Calculations

A Calculation is very much like a database column, except that its value, rather than being stored, is dynamically computed each time it is requested. This is achieved by creating a JavaScript function which is bound to a database table. The function takes no arguments and returns a value, which like a real database column, is declared to be one of the five general data types. The scope of the JavaScript function is an individual record object. Therefore any of the record's other data providers, and related foundsets are immediately available, as well as global variables and globally related foundsets.

A calculation is declared at the solution level, and is available throughout the solution in which it is declared, as well as any modules containing it. To support this, there is one JavaScript file per table, per solution, which holds calculation functions. For example, the database tables 'orders' may have a corresponding file 'orders_calculations.js' in any solution. And this file could contain many individual calculation functions.

Just like real database columns, calculations may be placed as fields on forms, used in data labels, and requested programmatically.

> ⚠ **Performance**
>
> Calculations may be called often. Therefore, developers should use discretion when implementing their JavaScript function. Most in-memory operations are very fast. However, actions which result in SQL queries or file operations may be slower and less predictable. For example, a calculation may be shown in a scrolling table-view form, in which case it may be called hundreds of times.

Example

A simple calculation *subtotal* on a database table *order_details* which yields a Number value; the unit price, multiplied by the quantity, with a discount applied.

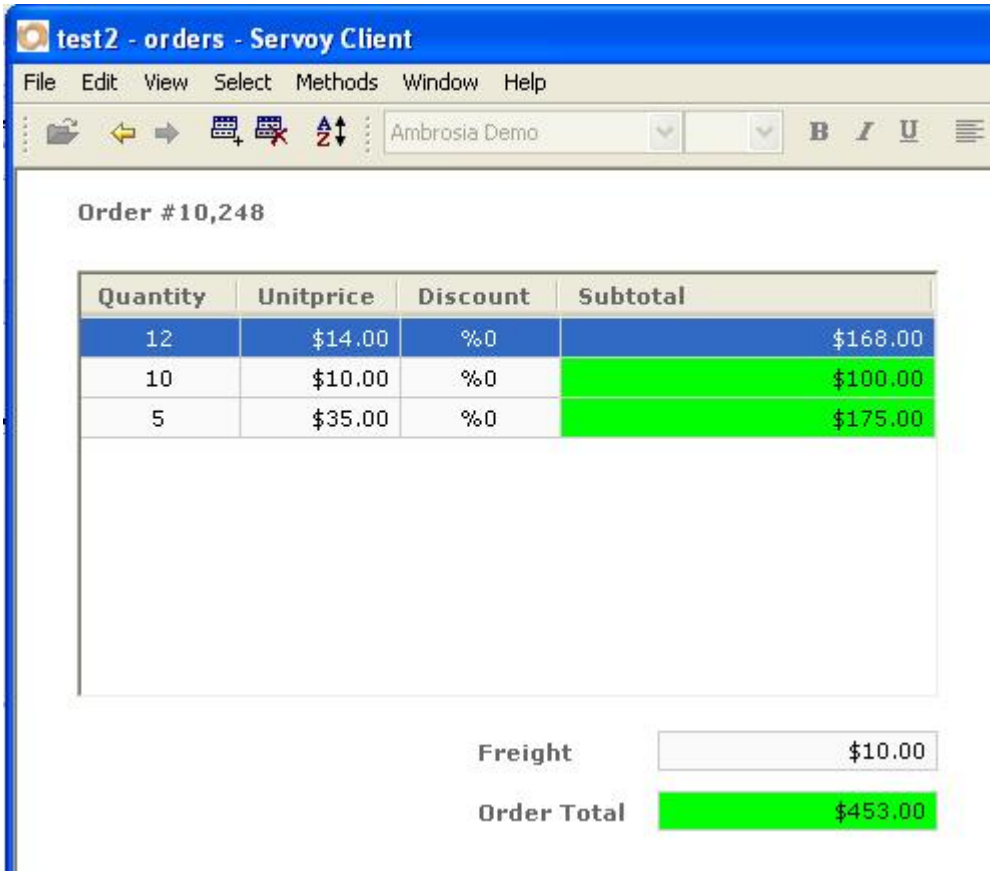
```
/**
 * @properties={type:6,typeid:36,uuid:"644DCF7D-11C7-475E-82CD-F4F60ED00D77"}
 */
function subtotal()
{
	return unitprice * quantity * (1 - discount);
}
```

This function is executed for the scope of an individual record. Notice that the record's other data providers, *unitprice* and *quantity* are immediately available.

This calculation could now be placed on a form or used in a method, just like any other real database column. The next code example shows another calculation, this time declared for the orders table, which uses the previous calculation to determine the total amount for the entire order.

```
/**
 * @properties={type:6,typeid:36,uuid:"d5458801-bbd2-43a1-994d-4ac2f6d12595"}
 */
function order_total()
{
	var sum = 0;
	for(var i = 1; i <= orders_to_order_details.getSize(); i++){
		sum += orders_to_order_details.getRecord(i).subtotal;
	}
	return sum + frieght;
}
```

Notice that the related foundset, *orders_to_order_details*, a property of an order record, is immediately available. Also notice that the previous calculation, *subtotal*, may be referenced, just like any other column. Note that while, calculations are a JavaScript function, unlike regular methods, one does not use the parentheses to invoke their value. Both calculations may be placed on forms like any other data provider. (See image below - calculations are highlighted in green)



Stored Calculations

Calculations may be optionally stored back to a real database column. This is called a *Stored Calculation,* and is achieved by creating a calculation which is the same name and data type as a real column. When the calculation is referenced, it will be executed and its result will be stored back in the database column.

> ⚠ Bear in mind that a stored calculation is not guaranteed to be recently calculated when used in a find, a sort definition, or as the right-hand key in a relation, because its value is computed in memory and the database value only represents the most recent execution of the calculation.

Calculation As a Record-Level Variable

In general, a calculation is a ready-only data provider, its value being generated each time it is read. However, in one exception a calculation may be used to cache data for a record, thus becoming a read/write in-memory data provider. This is done by creating a calculation which has no *return* statement. Such a calculation will be treated a little differently. It can actually store a value in memory for an individual record.

Example In this example, a record-level variable is created by defining an empty calculation.

```
/**
 * defined in my table.js
 * @properties={type:12,typeid:36,uuid:"32BE69DF-289E-45A6-A347-50271F3F29D7"}
 */
function record_flag()
{
        // NO return statement / value
}
```

Next a method illustrates how the calculation may be used to store, in memory, information about the record.

```
/**
 * @param {JSEvent} event
 *
 * @properties={typeid:24,uuid:"75E3667B-AED9-4E79-ADE8-CCBED66A8D2F"}
 */
function flagSelectedRecord(event){

	// toggles the flag between 0 & 1
	if(record_flag){
		record_flag = 0
	} else {
		record_flag = 1;
	}
}
```