# eCart Calculations, Discount and Charge Rules

## Introduction

This TechNote discusses calculations in WA eCart. It provides both general guidelines for use and specific examples of custom calculations.

There are two main ways in which your eCart object can be customized. The first and most basic update is to add custom columns to store data specific to what you are selling. For instance Size, Color, and Thumbnail are all common columns to add. Custom columns must be created for any data you will need later to properly bill and fulfill the cart order.

WA eCart objects include six columns by default:

- ID
- Name
- Description
- Weight
- Quantity
- Price

The second customization is to use calculations or calculated columns. Calculations allow you to define a formula to calculate an item-by-item value that is derived from the values stored in the cart columns.

## General rules for the Calculations tab:

The WA eCart Calculations tab comes with three default calculations:

| Name | Calculation Formula | Type |
|---|---|---|
| TotalWeight | [Weight] * [Quantity] | W |
| TotalPrice | [Price] * [Quantity] | C |
| FullDetails | [Quantity] & " " & [Name] & " (" & [ID] & ")" | T |

These three calculations can be changed but they cannot be deleted from the WA eCart Object.

- TotalWeight is used for weight based shipping implementations
- TotalPrice is used to find the total cost of any line of items in the cart… the sum of this column is the subtotal of the cart before discounts and charges. The TotalPrice value is passed to the shopping cart to calculate the GrandTotal in the WA eCart application.
- FullDetails is used to pass cart details to payment gateways that don't support individual line items passed from the cart. This string value is meant to be a quick summary to allow you to identify the contents of an order.

The order in which the rules are listed is significant. A custom calculation can use values from any other calculation that appears above the custom calculation in the list. For example, let's say you've created two custom calculations, TruePrice and TotalDiscount. If TotalDiscount includes TruePrice in it's forumula, TruePrice would have to be defined before TotalDiscount.

## Categories of item-level price adjustments:

Calculations are used whenever adjustments need to be made on a row by row basis in the cart.  This includes, but is not limited to:

- Offering options at an additional cost (oversized items, engraving, etc.)
- Quantity discounts based on the number of the same items ordered
- Taxable and non-taxable items in the same cart

Generally, custom calculations involve custom options that affect the price of the item being purchased. From lens and frame options for glasses, to custom engraving on awards and baseball bats, to popcorn flavors and pizza toppings - the gamut of custom options can be represented by a broad array. However, the principle applied to such situations is always the same: the base price is augmented by a calculated charge.

Another category of custom calculation is the breakpoint calculation. This usually involves a quantity charge like a shipping charge that is applied at certain weight intervals. Sometimes these kinds of charges involve rounding a number up or down so

that the range of the values to be assessed always yields a whole number that can be used to return a rate based on the specified range.

The third category of custom calculation is a line-item percent calculation. Occasionally taxes will be assesses on single items in the cart, and you'll want to collect that data as a line item charge for that product.

## Evaluating conditions in a calculation:

Calculations refer to columns in the shopping cart by enclosing that column in brackets. For example, lets say you add a column to your cart named "Shipping". Now you want to be able to add an amount to that column and have it be the per-item shipping cost. One way to do this is to adjust the existing calculation for TotalPrice. By default, the formula for TotalPrice is:

```
TotalPrice = [Price] * [Quantity]
```

To incorporate your Shipping column, you could rewrite it to:

```
TotalPrice = ([Price] + [Shipping]) * [Quantity]
```

With this revised calculation, the shipping cost is automatically included in the line total in the cart. Another method would be to leave the TotalPrice alone and use a Charge Rule to account for the price difference. Charge rules are shown below the cart instead of on the individual line. To use a charge rule, you would create your own calculation:

```
TotalShipping =  [Shipping] * [Quantity]
```

The values from charge rules like this are displayed below the cart contents.

Another example of using calculations involves taxable and non-taxable items. Let's say that you want to charge 8.5% for taxable items in your store. To do this, you could create a column in the cart called "Taxable" and set its value to 1. If the added item is taxable the value remains at 1; if it is not, the value should be set to 0. This shopping cart column can be bound to a true-false (or Boolean) database column. This arrangement allows you to use the following calculation:

```
Tax = [Taxable] * [TotalPrice] * 0.085
```

In this example, the Tax value is either zero or 8.5% of the [TotalPrice]. Since the Tax definition is below the TotalPrice calculation, you can reference TotalPrice just like you would a column, by enclosing it in square brackets: [TotalPrice].

## Incorporating server-side code in your calculation:

A calculation can include function calls using the server language your site is defined for. Each server language uses its own method for evaluating a Boolean (true/false) expression to return the numeric equivalent. The following table shows the available functions that can be used with calculations to return conditional values:

| | ASP-JScript | ASP-VBScript | ColdFusion | PHP |
|---|---|---|---|---|
| **Function** | Math.abs() | Abs() | abs() | abs() |

In each case the expression evaluated is placed in the parenthesis. If the condition is true, the value returned is 1, if the condition is false, the value is 0.

For example, if you wanted to have taxable and non-taxable items, but instead of using a cart column for Taxable, you already have a cart column for Category and you know everything is taxable except items with a Category of ‘grocery’. You could then create a calculation:

```
[ASP-JS]  Taxable = Math.abs([Category] != "grocery")

[ASP-VB]  Taxable = Abs([Category] <> "grocery")

[CF]  Taxable = abs([Category] DNE "grocery")

[PHP] Taxable = abs([Category] != "grocery")
```

In the above examples, the [Category] column in the eCart is evaluated to see if it has a value other than "grocery". If the condition is true, 1 is returned, if the condition is false, a 0 is returned.

The calculation can include any function calls and is written in the syntax of your declared server language. This means there is often more than one way something can be accomplished. For example: PHP and ASP-JS support the ternary operator which allows you to test for a condition followed by a question mark then express the true:false results as colon-separated values. CF has a native IIf function which allow for a similar result. We have included an IIf equivalent function for ASP-VB users. Here is the same calculation as above using this alternate syntax:

```
[ASP-JS]  Taxable = ([Category] != "grocery")?1:0

[ASP-VB]  Taxable = WA_eCart_IIf(([Category] DNE
"grocery"),1,0)
```

```
[CF]  Taxable = IIf(([Category] DNE "grocery"),1,0)

[PHP] Taxable = ([Category] != "grocery")?1:0
```

## Concatenating a sequence of values to return a single value:

You can also add a series of conditional statements together to return a single value for all of the possible conditions to be met. An example of this is shown in the WA eCommerce Recipes, Quantity Discounts, in which a calculation determines the price with a 10% discount if  3 or more items are ordered and a 20% discount if the customer orders 5 or more:

```
[ASP-JS] [Price] - ((Math.abs([Quantity]>=3)*[Price]*0.1) +
(Math.abs([Quantity]>=5)*[Price]*0.2))

[ASP-VB] [Price] - ((Abs([Quantity]>=3)*[Price]*0.1) +
(Abs([Quantity]>=5)*[Price]*0.2))

[CF] [Price] - ((abs([Quantity] GTE 3)*[Price]*0.1) +
(abs([Quantity] GTE 5)*[Price]*0.2))

[PHP] [Price] - ((abs([Quantity]>=3)*[Price]*0.1) +
(abs([Quantity]>=5)*[Price]*0.2))
```

Here the absolute value function evaluates an expression that returns a 1 or a 0 which is then multiplied by the Price times the percentage discount.  Only the true condition renders a discount value, which is then subtracted from the Price.

The eCart Discounts Solution Recipe provides more details on how to do various calculations.

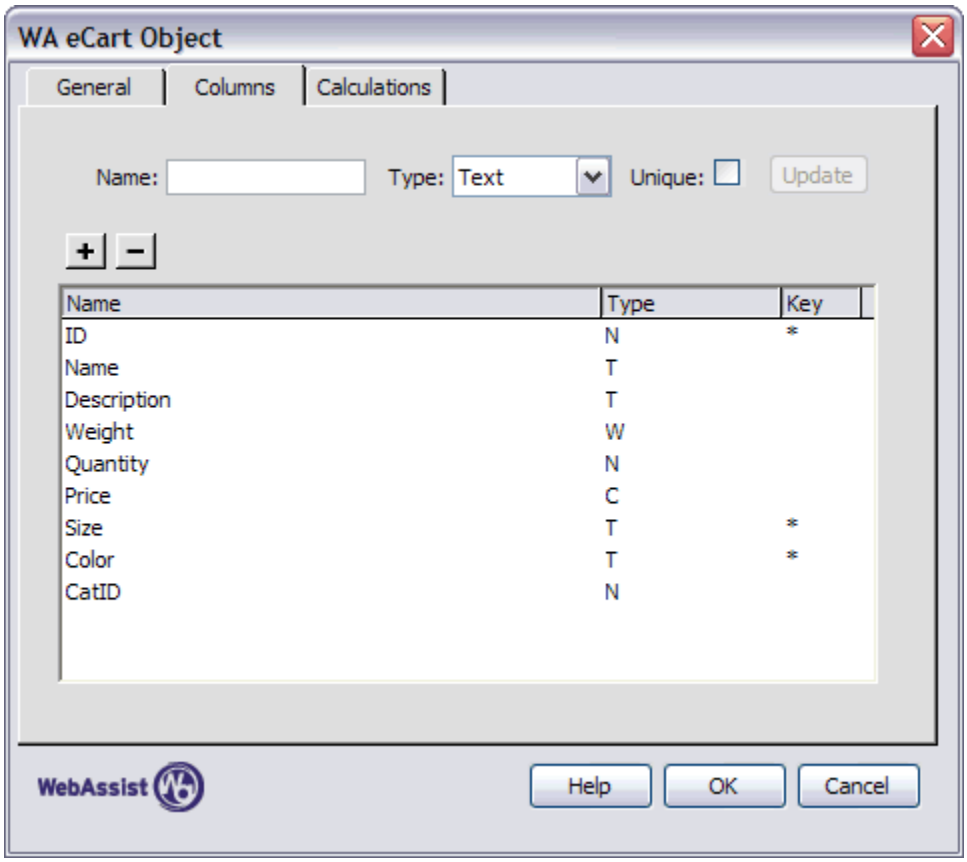
## Calculation that augments the [TotalPrice]

In this example, you'll see how to charge a different price for an option, such as an oversized item.  For this example, let's make the following assumptions:

- Mens shoes will go up by $2 if they are above a size 8.
- There will be a $4 charge if they are over size 10.

- A $6 surcharge is added if they are over size 12.
- Womens shoes will have similar increments, but for sizes 7, 9, and 11 respectively.

Since our price will be determined by both the product category (mens and womens) and the size, we need to make sure both of those values will be available to us in the cart. Before adding the calculation, we must add a custom column to the eCart Object to store the ProdCatID value from the Products table in the database. This column distinguishs between Mens and Womens shoes; a Size column is also needed, but already included in the shoe store example.




Next, we add a custom calculation that will return an incremental increase of $2 for men's shoes at sizes 8, 10 and 12 and for women's shoes at sizes 7, 9 and 11. The basic formula is going to evaluate the following expression:

```
([CatID]=n AND [Size] >= x)
```

where *n* is the CatID taken from the ProdCatID - which returns 1 for men's shoes and 2 for women's shoes - and *x* is the size break point. Here's what the calculations would look like:

[ASP JScript]

```
((Math.abs([CatID]==1 AND [Size] >= 8)) * 2) +
((Math.abs([CatID]==1 AND [Size] >=10)) * 2) +
((Math.abs([CatID]==1 AND [Size] >= 12)) * 2) +
((Math.abs([CatID]==2 AND [Size] >= 7)) * 2) +
((Math.abs([CatID]==2 AND [Size] >= 9)) * 2) +
((Math.abs([CatID]==2 AND [Size] >= 11)) * 2)
```

[ASP VBScript]

```
((Abs([CatID]=1 AND [Size] >= 8)) * 2) + ((Abs([CatID]=1
AND [Size] >= 10)) * 2) + ((Abs([CatID]=1 AND [Size] >=
12)) * 2) + ((Abs([CatID]=2 AND [Size] >= 7)) * 2) +
((Abs([CatID]=2 AND [Size] >= 9)) * 2) + ((Abs([CatID]=2
AND [Size] >= 11)) * 2)
```

[CF]

```
((abs([CatID] EQ 1 AND [Size] GTE 8)) * 2) + ((abs([CatID]
EQ 1 AND [Size] GTE 10)) * 2) + ((abs([CatID] EQ 1 AND
[Size] GTE 12)) * 2) + ((.abs([CatID] EQ 2 AND [Size] GTE
7)) * 2) + ((abs([CatID] EQ 2 AND [Size] GTE 9)) * 2) +
((abs([CatID] EQ 2 AND [Size] GTE 11)) * 2)
```

```
[PHP]
((abs([CatID]==1 && [Size] >= 8)) * 2) + ((abs([CatID]==1
&& [Size] >= 10)) * 2) + ((abs([CatID]==1 && [Size] >= 12))
* 2) + ((abs([CatID]==2 && [Size] >= 7)) * 2) +
((abs([CatID]==2 && [Size] >= 9)) * 2) + ((abs([CatID]==2
&& [Size] >= 11)) * 2)
```

With this calculation, when the shopper selects a men's shoe size under 8, there is no increment, at size 8 to 9.5 the increment will be $2. At size 10 to 11.5, the price will be incremented to $4 and any size over 12 will be incremented by $6. For women's shoes there will be no increment for shoes under size 7. At size 7 to 8.5 a $2 increment will be added, at size 9 to 10.5 a $4 increment will be added and all sizes from 11 and above will incur a $6 increment.

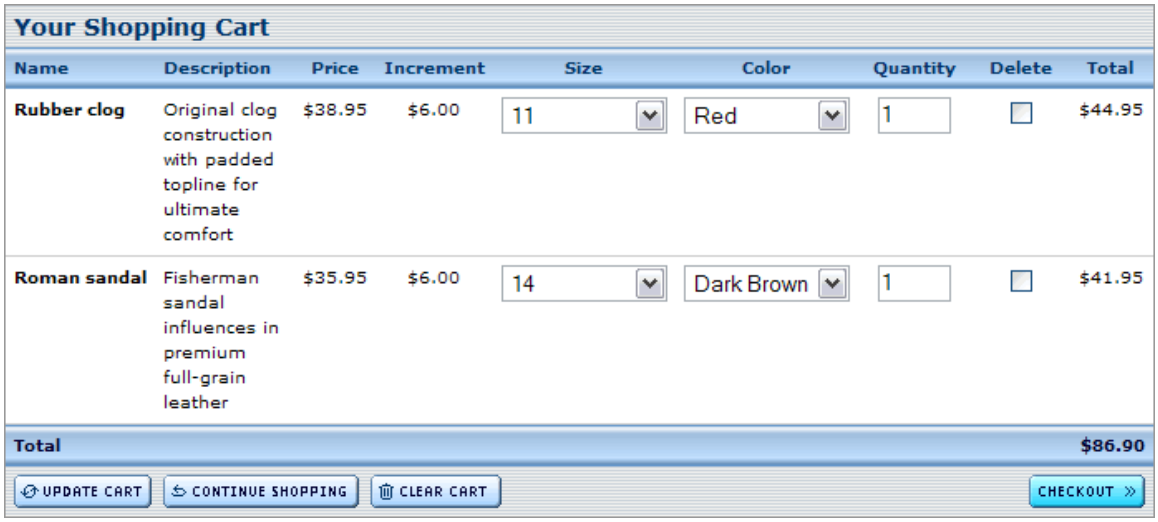
But this calculation has not been applied to the TotalPrice yet. The next step is to copy this calculation and add it to the TotalPrice calculation. To do this, copy the SizeIncrement calculation, then click on the TotalPrice calculation and place a parenthesis before the first square bracket for [Price] then after the ending squre bracket for [Price] add a "+" followed by an open and close parenthesis. Add a close parenthesis after that so that it looks like this:

```
([Price]+()) * [Quantity]
```

Place your cursor between the open and close parenthesis after the + operator and paste the SizeIncrement calculation between those to parenthesis like this:

```
[ASP-VB]
([Price]+(((abs([CatID]=1 AND [Size] >= 8)) * 2) +
((abs([CatID]=1 AND [Size] >= 10)) * 2) + ((abs([CatID]=1
AND [Size] >= 12)) * 2) + ((abs([CatID]=2 AND [Size] >= 7))
* 2) + ((abs([CatID]=2 AND [Size] >= 9)) * 2) +
((abs([CatID]=2 AND [Size] >= 11)) * 2))) * [Quantity]
```

When you test your application you should see that the TotalPrice value in the cart display shows the increment that is generated when you change the size value and update the cart:




## Adding a line-item percentage charge

Our next example shows how to add a line-item tax percentage charge using the Calculations panel. Let's say certain of your products require a Value Added Tax while others do not. For this calculation, the first step is to create a custom column in your shopping cart to hold the tax rate (here, called TaxRate), as set in a database column. Then, in the Calculations tab of the WA eCart object, set Tax to be equal to the TotalPrice times the TaxRate:

| Name | Calculation Formula | Type |
|---|---|---|
| Tax | [TaxRate] * [TotalPrice] | C |

You can then use the *Based on column subtotal* rule calculation in the Merchandising Wizard when you define your overall charges. Use the custom Tax column as the column to be used in the charge calculation. In this way, an overall tax can be assessed but will only be charged for items that have a TaxRate value applied to them.

To include your tax value in the line item TotalPrice, remember the rule about the order of your calculations. You'll need to add you're your Tax calculation to the TotalPrice like this:

```
[Price] * [Quantity] + (([Price] * [Quantity]) * [TaxRate])
```

The value of ([Price] * [Quantity]) is multiplied times the TaxRate and then that value is added to the original [Price] * [Quantity] calculation.

## Math operator reference

Each server language has its own set of Math functions and operators that you can use in your calculations. Here are some resources you should acquaint yourself with:

JavaScript Operators:

http://www.w3schools.com/js/js_operators.asp

JavaScript Functions:

http://www.w3schools.com/js/js_functions.asp

VBScript Functions:

http://www.w3schools.com/vbscript/vbscript_ref_functions.asp

ColdFusion Expressions: Operands, Operators, and other Constructs:

http://www.macromedia.com/devnet/server_archive/articles/cf_expressions.html

PHP Operators:

http://us3.php.net/manual/en/language.operators.php

PHP Functions:

http://us3.php.net/manual/en/ref.math.php

.