOk())" /> </validation>
    </type>
  </datatype>
```

Searchlists

Searchlists allow the user to select a value from a list (instead of entering it). The searchlist is always based on values from a database table. Have a look at this example:

```
<datatype id="code">
  <type base="string" />
  <searchlist>
    <sql>
      <dialect types="default">
        <literal>select code, short_description, long_description from
```

4. <http://java.sun.com/j2se/1.4.1/docs/api/java/util/regex/Pattern.html>

5. [http://java.sun.com/j2se/1.4.2/docs/api/java/text/DateFormat.html#parse\(java.lang.String\)](http://java.sun.com/j2se/1.4.2/docs/api/java/text/DateFormat.html#parse(java.lang.String))

6. daisy:91-cd (Expression Language)

```

        codetabel</literal>
    </dialect>
</sql>
<output-fields value="code" short-description="short_description"
    long-description="long_description"/>
</searchlist>
</datatype>

```

The content of the sql element offers the same possibilities as in report definitions, we refer to the [report documentation](#)⁷ for more information on this. The SQL statement will be executed on the same datasource as the one on which the report (wherein the datatype is used) is executed.

The output-fields element specifies which values should be retrieved from the result set. Only the value attribute is mandatory, short-description and long-description are optional. The type of the field defined in the value attribute must be compatible with the basetype of the datatype. For example, if the basetype is date, then the field should also be a date-type. short-description and long-description should be strings.

Extending datatypes

It is possible to define a new datatype based on an existing datatype (this is called "extending" the datatype). This is useful if e.g. a datatype is needed that is the same as another one, but with another prompt.

Extending a datatype can be done both within the datatype catalog itself, and inside report definitions. This last option is described in the documentation on report definitions.

Below is an example using extending.

```

<datatypes>
  <datatype id="A">
    <type base="string">
      <validation>
        <maxLength value="5"/>
      </validation>
    </type>
  </datatype>

  <datatype id="B" extends="A">
    <searchlist>
      <!-- details omitted -->
    </searchlist>
  </datatype>
</datatypes>

```

In this example there are two datatypes, A and B, whereby B extends from A and adds a searchlist. Datatype B inherits all the properties from datatype A, thus datatype B will also accept maximum 5 characters.

A datatype can only extend from a datatype which occurs before it in the datatype catalog.

Next to adding properties, existing properties can also be changed. The following example shows how datatype B assigns a new default value.

```

<datatypes>
  <datatype id="A">
    <default expression="'tralala'"/>
    <type base="string"/>
  </datatype>

  <datatype id="B" extends="A">

```

7. daisy:86-cd (Reports)

```
    <default expression="troeloe" />
  </datatype>
</datatypes>
```

The redefining and adding of properties is always done based on elements which are a direct child of the datatype element. Thus if datatype B contains a new type element, it replaces the type element from datatype A. So it is not possible to, for example, inherit the validation rules from datatype A and add formatting patterns to that, since both are inside the type element.

Summary datatype syntax

If we combine all the elements for describing datatypes, the following structure is obtained (the order of the elements does not matter):

```
<datatype id="" extends=""? >
  <prompt>resourcebundle.key</prompt> ?
  <title>resourcebundle.key</title> ?
  <info>resourcebundle.key</info> ?
  <default expression="" /> ?
  <searchlist>
    <sql>
      <dialect .... /> +
    </sql>
    <output-fields value="" short-description=""? long-description=""? />
  </searchlist> ?
  <type base="string|long|bigdecimal|date|time|datetime" lenient="true|false"> <!-- lenient
attribute is only valid on date, time, and datetime -->
    <patterns>
      <pattern locale="" value="" /> +
    </patterns> ?
    <validation>
      <!-- validationrules, depending on basetype -->
    </validation> ?
  </type> ?
</datatype>
```

If the datatype does not extend from another one, the type element is obligatory.