Custom Functions

Overview

top

Custom functions are functions that are hand designed and that are stored in XML format on the Yellowfin server. These functions usually consist of advanced SQL functions that cannot be easily generated by the formula builder. These functions are configured by your system administrator.

The XML schema allows for the custom defined SQL functions to have parameters embedded so that numeric or column values can be assigned when the column is inserted into a report.

Adding new Custom Functions

To add new XML functions into Yellowfin:

1. Copy your XML into the custom-functions.xml file located in: `Yellowfin/appserver/webapps/ROOT/WEB-INF/`
2. Once these have been added into the directory you will need to restart Yellowfin for these to become available.

> *See Calculated Fields for more information on using Custom Functions within Yellowfin.*

Schema

top

Parent Object	Object	Description
	`<custom-function>`	Most senior hierarchical object, encapsulates entire document. Only has one child type, `<function>`.
`<custom-function>`	`<function>`	This defines a unique function, and encapsulates the data fields needed to define it. Child objects include `<name>`, `<argument>` , `<sql>`, `<return>` and `<aggregate>`.
`<function>`	`<name>`	Name defines the display name of the custom function within the application. It is the primary identifier for each of the custom function, therefore the names of each function must be unique. Each function can only have one name only.
`<function>`	`<argument>`	Argument defines an argument (or parameter) that can be inserted into the custom SQL at report design time. Arguments are defined by 3 attributes, `<index>`, `<name>` and `<datatype>`.
`<argument>`	`<index>`	The index parameter of argument uniquely identifies the argument in the context of the function and must be an integral positive value. The index is used for inserting the argument at the defined location within the custom SQL statement. If the index is “1” then the argument will be replaced in the SQL statement for every instance of “$1”.
`<argument>`	`<name>`	The name parameter of argument is the display name for the argument in the application.

<argument>	<datatype>	The datatype parameter of argument defines the datatype of the argument. This allows the application to only allow compatible columns and values to be entered into this argument. Datatype must be one of the following: • TEXT • NUMERIC • DATE • TIME • DATETIME • TIMESTAMP • GEOMETRY
<function>	<sql>	SQL defines the actual database SQL statement to be made for this custom function. The SQL in this field will be inserted into a parent SQL statement as a single column, so a full SELECT FROM WHERE is not required and therefore the SQL must be compatible with single column syntax. See the example below for a simple CASE WHEN ELSE END example of a single column custom function. This SQL can also contain variables where arguments should be inserted. “$1” will be replaced with the column or data value assigned to the argument with index 1.
<function>	<aggregate>	The aggregate parameter defines which columns are aggregated within the custom function. This tells the application to not place these columns in the GROUP BY clause when generating the report SQL. The value of the aggregate parameter can also be a argument variable, for instance “$1” for the argument with index 1.
<function>	<groupby>	The group by parameter defines which columns should be inserted into the GROUP BY clause when the application is generating the report SQL. The value of the group by parameter can also be a argument variable, for instance “$1” for the argument with index 1.
<function>	<database>	This specifies which database this function should be available for. If none are specified it will be shown for all. There should be one object per database. Examples are: SQLServer, PostgreSQL, OpenEdge, Progress, Oracle, DB2, Access, Notes, ODBC, HSQL, or MySQL.
<function>	<return>	The return function defines the data type of the information that is returned by the entire custom function. This must be one of the following: • TEXT • NUMERIC • DATE • TIME • DATETIME • TIMESTAMP • GEOMETRY

Example CASE Statement

custom-functions.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<custom-functions>

<!-- functions are stored by name, so names must be unique, even across different databases -->

  <!-- ratio function -->
  <function>
    <name>Ratio</name>
    <argument>
      <index>1</index>
      <name>Numerator</name>
      <datatype>numeric</datatype> <!-- numeric, text, datetime -->
    </argument>
    <argument>
      <index>2</index>
      <name>Denominator</name>
      <datatype>numeric</datatype> <!-- numeric, text, datetime -->
    </argument>
    <sql>
      <![CDATA[
        CASE
          WHEN SUM($2) != 0 THEN SUM($1) / SUM($2)
          ELSE NULL
          END
      ]]>
    </sql>
    <aggregate>$1</aggregate>
    <aggregate>$2</aggregate>
    <database>SQLServer</database> <!-- Available for what DBs? SQLServer, PostgreSQL, OpenEdge, Progress,
Oracle, DB2, Access, Notes, ODBC, HSQL, MySQL -->
    <database>HSQL</database>
    <return>numeric</return> <!-- numeric, text, datetime -->
  </function>

</custom-functions>
```

top