SeriesComparisonCheck

Checks the specified expression for the available variables and alters the flags for the time steps where the expression succeeds. For instance, a check can compare the water flow before and after a river junction. It is required that all referenced variables in the expression have the same size. Either all variables are regular with the same timestep, or all variables are non-equidistant with the same times. The comparison can be for different parameters at one location or per location of a locationSet, or for one parameter at two locations or at one location and the locations of a location set, or between multiple equal location sets.

Contents of check for seriesComparisonCheck

- **id**: identifier of the check.
- **variableDefinition**: embedded variable definition (see above).
- **expression**: A comparison between one or more variableIds (see examples below).
- **validatingVariableId**: One or more identifiers for variables for which the flags have to be modified.
- **outputFlag**: New flag value for time steps for which there is valid data and the expression holds. Either *doubtful* or *unreliable*.
- **outputMode**: When this option is set to *logs_only*, the flags will not be updated but the log events will be generated.
- **logLevel**: Log level for the log message that is logged if a time series does pass the check. Can be DEBUG, INFO, WARN, ERROR or FATAL. If level is error or fatal, then the module will stop running after logging the first log message. Fatal should never be used actually.
- **logEventCode**: Event code for the log message that is logged if a time series does pass the check. This event code has to contain a dot, e.g. "TimeSeries.Check", because the log message is only visible to the master controller if the event code contains a dot.
- **logMessage**: Log message that is logged if a time series does pass the check. Some more options are available than in the other checks:
- **onErrorResumeNext** When true, makes the secondary validation continue when an error logging is applied.

Tag	Replacement
%AMOUNT_CHANGED_FLAGS%	The number of flags that has been altered.
%CHECK_ID%	The id of the check that caused the flags to be altered.
%EXPRESSION%	The expression that caused the flags to be altered.
%HEADER%	The header names of the timeseries for which the flags were altered.
%LOCATION_ID%	The locationId where the alterations took place.
%LOCATION_NAME%	The name of the locations where the alterations took place.
%OUTPUT_FLAG%	The flag that has been set.
%PARAMETER_ID%	The parameterId where the alterations took place.
%PARAMETER_NAME%	The name of the parameter where the alterations took place.
%PERIOD%	The period in which flags were changed.
%NONE%	Hide the autogenerated locationId and parameterId in square brackets in the log message.
%VALUE%	The value of the last time that the flag for a timeseries has been updated.
`%INPUT_VALUE(<variableId>)%`	The input value of a specific variable

It is not possible to compare two different location sets both containing more than one location id, but the following comparisons can be configured:

- one location with a scalar
- all the locations in a location set with a scalar
- two different locations
- one location with all the locations in a location set
- two similar locationSets, containing exactly the same location ids

Configuration examples for seriesComparisonCheck

The expression is always a comparison. The comparison operator is within XML one of (.ne., .eq., .gt., .ge., .lt., .le.). Each variable has to be a single word without spaces. Mathematical symbols or functions like *e*, *pi* or *cos* cannot be used as variableId, but they will be interpreted mathematically. Note that in case one of the variables of the expression contains missing values for a timestep, the expression fails, and no flags will be altered for this timestep. Also manually edited flags will be left untouched. In 2012_01 and onwards, it is possible to use logical conditions in the expression. These can be in Fortran77 style *.and.* and *.or.* or with *&&* and ||.

Some mathematical functions worth mentioning are the following (these must be in lowercase):

Function	Description
avg(x1, x2, x3, ...)	Average
min(x1, x2, x3, ...)	Minimum

max(x1, x2, x3, ...)	Maximum
abs(x)	Absolute value
round(x)	Rounded value
floor(x)	Floor
ceil(x)	ceiling
sin, cos, tan	Trigonometric
mod(x , y)	x % y Modulus
sqrt(x)	SquareRoot
sum(x, y,...)	Sum of multiple variables

The following sample configuration sets the flags to unreliable when the values are smaller than 10 or bigger than 1000.

```
<secondaryValidation xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.wldelft.nl/fews http://fews.wldelft.nl/schemas/version1.0
/secondaryValidation.xsd">
    <seriesComparisonCheck id="checkWithScalar">
        <variableDefinition>
            <variableId>H_obs_location1</variableId>
            <timeSeriesSet>
                <moduleInstanceId>SeriesComparisonCheck</moduleInstanceId>
                <valueType>scalar</valueType>
                <parameterId>H.obs</parameterId>
                <locationId>location1</locationId>
                <timeSeriesType>external historical</timeSeriesType>
                <timeStep unit="minute" multiplier="15"/>
                <relativeViewPeriod unit="day" start="-30" end="0"/>
                <readWriteMode>read only</readWriteMode>
            </timeSeriesSet>
        </variableDefinition>
        <expression>H_obs_location .lt. 10 .or. H_obs_location .gt. 1000</expression>
        <validatingVariableId>H_obs_location1</validatingVariableId>
        <outputFlag>unreliable</outputFlag>
        <logLevel>INFO</logLevel>
        <logEventCode>TimeSeries.Check</logEventCode>
        <logMessage>%AMOUNT_CHANGED_FLAGS% flags set to %OUTPUT_FLAG% by [%CHECK_ID%, %EXPRESSION%].<
/logMessage>
    </seriesComparisonCheck>
</secondaryValidation>
```

A more complex sample does a comparison for different parameters in similar location sets, it will mark values that were reliable or doubtful as unreliable, in this case first for location1 and then for location2, when the difference between them is bigger than three:

```
<secondaryValidation xmlns="http://www.wldelft.nl/fews" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.wldelft.nl/fews http://fews.wldelft.nl/schemas/version1.0/secondaryValidation.
xsd">

<!-- comparison of variables with similar location sets, different parameters, does comparison per location  -->
        <seriesComparisonCheck id="similarLocationSetSeriesComparisonCheck">
                <!-- referred to by locationset1 and locationset2-->
                <variableDefinition>
                        <variableId>H_obs1_location1</variableId>
                        <timeSeriesSet>
                                <moduleInstanceId>SeriesComparisonCheckTest</moduleInstanceId>
                                <valueType>scalar</valueType>
                                <parameterId>H.obs1</parameterId>
                                <locationId>location1</locationId>
                                <timeSeriesType>external historical</timeSeriesType>
                                <timeStep unit="minute" multiplier="15"/>
                                <relativeViewPeriod unit="day" start="-30" end="0"/>
                                <readWriteMode>read only</readWriteMode>
                        </timeSeriesSet>
                </variableDefinition>
```

```
<!-- referred to by locationset1 and locationset2-->
<variableDefinition>
        <variableId>H_obs1_location2</variableId>
        <timeSeriesSet>
                <moduleInstanceId>SeriesComparisonCheckTest</moduleInstanceId>
                <valueType>scalar</valueType>
                <parameterId>H.obs1</parameterId>
                <locationId>location2</locationId>
                <timeSeriesType>external historical</timeSeriesType>
                <timeStep unit="minute" multiplier="15"/>
                <relativeViewPeriod unit="day" start="-30" end="0"/>
                <readWriteMode>read only</readWriteMode>
        </timeSeriesSet>
</variableDefinition>
<!-- referred to by locationset1 and locationset2-->
<variableDefinition>
        <variableId>H_obs2_location1</variableId>
        <timeSeriesSet>
                <moduleInstanceId>SeriesComparisonCheckTest</moduleInstanceId>
                <valueType>scalar</valueType>
                <parameterId>H.obs2</parameterId>
                <locationId>location1</locationId>
                <timeSeriesType>external historical</timeSeriesType>
                <timeStep unit="minute" multiplier="15"/>
                <relativeViewPeriod unit="day" start="-30" end="0"/>
                <readWriteMode>read only</readWriteMode>
        </timeSeriesSet>
</variableDefinition>
<!-- referred to by locationset1 and locationset2-->
<variableDefinition>
        <variableId>H_obs2_location2</variableId>
        <timeSeriesSet>
                <moduleInstanceId>SeriesComparisonCheckTest</moduleInstanceId>
                <valueType>scalar</valueType>
                <parameterId>H.obs2</parameterId>
                <locationId>location2</locationId>
                <timeSeriesType>external historical</timeSeriesType>
                <timeStep unit="minute" multiplier="15"/>
                <relativeViewPeriod unit="day" start="-30" end="0"/>
                <readWriteMode>read only</readWriteMode>
        </timeSeriesSet>
</variableDefinition>

<variableDefinition>
        <variableId>locationSet1</variableId>
        <timeSeriesSet>
                <moduleInstanceId>SeriesComparisonCheckTest</moduleInstanceId>
                <valueType>scalar</valueType>
                <parameterId>H.obs</parameterId>
                <locationSetId>locationset1</locationSetId>
                <timeSeriesType>external historical</timeSeriesType>
                <timeStep unit="minute" multiplier="15"/>
                <relativeViewPeriod unit="day" start="-30" end="0"/>
                <readWriteMode>read only</readWriteMode>
        </timeSeriesSet>
</variableDefinition>

<variableDefinition>
        <variableId>locationSet2</variableId>
        <timeSeriesSet>
                <moduleInstanceId>SeriesComparisonCheckTest</moduleInstanceId>
                <valueType>scalar</valueType>
                <parameterId>H.obs</parameterId>
                <locationSetId>locationset2</locationSetId>
                <timeSeriesType>external historical</timeSeriesType>
                <timeStep unit="minute" multiplier="15"/>
                <relativeViewPeriod unit="day" start="-30" end="0"/>
                <readWriteMode>read only</readWriteMode>
        </timeSeriesSet>
</variableDefinition>
```

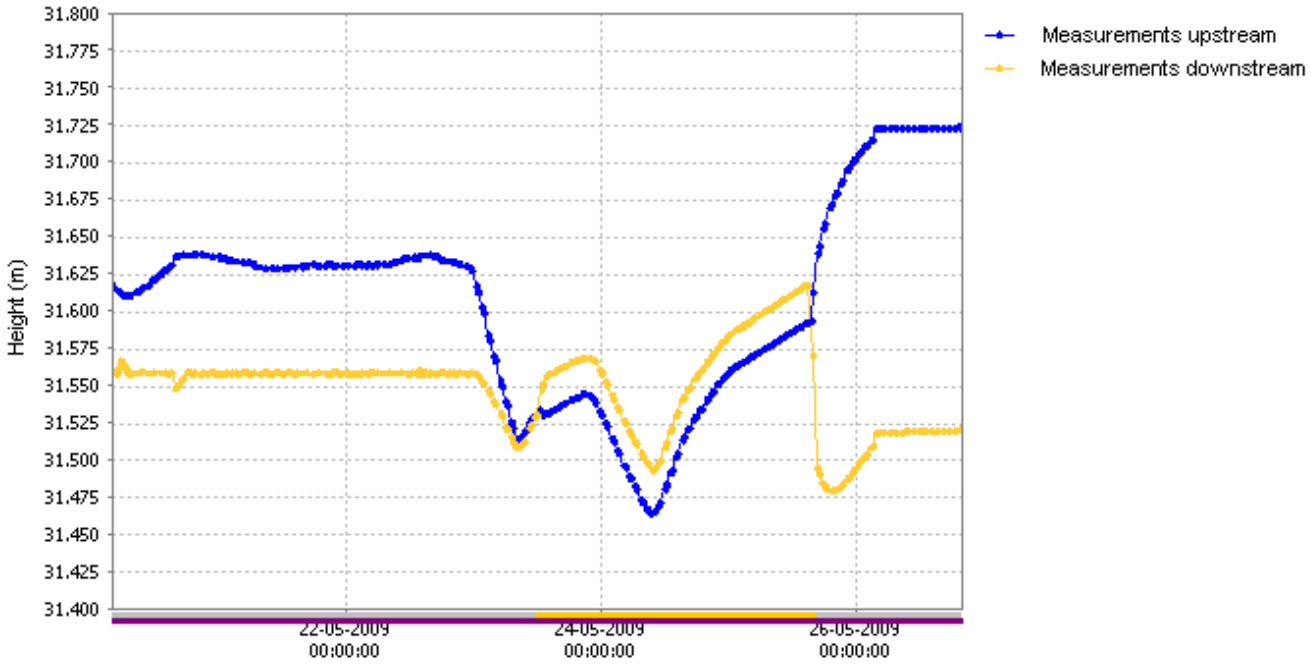
```
                <expression>abs(locationSet1 - locationSet2) .gt. 3</expression>
                <validatingVariableId>locationSet1</validatingVariableId>
                <validatingVariableId>locationSet2</validatingVariableId>
                <outputFlag>unreliable</outputFlag>
                <logLevel>INFO</logLevel>
                <logEventCode>TimeSeries.Check</logEventCode>
                <logMessage>%AMOUNT_CHANGED_FLAGS% flags set to %OUTPUT_FLAG% by %CHECK_ID%.</logMessage>
        </seriesComparisonCheck>
</secondaryValidation>
```

An even more smart configuration is where you use a locationRelation to the downstream and upstream locations. Then the configuration looks like:

```
<seriesComparisonCheck id="Check_if_waterlevel_upstream_higher_than_downstream">
    <variableDefinition>
        <variableId>H_us</variableId>
        <timeSeriesSet>
            <moduleInstanceId>SeriesComparisonCheckTest</moduleInstanceId>
            <valueType>scalar</valueType>
            <parameterId>H.obs</parameterId>
            <locationRelation>H_US_LOC_ID</locationRelation>
            <locationSetId>Locations_with_US_and_DS_H</locationSetId>
            <timeSeriesType>external historical</timeSeriesType>
            <timeStep unit="minute" multiplier="15"/>
            <relativeViewPeriod unit="day" start="-30" end="0"/>
            <readWriteMode>read only</readWriteMode>
        </timeSeriesSet>
    </variableDefinition>
    <variableDefinition>
        <variableId>H_ds</variableId>
        <timeSeriesSet>
            <moduleInstanceId>SeriesComparisonCheckTest</moduleInstanceId>
            <valueType>scalar</valueType>
            <parameterId>H.obs</parameterId>
            <locationRelation>H_DS_LOC_ID</locationRelation>
            <locationSetId>Locations_with_US_and_DS_H</locationSetId>
            <timeSeriesType>external historical</timeSeriesType>
            <timeStep unit="minute" multiplier="15"/>
            <relativeViewPeriod unit="day" start="-30" end="0"/>
            <readWriteMode>read only</readWriteMode>
        </timeSeriesSet>
    </variableDefinition>
    <expression>H_us .gt. H_ds</expression>
    <validatingVariableId>H_us</validatingVariableId>
    <validatingVariableId>H_ds</validatingVariableId>
    <outputFlag>unreliable</outputFlag>
    <logLevel>INFO</logLevel>
    <logEventCode>TimeSeries.Check</logEventCode>
    <logMessage>%AMOUNT_CHANGED_FLAGS% flags set to %OUTPUT_FLAG% by %CHECK_ID%.</logMessage>
</seriesComparisonCheck>
```

Sample screenshot

The sample screenshot below demonstrates the use of the seriesComparisonCheck. In this case it has been used to set flags to unreliable for timesteps where the waterlevel measurements upstream are below the measurements downstream. The different output flags have been displayed using different colors at the bottom of the screenshot. In this case the flags of the values above the yellow part have been set to unreliable, whereas the flags of the values above the purple line have remained the same.


Measurements upstream
Measurements downstream
Height (m)
31.800
31.775
31.750
31.725
31.700
31.675
31.650
31.625
31.600
31.575
31.550
31.525
31.500
31.475
31.450
31.425
31.400
22-05-2009 00:00:00
24-05-2009 00:00:00
26-05-2009 00:00:00