

Statistics Presenter - Documentation

XML Structure – note tags are case-sensitive!

(You can find the EUROMOD templates under “C:\ProgramData\euromod\UserSelectableTemplates”)

```
<TemplateInfo>
  Properties: see below
  <RequiredVariables>
    <RequiredVariable>
      Properties: see below
    </RequiredVariable>
  </RequiredVariables>
  <OptionalVariables>
    <OptionalVariable>
      Properties: see below
    </OptionalVariable>
  </OptionalVariables>
  <UserVariables>
    <UserVariable>
      Properties: see below
      <ComboItems>
        <ComboItem>
          Properties: see below
        </ComboItem>
      </ComboItems>
    </UserVariable>
  </UserVariables>
  <AdditionalCalculationLevels>
    </AdditionalCalculationLevel>
    Properties: see below
  </AdditionalCalculationLevel>
  </AdditionalCalculationLevels>
  <SDCDefinition>
    Properties: see below
    <MinObsAlternatives>
      <MinObsAlternative>
        Properties: see below
      </MinObsAlternative>
    </MinObsAlternatives>
  </SDCDefinition>
</TemplateInfo>

<Globals>
  <Filters>
    <Filter>
      Properties: see below
      <Parameters>
        <Parameter>
          Properties: see below
        </Parameter>
      </Parameters>
    </Filter>
  </Filters>
  <Actions>
    <Action>
      Properties: see below
      <Parameters>
        <Parameter>
          Properties: see below
        </Parameter>
      </Parameters>
      <Filter>
        Same as <Filter> above
      </Filter>
    </Action>
  </Actions>
</Globals>

<Pages>
  <Page>
    Properties: see below
    <Tables>
      <Table>
        Properties: see below
        <Actions>
          <Action>(see below)</Action>
        </Actions>
        <CellAction>(see below)</CellAction>
        <SDCDefinition>Properties: see below</SDCDefinition>
        <Columns>
          <Column>
            Properties: see below
            <CellAction>(see below)</CellAction>
            <SDCDefinition>Properties: see below</SDCDefinition>
          </Column>
        </Columns>
        <ReformColumns>
          <ReformColumn>
            Properties: see below
            <CellAction>(see below)</CellAction>
            <SDCDefinition>Properties: see below</SDCDefinition>
          </ReformColumn>
        </ReformColumns>
        <Rows>
          <Row>
            Properties: see below
            <CellAction>(see below)</CellAction>
            <SDCDefinition>Properties: see below</SDCDefinition>
          </Row>
        </Rows>
        <Cells>
          <Cell>
            Properties: see below
            <CellAction>(see below)</CellAction>
            <SDCDefinition>Properties: see below</SDCDefinition>
          </Cell>
        </Cells>
        <ReformCells>
          <ReformCell>
            Properties: see below
            <CellAction>(see below)</CellAction>
            <SDCDefinition>Properties: see below</SDCDefinition>
          </ReformCell>
        </ReformCells>
        <Graph>
          Properties: see below
          <Series>
            <Serie>
              Properties: see below
            </Serie>
          </Series>
        </Graph>
      </Table>
    </Tables>
  </Page>
</Pages>
```

Properties

TemplateInfo

- **Name:** text shown in template selection dialog
- **Title:** main heading shown in Statistic Presenter; allows for File-References and UserVariables (see [below](#))
- **Subtitle:** sub-heading shown in Statistic Presenter; allows for File-References and UserVariables (see [below](#))
- **Description:** text shown on clicking -button in template selection dialog; allows for HTML format
- **GeneralDescription:** text shown on clicking the **Info/General**-button in the Statistic Presenter; allows for HTML format
- **TemplateType:** Default / BaselineReform / Multi
- **Button:** text for the bottom tabs which in template allows for selecting the “file-package”, i.e.
TemplateType=Default: 1 “package” consists of 1 output-file
TemplateType=BaselineReform: 1 “package” consists of 1 base- and 1 to n reform-output-files
TemplateType=Multi: 1 “package” consists of n (independent) output-files
note that in user selectable templates “packages” are in fact only possible for TemplateType=Default (as users are not able to e.g. select several baseline-reforms), however system templates (PET, HHoT) use all three types
allows for File-References and UserVariables (see [below](#))
- **HideMainSelectorForSingleFilePackage:** if true, **Button** is hidden, if only one file-package is selected
- **MinFiles:** minimum number of files the user must select, applied as:
TemplateType=Default: output-files (default 1)
TemplateType=BaselineReform: reform-output-files (default 1)
TemplateType=Multi: output-files (default 1)
- **MaxFiles:** maximum number of files the user can select, applied as:
TemplateType=Default: output-files (default 50)
TemplateType=BaselineReform: reform-output-files (default 50)
TemplateType=Multi: output-files (default 50)
- **ExportDescriptionMode:** default: InSheets, defines how Descriptions are exported to Excel
SeparateSheet: all Descriptions are exported to a separate description-sheet (located after all other sheets)
InSheets: Page- and Table-Descriptions are inserted after the respective Page or Table, GeneralDescription is shown in a separate description-sheet (located after all other sheets)
No: Descriptions are not exported
- **RequiredVariables:** A list of the required variables (see [below](#))
- **OptionalVariables:** A list of the optional variables (see [below](#))
- **UserVariables:** A list of the user variables (see [below](#))
- **AdditionalCalculationLevels:** A list of additional calculation levels based on a grouping variable (e.g. Family in UKMOD templates) (see [below](#))
- **SDCDefinition:** The template global SDC definition (see [below](#))

RequiredVariable

- **Name:** name used in template to address Variable
- **ReadVar:** name of Variable in output-file
- **Monetary:** denotes whether this variable should be treated as monetary (i.e. summed in groups), default: true

Note that only files which contain all RequiredVariables are shown in the output-file selection dialog.

4 RequiredVariables are compulsory: *idperson, idhh, weight, age* (refers to **<Name>** and needs to be lowercase)

OptionalVariable

- **Name:** name used in template to address Variable
- **ReadVar:** name of Variable in output-file
- **Monetary:** denotes whether this variable should be treated as monetary (i.e. summed in groups), default: true
- **DefaultValue:** number; default=0

UserVariable

- **Name:** name used in template to address the user-variable
- **Monetary:** denotes whether this variable should be treated as monetary (i.e. summed in groups), default: true
- **UserInputType:**
 - **VariableName:** if the input value matches a variable in the output file, that variable is read from output file
 - **Numeric:** user is expected to indicate a number
 - **Categorical_VariableName:** user is expected to select a variable of a list as described by `<ComboItems>`
 - **Categorical_Numeric:** user is expected to select a number of a list as described by `<ComboItems>`
 - **PageSelection:** this is a not yet implemented future option
- **Title:** text shown as title of the input-box of the user-variable in the definition dialog
- **Description:** text shown on clicking the `i`-button displayed alongside the input-box of the user-variable
- **DisplayDescription:** default = false
if user variable is used in captions (see [below](#)), determines if it will display the value or the description
- **DefaultValue:** to be used if user does not provide input; must be appropriate for `<UserInputType>`
- **ComboItems:**
 - **ComboItem-Name:** text shown in the selection list and used as the user-input if selected, unless ...
 - **ComboItem-Value:** is defined, which is then used instead
- **PackageKey:** *is not used in the XML-template, but by programmes providing user-input (e.g. PET); default: null, used to make user input file-package-specific, for this purpose it needs to indicate the GUID of the file-package (EM_NewStatistics.FilePackageContent.Key)*
- **ReformNumber:** *is not used in the XML-template, but by programmes providing user-input (e.g. PET); default: -1, used to make user input reform-specific, for this purpose it needs to indicate the number of the reform (1-based)*

AdditionalCalculationLevel

- **Name:** a unique identifier for this Calculation Level, to be used in Actions and CellActions
- **GroupingVar:** the variable that will be used to create the new Calculation Level

Page

- **Name:** uniquely identifies the page; if a Title is not available, this value is used instead
- **Title:** page heading shown in Statistic Presenter; allows for File-References and UserVariables (see [below](#))
- **Subtitle:** page sub-heading shown in Statistic Presenter; allows for File-References and UserVariables (see [below](#))
- **Html:** the contents will be displayed directly under the subtitle; allows for HTML format
- **Description:** text shown on clicking the `Info/Page`-button in the Statistic Presenter; allows for HTML format
- **Visible:** true/false; default=true; whether this page should be visible
- **Active:** whether this page will be calculated
- **Button:** specifies the display of the page tab; can contain either:
 - caption of the tab as direct text; allows for File-References and UserVariables (see [below](#))
 - or the Common visual fields (see [below](#))
- **PerReform:** true/false; default=false; only relevant with `<TemplateType>` BaselineReform; if true, it will generate a separate page (with all tables) for each reform file selected, and the reference columns of any tables will not be duplicated
- **Actions:** a set of actions that will only be executed on page preparation (see [below](#))
- **Filters:** a set of filters that will only be executed on page preparation (see [below](#))
- **Tables:** (see [below](#))

Table

- **Name:** uniquely identifies the table within the page; if a Title is not available, this value is used instead
- **Title:** table heading shown in Statistic Presenter; allows for File-References and UserVariables (see [below](#))
- **Subtitle:** table sub-heading shown in Statistic Presenter; allows for File-References and UserVariables (see [below](#))
- **Description:** text shown on clicking the `Info/Table`-button in the Statistic Presenter; allows for HTML format
- **StringFormat:** number format to be applied by default to the whole table (for syntax see C# ToString())

- **ColumnGrouping**: only relevant with **<TemplateTypes>** BaselineReform and Multi SystemFirst: **<Columns>** of a system are shown next to each other, followed by **<Columns>** of the next system, ... ColumnFirst: **<Columns>** with associated content can be shown next to each other, see **<Column>-<TiesWith>**
- **Active**: whether this table should be calculated
- **Actions**: a set of actions that will only be executed on table preparation (see [below](#))
- **Filters**: a set of filters that will only be executed on page preparation (see [below](#))
- **CellAction**: the default action for all cells in the table (see [below](#))
- **PerReform**: true/false; default=false; only relevant with **<TemplateType>** BaselineReform; if true, it will generate a separate table for each reform file, and the reference columns of this table will not be duplicated
- **Graph**: (see [below](#))
- **Rows**: (see [below](#))
- **Columns**: (see [below](#))
- **ReformColumns**: (see [below](#))
- **Cells**: (see [below](#))
- **ReformCells**: (see [below](#))
- **SDCDefinition**: (see [below](#))

Column / ReformColumn

- **Common table fields**: (see [below](#))
- **IsVisible**: true/false; default=true; whether this Column should be visible
- **HasSeparatorBefore**: whether the column should have a separator on the left side
- **HasSeparatorAfter**: whether the column should have a separator on the right side
- **TiesWith**: the Name of the column this column should be tied to; only applicable for **<ReformColumns>** when the **<Table><ColumnGrouping>** is set to ColumnFirst

Example:

```
<Columns>
  <Column><Name>colA</Name></Column>
  <Column><Name>colB</Name></Column>
  <Column><Name>colC</Name></Column>
</Columns>
<ReformColumns>
  <ReformColumn><Name>colX</Name><TiesWith>ColB</TiesWith></ReformColumn>
  <ReformColumn><Name>colY</Name><TiesWith>ColC</TiesWith></ReformColumn>
</ReformColumns>
```

Arrangement of columns: colA, colB, colX^{1st Reform}, colX^{2nd Reform}, colC, colY^{1st Reform}, colY^{2nd Reform}

If more **<Columns>** tie with colB, they would follow after colX (in the order as within **<ReformColumns>**).

There are two specifics, for a **<ReformColumn>**:

- The column is repeated for each reform-scenario.
- The values of any Variables are taken from the respective reform-file.

Row

- **Common table fields**: (see [below](#))
 - **IsVisible**: true/false; default=true; whether this Row should be visible
 - **HasSeparatorBefore**: whether the row should have a separator on the top side
 - **HasSeparatorAfter**: whether the row should have a separator on the bottom side
 - **ForEachDataRow**: true/false; default=false
- If true the **<Row>** is duplicated for each data-row, i.e. for each person if **<CalculationLevel>** is set to Individual, or household if **<CalculationLevel>** is set to Household.¹
- Note that the **<Row>** is entirely replaced, that means the content (**<CalculationType>**, etc.) cannot be defined on

¹ Technically this is implemented by replacing the **<Row>** by appropriately many **<Rows>**, each equipped with a **<Filter>** "idperson=x" respectively "idhh=x". Note that this influences the **<Filter>** inheritance: any **<Filter>** defined on **<Table>** or **<Column>** level is overwritten, while **<Filters>** on **<Cell>** level do not make sense (i.e. generate arbitrary results).

<Row> level, but this can be accomplished on <Table> and/or <Column> level (as usual <Action> inheritance rules stay valid).

Also note that the original <Filter> of the <Row> can be used to restrict the new <Rows> to those persons or households fulfilling the condition. This needs to be done on <Row> level, see footnote 1.

Further note that this option can obviously not be used for <TemplateType> Multi (different data in each file).

This feature is required for the HHoT Budget Constraints table.

- **ForEachValueOf:** name of a Variable; default: empty.

If set, the <Row> is duplicated for each value of the respective Variable. For example,

<ForEachValueOf>deciles_eqDispy</ForEachValueOf> will most likely generate 10 <Rows>.

The “rules” as described for **ForEachDataRow** above also apply for this property (as the implementation is similar), except that it can be used for all <TemplateTypes>. For Multi values are assessed based on each dataset. For BaselineReform the values of the baseline are relevant.²

Further note that the **Name** may contain the placeholder [Value], which is replaced by the respective value. By default, i.e. if the placeholder is not used, **Name** is adapted to original name + [Value].

- **ForEachValueDescriptions:** used only in conjunction with ForEachValueOf; it contains a list of <ForEachValueDescription> nodes (each with a <Value> and <Description>) that is used to replace any matching [Value] with its Description.

Example for deciles one could have:

```
<ForEachValueDescriptions>
  <ForEachValueDescription><Value>1</Name><Description>one</Description></ForEachValueDescription>
  <ForEachValueDescription><Value>2</Name><Description>two</Description></ForEachValueDescription>
  <ForEachValueDescription><Value>3</Name><Description>three</Description></ForEachValueDescription>
</ForEachValueDescriptions>
```

Note that for values without a description, the value will be shown.

- **ForEachValueMaxCount:** used only in conjunction with ForEachValueOf; it sets the maximum number of values to be shown

Cell / ReformCell

Cells are defined to overwrite an the value of cell at a given column & row coordinate.

- **Common table fields:** (see [below](#))
- **ColName:** the Name of the Column (or ReformColumn) whose cell should be replaced
- **RowName:** the Name of the Row whose cell should be replaced

For a <ReformCell> the values of any Variables are taken from the respective reform-file.

Common table fields

- **StringFormat:** number format to be applied to the cell (or all included cells); for syntax google “c# double.ToString()”
- **CellAction:** the default action for this cell, or all included cells (see [below](#))
- **SDCDefinition:** (see [below](#))
- **Common visual fields:** (see [below](#))

² An option to allow users to decide whether baseline- or reform-values are considered is not difficult to implement and can be considered if ever necessary.

Common visual fields

- **Name:** uniquely identifies this element within the page; if a Title is not available, this value is used instead
- **Title:** the text that should be displayed; for Tables & Columns it allows for File-References and UserVariables (see [below](#))
- **Tooltip:** the tooltip text
- **Strong:** whether the text should be bold
- **TextAlign:** the alignment of the text
- **ForegroundColour:** the text colour
- **BackgroundColour:** the background colour

Action / CellAction

- **Name:** uniquely identifies this action within the template; allows for “merging” [<Action>](#)-definitions, (see [below](#))
- **CalculationLevel:** Individual or Household, this property can only be set on [<Table>](#)-level (i.e. if e.g. set on [<Column/Row/Cell>](#)-level, it is ignored)
- **CalculationType:**
Possible types: NA, CreateArithmetic, CreateEquivalized, CreateOECDScale, CreateEquivalenceScale, CalculateGini, CalculateMedian, CalculateS8020, CalculateMeanLogDeviation, CalculateAtkinson, CreateDeciles, CreateGroupValue, CreateHHValue, CalculateArithmetic, CalculateSumWeighted, CalculateWeightedAverage, CalculatePovertyGap, CreateFlag, CalculatePopulationCount, Empty, Info, Message
For a detailed description of those types see [below](#)
- **OutputVar:** Variable taking the result of the [<Action>](#)
Whether Variable is generated in Individual- or Household-data or as Saved Variable depends on [<CalculationType>](#) (see [below](#))
- **FormulaString:** (see [below](#))
- **Filter:** (see [below](#))
- **Reform:** true/false; default=true; only relevant if [<TemplateType>](#) is BaselineReform
If false all Variables used in the [<Action>](#) [<Parameters>](#) are taken from baseline-system
If [<Actions>](#) is defined in [<Globals>](#) section, calculations only take place for baseline-system
If used in [<Table>](#), false makes obviously only sense for [<ReformColumns>](#)
- **SaveResults:** true/false; default=true; if true (default!) it stores the outcome of the action as a SavedNumber for the following [<CalculationType>](#): CalculateArithmeticAction, CalculateGini, CalculateS8020, CalculatePovertyGap, CalculateMeanLogDeviation, CalculateAtkinson, CalculateMedian.
Specifically for CalculateArithmeticAction, CalculateGini, CalculateS8020, CalculateMeanLogDeviation, CalculateAtkinson, CalculateMedian
In this case, the second time you call the [<Action>](#) with the same [<OutputVar>](#), the value of the existing SavedNumber will be returned; this behaviour is designed to increase the efficiency of the library when the same result is required many times. You can change this behaviour by setting [<SaveResults>](#) to false.
- **BlendParameters:** true/false; default=true
If false no inheritance (see [below](#)) takes place. So if a [<Row>](#) has BlendParameters=false, the definitions of [<Column>](#) and [<Table>](#) are entirely ignored for this row.
- **Parameters:** (see [below](#))

Filter

- **Name:** uniquely identifies this Filter within the template; allows for “merging” [<Filter>](#)-definitions, (see [below](#))
- **Reform:** true/false; default=false
Only relevant if [<Filter>](#) is defined in [<Globals>](#) section and if [<TemplateType>](#) is BaselineReform
If false [<Filter>](#)-calculations only take place for baseline-system
- **FormulaString:** (see [below](#))
- **Parameters:** (see [below](#))

Parameter

- **Name:** a unique identifier for this parameter.
- **One of:**
 - **VarName:** the name of a variable; can refer to data, generated variables, or user input
 - **BoolValue:** a true/false value
 - **NumericValue:** a numeric value
- **Reform:** true/false; default=true; only relevant if **<TemplateType>** is BaselineReform
If false the Variable referred to by **<VarName>** is taken from baseline-system
If used in **<Table>**, false makes obviously only sense for **<ReformColumns>**

Graph

- **ShowTable:** true/false; default=true; whether the data table should be visible along with the graph
- **SeriesInRows:** true/false; default=true; whether the Series should be shown in rows or columns (i.e. transpose)
- **Series:** contains the descriptors for every series that needs to be displayed (see [below](#))
- **Title:** the title that will be shown for this graph
- **Round:** integer; default: -1; the number of decimal digits the tooltip should show when hovering over bars
- **Legend:** the properties of the graph legend (see [below](#))
- **AxisX:** the properties of the X axis (see [below](#))
- **AxisY:** the properties of the Y axis (see [below](#))

Legend

- **Title:** the title that will be shown for this legend
- **Docking:** the docking for the legend; can be "top", "bottom", "left", "right"
- **Visible:** true/false; default=true; whether the legend should be visible

AxisX / AxisY

- **Label:** the label that will be shown for this axis
- **ValuesFrom:** put a **<Serie>** Name, to take the axis values from there (used for the HHoT budget constraints graph)
- **StartFromZero:** true/false; default=true; whether the axis should always show the 0 point
- **Interval:** the interval of values to show for this axis

Serie

- **Name:** the name of the series (should match the name of a row/column of the table depending on **<Graph>** SeriesInRows)
- **Type:** the type of the series; can be StackedColumn, ColumnClustered, Point, Line
- **Size:** the size of the marker/line in px
- **Colour:** the colour of the marker/line/bar
- **MarkerStyle:** the style of the marker (only applicable if Type=Point or Line); can be: circle, cross, diamond, square, triangle, star4, star5, star6, star10 (*note: all stars will be shown the same in Excel, so avoid using them together*)
- **Visible:** true/false; default=true; whether the series should be visible

SDCDefinition (TemplateInfo)

- **MinObsDefault:** default: <0 (i.e. no minimum); template-wide SDC-observation-minimum
- **HideZeroObs:** default=true; if false, cells with zero observations are not hidden
- **MinObsAlternative:** (see [below](#))

see [below](#) for more comprehensive explanations

MinObsAlternative

- **Name:** name for the alternative, to be used in **<Table,...><SDCDefinition>**MinObsAlternative, see [below](#)

- **MinObs:** alternative SDC-observation-minimum
see [below](#) for more comprehensive explanations

SDCDefinition (Table, [Reform]Column, Row, [Reform]Cell)

- **MinObsAlternative:**
 - either a number specifying an alternative SDC-observation-minimum for the table-element
 - or name of an alternative minimum as defined in `<TemplateInfo><SDCDefinition><MinObsAlternative>`
 - **SecondaryGroups:** n elements with one of the following structures
 - `<NamedSecondaryGroup>Name of the group</NamedSecondaryGroup>`
 - `<AutoGroup/>`
where the latter is used for assigning all sub-elements (e.g. row-cells) to a group (the name is automatically generated)
 - **Suspend:** default=false; if true no SDC-checks are performed on the table-element
 - **SuspendSecondaryGroups:** default=false; if true any membership in `<SecondaryGroups>` is suspended for the table-element
 - **CountNonZeroObsOnly:** default = false; if true the SDC-observation-count omits observations with zero-value
 - **IgnoreActionFilter:** default = false; if true the SDC-observation-count ignores the `<Action>`'s filter
- see [below](#) for more comprehensive explanations

Usage of User Variables and File-References

Captions usually allow for using File References (see [below](#)) and User Variables (see [above](#)) in the form of e.g.

- `<Subtitle>`Results for [baseCountry]: [baseSys] vs [refSysList]`</Subtitle>` respectively
- `<Subtitle>`[@custom_variable]`</Subtitle>`

In addition `<UserVariables>` can be used as follows, if `<UserInputType>` is VariableName:

- as `<VarName>` of a `<Parameter>`, e.g. `<VarName>`custom_variable`</VarName>`
- as `<OutputVar>` of an `<Action>`, e.g. `<OutputVar>`[@custom_variable]`</OutputVar>`

Possible File / System References

- **[baseCountry]**: name of country: short name extracted from output-file name of baseline, translated by known-country-list
- **[baseSys] / [refSys]**: name of baseline / reform system, extracted from output-file name, e.g. cy_2018_std → cy_2018
- **[baseSysPretty] / [refSysPretty]**: refinement of [baseSys] / [refSys], e.g. Cyprus 2018
- **[baseTaxSysPretty] / [refTaxSysPretty]**: identifies the tax assumption in the base / reform output filename e.g. Constant quantities
- **[baseSysLabel] / [refSysLabel]**: allows passing a label programmatically through filePackages; used by InDepth
- **[baseFileName] / [refFileName]**: returns the output filename fo the baseline / reform
- **[baseDataset] / [refDataset]**: returns the dataset that was used in the baseline / reform
- **[baseRunLog] / [refRunLog]**: returns run information extracted from the log file
- **[baseRunLogExtended] / [refRunLogExtended]**: returns extra run information extracted from the log file
- **[baseRunWarnings] / [refRunWarnings]**: returns any related run warnings or errors found in the log file
- **[baseRunExtensionSwitches] / [refRunExtensionSwitches]**: returns the switch information for this run
- **[baseRunAddons] / [refRunAddons]**: returns the addon information for this run
- **[baseRunDuration] / [refRunDuration]**: returns the run duration
- **[baseRunStartTime] / [refRunStartTime]**: returns the run start time
- **[baseRunEndTime] / [refRunEndTime]**: returns the run end time
- **[refSysList]**: list of all reform systems separated by comma
- **[refFileList]**: list of all reform output filenames separated by comma

Calculation of Cell Value

The concrete value of a cell in a table of the Statistic Presenter is the result of an `<CellAction>`. Usually the definition of this `<CellAction>` is not restricted to the concerned `<Cell>` or `<ReformCell>`, but can reside at a higher level, e.g. a `<Column>` and thus be valid for all `<Cells>` of the `<Column>`. This is accomplished by a system of inheritance where for each `<Cell>` we merge the `<CellAction>` of the `<Table>`, `<Column>`, `<Row>` and `<Cell>` in that order so that any definition of a property on a lower level overwrites the definition of a higher level. More importantly, properties not defined on a higher level can be completed on a lower level. This reduces redundancy in the template and improves its flexibility and maintainability.

As a typical example: a cell's `<CellAction>` can be defined in the `<Column>`, which calculates a formula for e.g. calculating the average of some income, while the `<Row>` defines a `<Filter>`, which restricts the calculation to a certain population group.

The same mechanism works for the display of a cell, i.e. for the properties `<StringFormat>`, ``, `<Tooltip>`, `<TextAlign>`, `<ForegroundColor>` and `<BackgroundColor>`.

Action / CellAction

<Action> can exist only within <Globals>, <Page> or <Table>, whereas <CellAction> can exist only within <Table>, <Column>, <ReformColumn>, <Row> or <Cell>. An <Action> will save their value as a SavedVariable or create a new data variable, depending on the <CalculationType>.

- If the <CalculationType> starts with Create*, a Variable in data is created, i.e. one value per data-row.
- If the <CalculationType> starts with Calculate*, a single number is generated.

A <CellAction> can only have the second <CalculationType> as it needs to generate a value for a cell.

The name of the new Saved Variable or data Variable which is generated by an <Action> in the <Globals> section is defined by the Property <OutputVar>. As a rule of thumb, <OutputVar> is obligatory for Create* actions and optional for Calculate* actions. Note however, that not all <CalculationTypes> allow for an <OutputVar> (see [below](#) for details).

Data

The data an <Action> refers to depends on:

- its <CalculationLevel>, which can be either Individual or Household (or an AdditionalCalculationLevel defined in the TemplateInfo)
- on its <Filter>, which restricts the data according to the filter criteria (see [below](#))

Inheritance

As described [above](#), the <CellAction> for each cell within a <Table> is the result of combining definitions of different levels.

The priority (from lowest to highest) is:

- <Table>
- <Column> or <ReformColumn>
- <Row>
- <Cell> or <ReformCell>

Properties are in principle added in turn, but if a property (e.g. strong) is defined at both the <Table> and the <Row>, then the <Row> definition will win. Similarly any definition at the <Cell> or <ReformCell> will always win over other matching definitions. Typically you will see for example the <Table> defining the StringFormat, then one <Column> overwriting it to show percentages and maybe add a BackgroundColor, and one <Row> adding Strong.

The <Table> will definitely require <Column> and <Row> definitions, but unless you need to customize a specific cell, you don't need any <Cell> elements as these will be calculated based on the equivalent <Table>, <Column> and <Row> definitions.

The same applies to the <CellAction>, and included <Filter>. <Parameters> are in principle added, i.e. the final <Action> holds all <Parameters> of all levels, but <Parameters> with the same <Name> are overwritten in the same way as described for the cell visual properties above. If however, the <Action> <BlendParameters> Property is set to false, no inheritance takes place. This means that if a <Row> has BlendParameters=false, all its cells will ignore the <Table> and <Column> definitions.

CalculationType and FormulaString

The <Action> <CalculationType> specifies the type of calculation that needs to be performed, and whether it returns a single number or a number per data-row. One could categorise into <Actions> which make use of the Property <FormulaString> and <Actions> which do not.

- <Actions> not applying <FormulaString>, by and large perform special tasks and are described [below](#). They include CalculateGini, CalculateS8020, CalculatePovertyGap, CalculateMedian, CalculateMeanLogDeviation, CalculateAtkinson, CreateDeciles, CreateEquivalized, CreateGroupValue, CreateOECDScale, CreateEquivalenceScale and CreateFlag.
- <Actions> applying <FormulaString> include CreateArithmetic, CalculateArithmetic, CalculateSumWeighted, CalculateWeightedAverage and CalculatePopulationCount.

The <FormulaString> contains a formula allowing for

- arithmetic operations
- placeholders in the form of `SAVED_VAR[]`, `USR_VAR[]`, `DATA_VAR[]`, `TEMP_VAR[]`, `BASE_COL[]`, `REF_COL[]` and `REF_COL_PRE[]`.

Placeholders in FormulaString

All placeholders are further specified in the brackets and reference a variable, parameter, or (for column operations) column. For all apart from column placeholders, the value needs to be preceded by “@”.

- **SAVED_VAR[]**: The value within the brackets needs to refer to a Saved Variable name (e.g. **SAVED_VAR[@povLine]**)
A Saved Variable is a single number produced by another Calculate* **<Action>**.
All **<FormulaStrings>** can contain this placeholder.
- **USR_VAR[]**: The value within the brackets needs to refer to a User Variable (see [above](#)) (e.g. **USR_VAR[@myVar]**).
All **<CalculationTypes>** allowing for a **<FormulaString>**, except CalculateArithmetic, allow for the placeholder.
The interpretation depends on the User Variable type:
 - If the User Variable is numeric, the placeholder is replaced by this numeric value.
 - If the User Variable refers to a data Variable, then this is replaced with the equivalent **DATA_VAR[]**.
- **DATA_VAR[]**: The value within the brackets needs to refer to a variable, either from the data, or given by a user, or generated by another **<Action>** (e.g. **DATA_VAR[@yse]**). It can also refer to a named **<Parameter>** that in turn has a VarName that refers to a variable.
All **<CalculationTypes>** allowing for a **<FormulaString>** allow for the placeholder. Note however, that using a **DATA_VAR[]** in a CalculateArithmetic will result in picking the variable value for the first individual in the dataset.
- **TEMP_VAR[]**: The value within the brackets needs to refer to a **<Parameter>** Name and it allows for passing direct hard-coded values.
For example, one could have “3 * TEMP_VAR[@myVar]” in the **<FormulaString>**, and have a parameter:
<Parameter>
 <Name>myVar**</Name>**
 <NumericValue>4**</NumericValue>**
</Parameter>
The result of the calculation would be 12.
- **BASE_COL[]**: This value within the brackets needs to refer to a **<Column>** Name in the displayed table (e.g. **BASE_COL[Income]**). Note that the “@” is not required for column operations.
For example **BASE_COL[Employment income] + BASE_COL[Self-employment income]** generates, for each table row, the sum of these two columns. Note that only preceding columns can be used, i.e. one can not reference a column that is not defined yet (i.e. on the right of the current column). CalculateArithmetic is the only **<CalculationType>** allowing for the placeholder.
- **REF_COL[]** works exactly like **BASE_COL[]**, just for **<ReformColumn>**. Thus, obviously it only makes sense in templates with **<TemplateType>** BaselineReform.
As an example **REF_COL[Income] - BASE_COL[RefIncome]** generates, for each table row, the difference between the Income **<Column>** and the RefIncome **<ReformColumn>**.

Filter

<Filters> can be defined in three places:

- directly in the **<Globals>** section
- as element of an **<Action>**
- or as element of a **<CellAction>**

In fact the usage of **<Filters>** always takes place in an **<Action>** or **<CellAction>** to restrict the respective calculations to a certain group of data (as mentioned [above](#)). Filters follow the same inheritance rules as Actions, with the addition of one more level, the definitions in **<Globals>**, which have the lowest priority. If the applying **<Action>** defines a **<Filter>** with a **<Name>** that already exists, further specification is possible but not necessary.

FormulaString

The formula defined by the **<Filter>** **<FormulaString>** works in principle exactly like the formula for an **<Action>**, with one substantial difference. While the result of the **<Action>** formula always returns a number, the result of the **<Filter>** formula

needs to be a boolean expression. For example, the formula can have boolean expressions like, `DATA_VAR[@age]>18 && DATA_VAR[@age]<65` or `(DATA_VAR[@yem]+DATA_VAR[@yse])>0`.

Moreover, a `<Filter>` always works on observation level – put simply, if an observation (i.e. a household or individual) in data fulfils the condition stated in the `<FormulaString>`, the observation is part of the calculation, otherwise not.

For a detailed description of how placeholders work (see [above](#)).

Action CalculationTypes

General Terms used in the Explanations

All actions can have the following properties:

- **OutputVar**: required by all Create* actions and optional for Calculate* actions; defines the name of the output variable or SavedVariable that will be created accordingly
- **Observations**: refers to the observations of the data defined by `<CalculationLevel>`, restricted by `<Filter>`
- **Weighting**: if not indicated differently by named parameter *WeightVar*, the `<RequiredVariable>` *weight* is used for weighting
- **Named Parameters**: the name of the parameter is indicated by `<Name>`, while the respective value is indicated by either by `<VarName>`, `<NumericValue>` or `<BoolValue>`; all named parameters are optional unless otherwise specified

The unnamed parameter OutputVar is required for all Create* actions and optional for Calculate* variables. Currently the available calculation types are: CreateArithmetic, CreateEquivalized, CreateOECDScale, CreateEquivalenceScale, CreateDeciles, CreateGroupValue, CreateHHValue, CreateFlag, CalculateGini, CalculateMedian, CalculateS8020, CalculateMeanLogDeviation, CalculateAtkinson, CalculateArithmetic, CalculateSumWeighted, CalculateWeightedAverage, CalculatePovertyGap, CalculatePopulationCount, Empty, Info

CreateArithmetic

Generates the Variable indicated by `<OutputVar>` and fills it with the result of the `<FormulaString>` for each observation. The used Variables are addressed by the placeholder `DATA_VAR[]`.

Named Parameters

RecodeNegatives: default = false; if true, negative values are set to zero

Monetary: default = true; defines whether the generated variable will be monetary

Example

Generates a new Variable Earns, containing for each observation yem+yse

```
<Action>
  <CalculationType>CreateArithmetic</CalculationType>
  <OutputVar>Earns</OutputVar>
  <FormulaString><![CDATA[DATA_VAR[@yem] + DATA_VAR[@yse]]]></FormulaString>
</Action>
```

CreateEquivalized

Generates the Variable indicated by `<OutputVar>` and fills it with the household sum of “enumerator” divided by “numerator” (see parameters below for definition of enumerator and numerator)

The Variable is generated for both, household- as well as individual data (equal for each HH-member for the latter)

Note: if the `<OutputVar>` already exists, it does not run again!

Unnamed Parameter

The enumerator, i.e. the variable to equalise. **Attention!** This is the only case, where a parameter is not named and it only includes a VarName. This should be changed (for example to Name=IncomeVar) to be in line with all other CalculationTypes.

Named Parameters

EquivalenceScale: **required**; numerator, i.e. the equivalence scale to be used.

Example

```
<Action>
  <CalculationType>CreateEquivalized</CalculationType>
  <OutputVar>eqDispy</OutputVar>
  <Parameters>
    <Parameter>
      <VarName>ilsDispy</VarName>
    </Parameter>
  </Parameters>
```

```

    <Parameter>
      <Name>EquivalenceScale</Name>
      <VarName>OECDscale</VarName>
    </Parameter>
  </Parameters>
</Action>

```

CreateOECDScale

Generates the Variable indicated by <OutputVar> and fills it with the OECD Equivalence Scale.

The Variable is generated for both household and individual data (equal for each HH-member for the latter)

Note: if the <OutputVar> already exists, it does not run again!

Named Parameter

Oxford: default = false;

If false the scale is generated by adding 1 for the first person in HH, 0.5 for each other adult and 0.3 for each other child

If true the scale is generated by adding 1 for the first person in HH, 0.7 for each other adult and 0.5 for each other child

Children are defined as individuals younger than 14

Example

```

<Action>
  <Reform>>false</Reform>
  <CalculationType>CreateOECDScale</CalculationType>
  <OutputVar>OECDscale</OutputVar>
</Action>

```

CreateEquivalenceScale

Generates the Variable indicated by <OutputVar> and fills it with an Equivalence Scale, based on the given parameters.

The Variable is generated for both household and individual data (equal for each HH-member for the latter)

Note: if the <OutputVar> already exists, it does not run again!

Named Parameter

EquivalenceScaleBand: **required;** multiple; each parameter defines a band, using the name of a saved (e.g. global) filter as the VarName and the value to be applied as the NumericValue. For example:

Example

```

<Action>
  <CalculationType>CreateEquivalenceScale</CalculationType>
  <OutputVar>eqDispy</OutputVar>
  <Parameters>
    <Parameter>
      <Name>EquivalenceScaleBand</Name>
      <VarName>FilterEquivAdult</VarName>
      <NumericValue>0.33</NumericValue>
    </Parameter>
    <Parameter>
      <Name>EquivalenceScaleBand</Name>
      <VarName>FilterEquivChild</VarName>
      <NumericValue>0.2</NumericValue>
    </Parameter>
    <Parameter>
      <Name>EquivalenceFirstPerson</Name>
      <NumericValue>0.67</NumericValue>
    </Parameter>
  </Parameters>
</Action>

```

CreateDeciles

Generates the Variable indicated by <OutputVar> and fills it with a number for 1 to DecNo for each observation

The number is calculated by splitting observations into DecNo parts (taking weights into account), ordered by the IncomeVar.

Also generates DecNo-1 Saved Variables for taking the cut-offs, by appending “~dec~No” to the named **<OutputVar>**, e.g. deciles_eqDispy~dec~5 for the Median if **<OutputVar>**=deciles_eqDispy

Note: if the **<OutputVar>** already exists, it does not run again!

Named Parameters

WeightVar: described [above](#)

DecNo: default = 10, allows for producing other quantiles than deciles (e.g. quintiles with 5)

EqualDeciles: default = false; if true splitting observations into equal parts is prioritised over preventing observations with equal income in different deciles

IncomeVar: Variable used as base for splitting

GroupingVar: **!!! IMPORTANT !!!** (assuming individual data and decile-building for simplicity of explanation)

Using a GroupingVar does not – as one may assume – split data in 10 deciles, where each decile contains an equal³ number of **groups**. Instead it still splits data in 10 deciles, where each decile contains an equal⁴ number of **persons**.

The actual effects of using a GroupingVar are the following:

- it makes sure that persons within a group (i.e. usually HH-members) stay in the same decile
- if idhh is the GroupingVar, it writes the deciles not only to individual data, but also to household data

Consequently, the EUROMOD standard statistics use individual data with GroupingVar=idhh for creating deciles.

If one wants to create household quantiles one would use household data without a GroupingVar.

Actually building quantiles for another group (e.g. a taxunit) is currently not possible.

SdcQuantileObsOnly: default = false; if true the SDC observation count (see [below](#)) for the cut-offs counts the observations of respective quantile only

Example

```
<Action>
  <CalculationType>CreateDeciles</CalculationType>
  <OutputVar>deciles_eqDispy</OutputVar>
  <Parameters>
    <Parameter>
      <Name>IncomeVar</Name>
      <VarName>eqDispy</VarName>
    </Parameter>
    <Parameter>
      <Name>GroupingVar</Name>
      <VarName>idhh</VarName>
    </Parameter>
  </Parameters>
</Action>
```

CreateFlag

Requires a **<Filter>** to generate the Variable indicated by **<OutputVar>** and fill it with 1 if the **<Filter>** calculations yield true and zero if they yield false

The Variables are generated for the data indicated by the **<CalculationLevel>**

Named Parameters

Monetary: default = true; defines whether the new flag variable will be treated as monetary or not

Example

```
<Action>
  <CalculationType>CreateFlag</CalculationType>
  <OutputVar>isChild</OutputVar>
  <Filter>
    <FormulaString><![CDATA[DATA_VAR[@age] < 18]]></FormulaString>
  </Filter>
</Action>
```

³ leaving imprecise splitting aside for simplicity of explanation

⁴ again leaving imprecise splitting aside for simplicity of explanation

CreateGroupValue

This action performs different value transfer tasks between the individual level and the action CalculationLevel, based on the parameters.

Named Parameters

SumVar: default = <OutputVar>; Variable to sum or transfer from the individuals to the <CalculationLevel>

CopyToIndividuals: default = false; if true <OutputVar> is also generated in individual data and filled with the household sum of <SumVar>; obviously this requires <OutputVar> to be different from <SumVar> and generates the same number for each individual in household

CopyToGroup: if a value is defined and it is true, then just copy the value of the Head to the <CalculationLevel>; if the parameter is not defined, copy the head or sum or values depending on whether the variable is monetary or not.

Example

```
<Action>
  <CalculationType>CreateGroupValue</CalculationType>
  <CalculationLevel>Household</CalculationLevel>
  <OutputVar>nChildrenInHH</OutputVar>
  <Parameters>
    <Parameter>
      <Name>SumVar</Name>
      <VarName>isChild</VarName>
    </Parameter>
    <Parameter>
      <Name>CopyToIndividuals</Name>
      <BoolValue>True</BoolValue>
    </Parameter>
  </Parameters>
</Action>
```

CalculateGini

Calculates the Gini index, based on the parameters below.

Note: if the <OutputVar> already exists, it returns the stored result.

Named Parameters

WeightVar: described [above](#)

GiniVar: **required;** the (income) variable that will be used to calculate the Gini index

GroupingVar: **required;** the variable that defines the level of calculations

RecodeNegatives: default = false; if true, negative values are set to zero

ConcentrationIndexSortVar: default = *GiniVar*; allows changing the order of the observations

OutputVar: specifies the name of the SavedVariable to store the outcome

Example

```
<CellAction>
  <CalculationType>CalculateGini</CalculationType>
  <Parameters>
    <Parameter>
      <Name>GiniVar</Name>
      <VarName>eqOrigy</VarName>
    </Parameter>
    <Parameter>
      <Name>GroupingVar</Name>
      <VarName>idhh</VarName>
    </Parameter>
    <Parameter>
      <Name>RecodeNegatives</Name>
      <BoolValue>true</BoolValue>
    </Parameter>
  </Parameters>
</CellAction>
```

CalculateMedian

Calculates the median value, based on the parameters below.

Note: if the `<OutputVar>` already exists, it returns the stored result.

Named Parameters

WeightVar: described [above](#)

IncomeVar: **required**; the (income) variable that will be used to calculate the median

OutputVar: specifies the name of the SavedVariable to store the outcome

Example

```
<Action>
  <CalculationType>CalculateMedian</CalculationType>
  <Parameters>
    <Parameter>
      <Name>IncomeVar</Name>
      <VarName>ilsDispy</VarName>
    </Parameter>
  </Parameters>
</Action>
```

CalculateS8020

Calculates the division of the weighed average of the top and bottom deciles, based on the parameters below.

Allows for a boolean parameter "RecodeNegatives", that recodes negative values as zeroes.

Note: if the `<OutputVar>` already exists, it returns the stored result.

Named Parameters

WeightVar: described [above](#)

DecileVar: **required**; a Variable with values from 1 to 10 for each observation (usually created by CreateDeciles see [below](#))

S8020Var: **required**; the Variable upon which to create the index

STop: default = 80; the top percentile cutoff

SBottom: default = 20; the top percentile cutoff

RecodeNegatives: default = false; if true, negative values are set to zero

SdcQuantileObsOnly: default = false; if true the SDC observation count (see [below](#)) counts the observations of the upper or lower quintile (whichever count is lower) only

OutputVar: specifies the name of the SavedVariable to store the outcome

Example

```
<CellAction>
  <CalculationType>CalculateS8020</CalculationType>
  <Parameters>
    <Parameter>
      <Name>DecileVar</Name>
      <VarName>deciles_eqDispy</VarName>
    </Parameter>
    <Parameter>
      <Name>S8020Var</Name>
      <VarName>eqIncome</VarName>
    </Parameter>
  </Parameters>
</CellAction>
```

CalculateMeanLogDeviation

Calculates the mean log deviation, based on the parameters below.

Note: if the `<OutputVar>` already exists, it returns the stored result.

Named Parameters

WeightVar: described [above](#)

IncomeVar: **required**; the (income) variable that will be used to calculate the median

RecodeNegatives: default = false; if true, negative values are set to zero

Example

```
<Action>
  <CalculationType>CalculateMeanLogDeviation</CalculationType>
  <OutputVar>incomeMLD</OutputVar>
  <Parameters>
    <Parameter>
      <Name>IncomeVar</Name>
      <VarName>ilsDispy</VarName>
    </Parameter>
  </Parameters>
</Action>
```

CalculateAtkinson

Calculates the Atkinson index, based on the parameters below.

Note: if the **<OutputVar>** already exists, it returns the stored result.

Named Parameters

WeightVar: described [above](#)

IncomeVar: **required**; the (income) variable that will be used to calculate the median

RecodeNegatives: default = false; if true, negative values are set to zero

InequalityAversion: default = 0.25; the inequality aversion

OutputVar: specifies the name of the SavedVariable to store the outcome

Example

```
<CellAction>
  <CalculationType>CalculateAtkinson</CalculationType>
  <Parameters>
    <Parameter>
      <Name>IncomeVar</Name>
      <VarName>ilsDispy</VarName>
    </Parameter>
  </Parameters>
</CellAction>
```

CalculatePovertyGap

Calculates poverty gap as:

- if UseSwitchApproach is false: PovertyLine minus median of IncomeVar of population under PovertyLine, divided by PovertyLine
- else: $\frac{1}{N} \sum_{i=1}^H \frac{z - y_i}{z}$, where N is the weighted number of units total population, H is (weighted) units with IncomeVar (see [below](#)) below PovertyLine (see [below](#)), z is the PovertyLine and y is income as denoted by IncomeVar. Hence, this option gives the product between the headcount poverty rate (H/N) and a poverty gap measure equal to PovertyLine minus mean (rather than median) IncomeVar of the population under PovertyLine, divided by PovertyLine.

Named Parameters

PovertyLine: **required**; a Saved Variable containing the poverty line

IncomeVar: **required**; used for calculating the Median as the 5th decile cut-off (i.e. as input for CreateDeciles)

UseSwitchApproach: default = false (see description above)

SdcPoorObsOnly: default = false; if true the SDC observation count (see [below](#)) counts the observations classified as poor only

Example

```
<Action>
  <CalculationType>CalculatePovertyGap</CalculationType>
  <OutputVar>povGap</OutputVar>
  <Parameters>
    <Parameter>
      <Name>IncomeVar</Name>
      <VarName>eqDispy</VarName>
    </Parameter>
  </Parameters>
</Action>
```

```

    </Parameter>
    <Parameter>
      <Reform>false</Reform>
      <Name>PovertyLine</Name>
      <VarName>povLine</VarName>
    </Parameter>
  </Parameters>
</Action>

```

CalculateArithmetic

Calculates what `<FormulaString>` instructs it to.

It does not work on all observations but just on pre-existing numbers (e.g. SavedNumbers, previous column results, or numeric literals).

Note: If the `<FormulaString>` refers to `DATA_VAR[]`, then the values of the first individual in the data are used.

Named Parameters

The `<FormulaString>` can refer to any number of arbitrary named parameters:

- Name/VarName pairs to refer to variables (as `DATA_VAR[]`) or SavedVariables (as `SAVED_VAR[]`)
- Or Name/NumericValue pairs to pass literals (as `TEMP_VAR[]`)

Example

Calculates the poverty line, i.e. 60% of the Variable deciles_eqDispy~dec~5 (typically generated by CreateDeciles)

```

<Action>
  <CalculationType>CalculateArithmetic</CalculationType>
  <FormulaString><![CDATA[SAVED_VAR[@MeanDec] * 0.6]]></FormulaString>
  <OutputVar>povLine</OutputVar>
  <Parameters>
    <Parameter>
      <Name>MeanDec</Name>
      <VarName>deciles_eqDispy~dec~5</VarName>
    </Parameter>
  </Parameters>
</Action>

```

CalculateSumWeighted

Calculates a weighted sum of the observations, based on the `<FormulaString>`. Note that the sum of each `DATA_VAR[]` is calculated before the rest of the `<FormulaString>` is calculated.

Named Parameter

WeightVar: described [above](#)

RecodeNegatives: default = false; if true, negative values are set to zero

SdcNonZeroObsOnly: default = false; if true the SDC observation count (see [below](#)) omits observations with zero-value

Example

Calculates “weighted sum (yem) + weighted sum (yse)”, taking one as a parameter and the other directly in the formula.

```

<CellAction>
  <CalculationType>CalculateSumWeighted</CalculationType>
  <FormulaString><![CDATA[DATA_VAR[@SumVar1] + DATA_VAR[@yse]]]></FormulaString>
  <Parameters>
    <Parameter>
      <Name>SumVar1</Name>
      <VarName>yem</VarName>
    </Parameter>
  </Parameters>
</CellAction>

```

CalculateWeightedAverage

Calculates a weighted average of the observations, based on the `<FormulaString>`. Note that the average of each DATA_VAR[] is calculated before the rest of the `<FormulaString>` is calculated.

Named Parameter

WeightVar: described [above](#)

RecodeNegatives: default = false; if true, negative values are set to zero

SdcNonZeroObsOnly: default = false; if true the SDC observation count (see [below](#)) omits observations with zero-value

Example

```
<Action>
  <CalculationType>CalculateWeightedAverage</CalculationType>
  <FormulaString><![CDATA[DATA_VAR[@yem]]]></FormulaString>
  <CalculationLevel>Household</CalculationLevel>
  <OutputVar>avgYem</OutputVar>
</Action>
```

CalculatePopulationCount

Calculates a population count of the observations, based on the `<FormulaString>`. Note that the population count of each DATA_VAR[] is calculated before the rest of the `<FormulaString>` is calculated.

Named Parameters

WeightVar: described [above](#)

GiveTotals: default = false; if true, the number of observations larger than zero is calculated instead of the percentage

Threshold: default = 0; counts only observations with “value < -threshold || value > threshold”

RecodeNegatives: default = false; if true, negative values are set to zero

Example

Calculates the weighted number of observations for which the variable IsPoor is larger than zero

```
<CellAction>
  <CalculationType>CalculatePopulationCount</CalculationType>
  <FormulaString><![CDATA[DATA_VAR[@CountVar]]]></FormulaString>
  <Parameters>
    <Parameter>
      <Name>CountVar</Name>
      <VarName>IsPoor</VarName>
    </Parameter>
    <Parameter> <!-- by omitting this parameter a percentage instead of the number would be calculated -->
      <Name>GiveTotals</Name>
      <BoolValue>true</BoolValue>
    </Parameter>
  </Parameters>
</CellAction>
```

Empty

This is only applicable to `<CellAction>` and allows for empty cells.

Example

```
<CellAction>
  <CalculationType>Empty</CalculationType>
</CellAction>
```

Info

This is only applicable to `<CellAction>` and uses the `<FormulaString>` to display a text. Allows for File-References and UserVariables (see [above](#)).

Example

```
<CellAction>
  <CalculationType>Info</CalculationType>
```

```
<FormulaString>[baseSysPretty]</FormulaString>
</CellAction>
```

Statistical Disclosure Control (SDC)

SDC allows for:

- determining and hiding **primary SDC cells**, i.e. cells whose value is based on an observation count (number of unweighted data-rows) less than a certain minimum observation count (see below) and
- preventing deduction of primary SDC cells from other cells by determining and hiding **secondary SDC cells**, i.e. cells which
 - belong to the same **Secondary Group** (see below) as the primary SDC cell and
 - have the next smallest observation count.

The **minimum observation count** necessary to determine a primary SDC cell is defined by the property

`<TemplateInfo><SdcDefinition><MinObsDefault>` (see [above](#)). As the name suggests, this default can be overwritten on `<Table>`-, `<Column>`-, `<Row>`-, or `<Cell>`-level, by using the property `<SdcDefinition><MinObsAlternative>` (see [above](#)).

This property either provides a concrete number or a template-wide known alternative number, which can be defined by using the property `<TemplateInfo><SdcDefinition><MinObsAlternative>` (see [above](#)).

For secondary SDC it is necessary to put cells into **Secondary Groups**, which can be defined on table-, column-, row and cell-level. The most common application would be a column or row with a total as last cell, as illustrated in the following example:

Column 1	Column 2	Column 3	Column 4	Column 5	Total
5,023 (150 obs)	Prim SDC (20 obs)	4,525 (500 obs)	Sec SDC (50 obs)	7,649 (200 obs)	20,000 (920 obs)

For this purpose it is only necessary to equip the `<Row>` with

```
<SecondaryGroups>
  <AutoGroup/>
</SecondaryGroups>
```

If the interdependent cells do not form a whole row or column, one can exclude cells by using the property

`<SuspendSecondaryGroups>`. In the example, if the cells in Column 1 are independent, one would use this property on the `<Column>` definition for Column 1.

It is however also possible to put not neighbouring cells into a secondary group by using the property `<NamedSecondaryGroup>` (see [above](#)). This property must provide a template-wide unique name and thus links cells using this property with the same name, i.e. it is possible to have cells which do not even reside in the same `<Page>` in a secondary group.

Note that a cell can be part of more than one secondary group, by either defining directly two groups for the cell, or more likely: the cell is the cross section of a row with a `</AutoGroup>` and a column with another `</AutoGroup>`.