



Product Model Object Tags	
Tag	Product Tag Definitions
<*Quantity(attribute name)*>	returns the user entered quantity for the attribute
<*Price(attribute name)*>	returns the price adder of the current attribute selection
<*Value(attribute name)*>	returns the display value of the current attribute selection
<*ValueCode(attribute name)*>	returns the value of the current attribute selection
<*CatCode(attribute name)*>	returns the value catalogue code of the current attribute selection
<*IsSelected(attribute name.attribute display value)*>	returns "1" if "attribute name.attribute display value" is currently selected, and "0" if it is not selected
<*AttSel(attribute_name)*>	returns "1" if the "attribute_name" is currently selected, and "0" if "attribute_name" is not currently selected. Same as <*AttDef()*> but checks every selection of an attribute
<*AttDef(attribute_name)*>	returns "1" if the "attribute_name" is currently selected, and "0" if "attribute_name" is not currently selected. Same as <*AttSel()*> but checks a status flag and is faster for a long list of attribute values
<*CXSUB(intelligent sequence substitution name)*>	see chapter 30
<*Price(*)*>	returns the sum of all prices for all currently selected attributes and does not include "base price". This would be different than "TotalPrice" if there is a "base price" and/or if TotalPrice is calculated using "Product Price building sequence" See Product Definition section.
<*ProductCode*>	returns the part number of the current configuration
<*TotalPrice*>	returns the price of the current configuration which would include items which would be designated as separate line items but may be different if it is calculated using "Product Price building sequence."
<*ProductName*>	returns the product name
<*GETATVUPC(attribute_name : attribute_value)*>	returns the Price of the attribute Value of the currently selected attribute <*GETATVUPC(Color:Blue)*> - returns the Price of the Value "Blue" of the Attribute "Color"
<*SETATVQTY(AttributeName : StandardAttributeDisplayValue, Quantity)*>	sets the Quantity of the Value of the currently selected attribute EXAMPLE: <*SETATVQTY(Color:Bule,5)*> - sets the Attribute Value "Blue" of the Attribute "Color" to 5
<*GETATVQTY(AttributeName : StandardAttributeDisplayValue)*>	returns the Quantity of the Value of the currently selected attribute EXAMPLE: <*GETATVQTY(Color:Blue)*> - returns the Value "Blue" of the Attribute "Color"
<*BasePrice*>	returns the base price of the product (which is the static number listed in the product price field)



<code><*ProductDescription*></code>	returns the product description
<code><*LongProductDescription*></code>	returns the long product description
<code><*ProductLink*></code>	returns the link or a URL from the product link field
<code><*ProductImage*></code>	returns the product image
<code><*LargeProductImage*></code>	returns product large image
<code><*TotalPriceWoLi*></code>	returns the Total Product price excluding prices of attributes that are designated as separate Line Items
<code><blank/></code>	inserts a blank space in the part number
<code><space/></code>	inserts a blank space in the part number
<code><*VISITORGROUP*></code>	returns the name of a user group that a logged in user belongs to
<code><*VISITORNAME*></code>	returns the real name of the user
<code><*VISITORGROUPID*></code>	returns the group id of the user that is logged
<code><*VISITORID*></code>	returns the user id
<code><*PSEQ(sequence name)*></code>	product sequence
<code><*GSEQ(sequence name)*></code>	global sequence
<code><*SSEQ(sequence 1 name, sequence 2 name, ..., sequence n name)*></code>	nested sequences
<code><*EVAL(param)*></code>	<p>evaluates MATH expression passes as parameter "param" can contain operators, functions and everything else as available in basic scripting</p> <p><code><*EVAL(param)*></code> tag engages VBScript engine, so we can use all VBScript functions – see Appendix A for list of VBScript functions.</p> <p>EXAMPLE: <code><*EVAL(<*value(some attribute)*sin(<*valuecode (some other attribute)*>))*></code></p>
<code><*round (value, <number_of_digits_after_decimal>)*></code>	round "value" to number_of_digits_after_decimal – this parameter is optional
<code><*PRODUCTINFO(TYPE)*></code>	<p>returns defined product type NAME. This can be used when defining specifics correlated with product type (discount for instance) using auxiliary tables (probably combined with customer info).</p> <p>EXAMPLE: <code><*PRODUCTINFO(ProductType_CD)*></code> can return CD from the system</p>



<*XWS(AUXTABLE, ...)*>	See below
<*XWS(function, parameter)*>	for example <*XWS(getProductPrice, Product Name)*> or <*XWS(getProductCatCode, Product Name)*> or <*XWS(getProductDescription, Product Name)*> - will return external data, such as from another product (it is possible to retrieve some data even from another application)
<*AUX(Customer_Info, CUSTOMER_ZIP_CODE, ID, <*GI(CustomerID)*>)*>	Returns Customer Info. This tag works only if it is set for some user group, or globally, on application level, is set 'customer required' option.
<*XWS(VALIDATERAISE, CatNumber)*>	CatNumber Validation Example: <*XWS(VALIDATERAISE, <*Value(CatNumber1)*>)*> - validate RAISE Product CatNumber1
<*XWS(GETRAISEPRICE, CatNumber)*>	Returns the price of Ext. Raise product Example: <*XWS(GETRAISEPRICE, <*Value(CatNumber1)*>)*> If CatNumber is not Valid Alert appears and deletes the Value form Free Form input field.
<* XValue(Subproduct/Attribute)*>	The tag <*XValue()*> is used for Product Hierarchy (Nested Products). See Chapter 51.



Shopping Cart Object Tags

**<*CO_INFO(column_name_
in_users_table)*>**

retrieve information about cart's owner (table users)

The **<*CO_INFO(...)*>** tag is used to retrieve information about cart's owner. Similarly, the **<*CU_INFO(...)*>** tag can be used to retrieve information about a current user.

Additional information which can be accessed with that tag are:

<*CO_INFO(TERRITORY_NAME)*>- returns territory name

<*CO_INFO(USER_TYPE_NAME)*> - returns group of cart's owner

<*CO_INFO(COMPANY_NAME)*> - cart's owner's company

<*CO_INFO(APPROVE_PARENT_NAME)*> - returns name of the approve parent

<*CO_INFO(ORDERING_PARENT_NAME)*> - returns name of the ordering parent

<*CO_INFO(MANAGING_PARENT_NAME)*> - returns name of the managing parent



<*CO_INFO(column_name_
in_users_table)*>

<*CO_INFO(Name)*> - returns Name of cart's owner
<*CO_INFO(Email)*> - returns Email of cart's owner
<*CO_INFO(expdate)*> - returns Expiration Date
<*CO_INFO(companyID)*> - returns company ID of cart's owner
<*CO_INFO(user_id_code)*> - returns id code of cart's owner
<*CO_INFO(user_ZIP_CODE)*> - returns Zip Code of cart's owner
<*CO_INFO(user_CITY)*> - returns City of cart's owner
<*CO_INFO(user_state)*> - returns State of cart's owner
<*CO_INFO(multiplier)*> - returns multiplier of cart's owner
<*CO_INFO(phone)*> - returns phone of cart's owner
<*CO_INFO(fax)*> - returns fax of cart's owner
<*CO_INFO(country)*> - returns country of cart's owner
<*CO_INFO(TITLE)*> - returns title of cart's owner
<*CO_INFO(territory_id)*> - returns territory ID of cart's owner

EXAMPLE:

**[EQ](<*CU_INFO(TERRITORY_ID)*>,
<*CO_INFO(TERRITORY_ID)*>)**

The above expression will evaluate to true if a Current User (<*CU_INFO(...)*> stands for Current User) belongs to the same territory ([EQ] stands for equal) as a Cart Owner (<*CO_INFO(...)*>). This expression can be used in the quote (cart) sharing rules. This means that a current user will be able to see all quotes created by other users that have the same territory that his/she has.



<p><*BT_INFO(column_name_from_table_customer_info)*></p>	<p>retrieve billing information about customer (table customer_info)</p> <p>Tag BT_INFO returns information about customer (table customer_info):</p> <p><*BT_INFO(column_name_in_table_customer_info)*>.</p> <p>Additional information which can be accessed with that tag are:</p> <p><*BT_INFO(CUSTOMER_NAME)*> - customer last_name + ' ' + customer first name</p> <p><*BT_INFO(COUNTRY_NAME)*> - customer's country</p> <p><*BT_INFO(TERRITORY_NAME)*> - customer's territory</p> <p><*BT_INFO(Customer_E-Mail_Addr)*> - returns customer's Email</p> <p><*BT_INFO(Customer_Title)*> - returns customer's Title</p> <p><*BT_INFO(Customer_Company)*> - returns customer's Company</p> <p><*BT_INFO(Customer_City)*> - returns customer's City</p> <p><*BT_INFO(Customer_State)*> - returns customer's State</p> <p><*BT_INFO(Customer_Province)*> - returns customer's Province</p> <p><*BT_INFO(Customer_Zip_Code)*> - returns customer's Zip Code</p> <p><*BT_INFO(Customer_Business_Phone)*> - returns customer's Business Phone</p> <p><*BT_INFO(Customer_Business_Fax)*> - returns customer's Business Fax</p> <p><*BT_INFO(Customer_Primary_Industry)*> - returns customer's Primary Industry</p> <p><*BT_INFO(customer_id)*> - returns customer's ID</p> <p><*BT_INFO(territory_id)*> - returns customer's territory ID</p> <p><*BT_INFO(active)*> - returns 1 if customer is active, otherwise returns 0</p> <p><*BT_INFO(customer_code)*> - returns customer's code</p> <p>EXAMPLE:</p> <p>- <*BT_INFO(TERRITORY_ID)*> - returns customer's territory_id</p> <p>- <*BT_INFO(TERRITORY_NAME)*> - returns customer's territory name</p>
---	---



<p><*CO_STINFO(column_name_in_ship_to_table)*></p>	<p>retrieve shipping information about customer (table ship_to)</p> <p>Tag CO_STINFO returns shipping information - <*CO_STINFO(column_name_in_ship_to_table)*>.</p> <p>Additional information which can be accessed with that tag are:</p> <p><*CO_STINFO(COUNTRY_NAME)*> – ship country <*CO_STINFO(CUSTOMER_NAME)*> – ship last_name + ' ' + ship first_name</p> <p><*CO_STINFO(Shop_Cart_Id)*> – returns Shop Cart ID <*CO_STINFO(Ship_E-Mail_Addr)*> – returns ship Email <*CO_STINFO(Ship_Title)*> – returns ship Title <*CO_STINFO(Ship_Company)*> – returns ship Company <*CO_STINFO(Ship_City)*> – returns ship City <*CO_STINFO(Ship_State)*> – returns ship State <*CO_STINFO(Ship_Province)*> – returns ship Province <*CO_STINFO(Ship_Zip_Code)*> – returns ship Zip Code <*CO_STINFO(Ship_Business_Phone)*> – returns ship Business Phone <*CO_STINFO(Ship_Business_Fax)*> – returns ship Business Fax <*CO_STINFO(Ship_Primary_Industry)*> – returns ship Primary Industry <*CO_STINFO(ship_id)*> – returns ship ID</p> <p>EXAMPLE: [EQ](<*CU_INFO(USER_ZIP_CODE)*>, <*CO_STINFO(SHIP_ZIP_CODE)*>) The above expression will evaluate to true if a Current User has the same zip code as the zip code entered in customer's ship to address. This expression can be used in the cart (quote) sharing rules. This syntax would mean that a current user will be able to see all quotes created by other users if those quotes are intended to be shipped to the same zip code as current user's zip code.</p>
<p><*CARTAMOUNT*></p>	<p>returns total amount of the cart</p>
<p><*USERAMOUNT*></p>	<p>returns max approve amount for currently logged in user</p>
<p><*USERID*></p>	<p>returns users.id - same as <*CU_INFO(id)*></p>
<p><*CARTOWNER*></p>	<p>users.id for cart's owner – same as <*CO_INFO(id)*></p>
<p><*SAMETERRITORY*></p>	<p>for Quote Visibility, currently logged in user and cart's owner must be on the same territory, same as [eq](<*cu_info(territory_id)*>,<*co_info(territory_id)*>), returns 1 if it is true, 0 if it is false</p>
<p><*TERRITORY*></p>	<p>returns territory name for cart's owner, same as <*co_info(territory_name)*></p>



<*SAMEGROUP*>	for Group Visibility, currently logged in user and cart's owner must be in the same Group, same as [eq](<i><*cu_info(type)*></i>,<i><*co_info(type)*></i>) , returns 1 if it is true, 0 if it is false
<*GROUP*>	returns cart's owner group name, same as <*co_info(USER_TYPE_NAME)*>
<*ISRESPAPPROV*>	approve process - returns 1 if it is true, 0 if it is false
<*CHILDREN*>	approve process – is currently logged in user ordering parent for cart's owner?- returns 1 if it is true, 0 if it is false
<*CHILD*>	approve process – is currently logged in user direct ordering parent for cart's owner? - returns 1 if it is true, 0 if it is false
<*FIRSTAPPROVEC*>	approve process – is currently logged in user approve parent for cart's owner? - returns 1 if it is true, 0 if it is false
<*ALLAPPROVEC*>	approve process – is currently logged in user direct approve parent for cart's owner? - returns 1 if it is true, 0 if it is false
<*SAMEZIPCODE*>	Is the zip codes of currently logged in user and cart's owner the same? (Cart Visibility) and Is the zip codes of currently logged in user and customer the same? (Customer Visibility) returns 1 if it is true, 0 if it is false
<*ZIPCODE*>	returns User zip code
<*SAMESHIPZIPCODE*>	Is zip code of currently logged in user the same as zip code of the city where quote is delivered? - returns 1 if it is true, 0 if it is false
<*SHIPZIPCODE*>	returns Zip code of “Ship to:” address
<*SAMECUSTOMERZIPCODE*>	Is zip code of currently logged in user the same as customer's zip code? - returns 1 if it is true, 0 if it is false
<*CUSTOMERZIPCODE*>	returns Customer zip code
<*CUSTOMERCOUNTRY*>	returns Customer's Country
<*CUSTOMERCOMPANY*>	returns Customer's Company
<*APPROVALREQUIRED*>	This tag is often used as Condition for showing up "Submit for approval" button. Condition is true (1) if there is needed to execute Approval process for the loaded Cart. For example, if Administrator doesn't want to execute “Generate Quote” Action if is necessary to execute Approval Process before then the condition for “Generate Quote” Action will be [NOT]<*APPROVALREQUIRED*>
<*ISAPPROVED*>	This tag is used for checking if Quote is Approved (Quote passed through Approval Process and now is Approved) or not.



<*ISREJECTED*>	This tag is used for checking if Quote is Rejected (Quote passed through Approval Process and now is Rejected) or not.
<*ISWAITING*>	This tag is used for checking if Quote is into Approval Process but is neither Approve nor Rejected
<*CUSTOMERTERRITORY*>	Customer's Territory
<*GETATTNUMVAL*>	returns numeric value of attribute
<*SAMEUSERCOMPANY*>	If the current user works for the same company as the user who created a cart (quote) this expression will evaluate to 1, otherwise it is 0.
<*USERCOMPANY*>	Returns name of user company
<*AUX(AUXTABLE, ...)*>	See below, after description of XWS tags
<*CSIGN*>	returns current currency sign as defined in database
<*MFACTOR*>	returns market factor
<*MCODE*>	returns market code - if needed to access additional price books via AUX tags
<*CRATE*>	returns current currency rate (related to default currency as administered)
<*USER_CO_CRM_ACCOUNT_ID*>	returns CRM ID for a company that a user belongs to. This tag is used in a field for default discount, in side of a AUX tag to find a default discount for a company that user belongs to. Example of a formula in discounting rules for default discount: AUX(some table, discount column, crm id column,<*USER_CO_CRM_ACCOUNT_ID*>)
<*TotalItemsAmount*>	Returns Total Items Amount before shipping or commission costs
<*TotalItemsCost*>	Returns Total Items Cost before shipping or commission costs
<*TotalGrossMargin*>	Returns Total Gross Margin before shipping or commission costs
<*FREIGHTAMOUNT*>	Returns freight amount. Example: [if]([GT](<*FREIGHTAMOUNT*>,0)){1}{0}[endif]

WebSource also allows for external (non current product) data to be evaluated and used inside expressions.

The <*XWS()*> tag is used for that purpose. This tag should be used if a product needs to use data from another product, from any table located in the database, or from some other external source.

For example <*XWS(GETPRODUCTPRICE, Product Name)*> or <*XWS(GETPRODUCTCATCODE, Product Name)*> or <*XWS(GETPRODUCTDESCRIPTION, Product Name)*> would return product price, product part number or product description for a referenced product.



Another standard example of using the **XWS** tag would be using it to obtain data from a table within the configurator database. The **XWS** tag would look like: **<*XWS(AUXTABLE, table name, field name to be returned, field name for condition, condition value, field name 2 for condition, condition 2 value, ...)*>**. For example, if you want to return a value from the field CONTRACT_PRICE from a table called CONTRACTS, where part number in the PartNumber field is equal to the configured part number of the product, you would write the tag like this: **<*XWS(AUXTABLE,CONTRACTS,CONTRACT_PRICE,PartNumber,<* ProductCode *)*>**

AUX(AUXTABLE, ...) tags:

Same as **<*XWS(AUXTABLE, table name, field name to be returned, field name for condition, condition value, field name 2 for condition, condition 2 <value, .) *>** but works faster than **XWS** tags.

To avoid using http post (XWS tag with AUXTABLE function), use AUX tag instead to access uploaded table data syntax:

<*AUX(<table_name>, <return_column_name>, <condition_column_1_name>, <condition_value_1>, <condition_column_2_name>, <condition_value_2>.....) *>

General recommendation is to use **AUX** tags instead of **XWS** tags when work with XLS Tables.

Important: Use Product Model Object Tags for Product Modeling, Shopping Cart Object Tags for Shopping Carts, Word Generation Engine Tags for Word Quotes.



IF Statement

In addition to using tags, expressions can use “If” statements. The WebSource “If” syntax looks like this.

[If](condition) {expression if true}{expression if false –note this is optional} [ENDIF]

The operators used in “If” statements are as follows. Note that there can only be two variables with an operator.

[AND] - and

[OR] - or

[LT] – less than

[GT] – greater than

[GEQ] – greater or equal than

[LEQ] – less or equal than

[EQ] – equal

[NEQ] – not equal

[INR](*<value>*, *<bottom value>*, *<top_value>*) - returns 1 if value is in range between bottom and top values, 0 otherwise

[IN](*<value>*,*<value_1>*,*<value_2>*,..... *<value_2>*) - returns 1 if value is one from the list of values, 0 otherwise

EXAMPLE:

[IF](*<*ANYSEL(Color:Blue)*>*){*<*SELATV(Level:low)*>*}[ENDIF]

Means: If selected color is blue then value of attribute Level is set to Low.



Tags for Rules and Triggers Product Model Object Tags	
Tag	Description
<*anyssel(standard attribute name: standard attribute value)*>	Parameters: attributes with their values, separated by a colon. More than one Attribute/Value pair can be entered. Every pair is separated from other pairs by a comma. Result: If any value from the entered list is selected the result is TRUE, otherwise it is FALSE
<*anysselatt(standard attribute name)*>	Parameters: attribute names. More than one attribute can be selected. Separate them by commas. Result: If any value from the specified attributes is selected the result is TRUE, otherwise it is FALSE
<*allssel(standard attribute name: standard attribute value)*>	Parameters: attributes with their values, separated by a colon. More than one Attribute/Value pair can be entered. Every pair is separated from other pairs by a comma. Result: If all of the specified values are selected the result is TRUE, otherwise it is FALSE
<*allsselatt(standard attribute name)*>	Parameters: attribute names. More than one Attribute can be entered. Each separated from the other a comma. Result: If every specified attribute has its selection the result is TRUE, otherwise it is FALSE
<*allowatt(standard attribute name)*>	Parameters: attribute names. More than one Attribute can be entered. Each separated from the other a comma. Result: Allows all values from all specified attributes.
<*allowatv(standard attribute name: standard attribute value)*>	Parameters: attributes with their values, separated by colon. More than one Attribute/Value pair can be entered. Every pair is separated from other pairs by a comma. Result: Allows all values from the entered list



<p><*assignff(attribute name: value)*></p>	<p>Parameters: the name of the FreeFormAttribute, and the value to be assigned to the attribute, separated by colon. More than one FreeFormAttribute/FreeInputValue pairs can be entered. Every pair is separated from other pairs by a comma. Result: Every FreeInputValue is assigned to appropriate FreeFormAttribute. If FreeInputValue contains function, the function is executed before assignment. For example: <*assignff(attribute_name:<*catcode(other_attribute)*>)*> gives as the result the CatCode string from the "other_attribute", assigned to FreeForm field in the "attribute_name" attribute. <*assignff(attrib_name:Webcom)*> assigns the string "Webcom" to the "attrib_name" attribute. Non-FreeForm attributes cannot accept the assigned values.</p>
<p><*assignnpc(attribute name: value)*></p>	<p>Parameters: attributes with their prices to be assigned, separated by colon. More than one Attribute/Price pair can be entered. Every pair is separated from other pairs by a comma. Result: The same as previous, except that it assigns "standard attribute value" to the Price field of the "standard attribute name", and it is not limited to FreeForm attributes.</p>
<p><*disallowatv(standard attribute name: standard attribute value)*></p>	<p>Parameters: attributes with their values, separated by colon. More than one Attribute/Value pair can be entered. Every pair is separated from other pairs by a comma. Result: Disallows every specified value in every attribute in the list.</p>
<p><*disallowatt(standard attribute name)*></p>	<p>Parameters: attribute names. More than one Attribute can be entered. Each separated from other by a comma. Result: If condition is TRUE, all listed attributes are disallowed. If condition, during some of the following steps becomes FALSE, the attributes are NOT re-allowed. Recommended here is to create complementary Rules: for every Rule made as condition - > disallowatt(attribute_list) create another as [not]condition - > allowatt(attribute_list). Additionally, make sure that every complementary Rule has its Rank higher than the Rank of the primary Rule.</p>
<p><*resetatt(standard attribute name)*></p>	<p>Parameters: attribute names. More than one Attribute can be entered. Each is separated from other by a comma. Result: Resets every value of every listed attribute.</p>



<*resetatv(standard attribute name:standard attribute value)*>	Parameters: attributes with their values, separated by colon. More than one Attribute/Value pair can be entered. Every pair is separated from other pairs by a comma. Result: Resets every listed attribute value.
<*selatv(standard attribute name: standard attribute value)*>	Parameters: attributes with their values, separated by colon. Result: Selects attribute values.

GI Tags Shopping Cart Object Tags	
Tag	Description
<*setGI(key,value)*>	Sets Global Info value in a map with key
<*GI(keyV)*>	gets Global Info from global cashe under key, evaluates to null if there is no info under key
<*GI*>	Sets Global Info value in a map
<*remGI(key)*>	Removes key
<*clearGI*>	Removes all globals
<*TESTGI(keyValue)*>	returns 1 if GI(keyValue) is defined else returns 0 purpose: it is more convinient to test for the existance of a variable before using it in calculations/conditions
<*GI(CUSTOMERID)*>	Return Customer ID
<*getAttNumVal(attribute name)*>.	Read attributes numeric value directly from the database (from the cart items table)



Word Quote Tags Word Generation Engine Tags

<p><<Q_QUOTE(COMMENT)>> <<Q_QUOTE(NUMBER)>> <<Q_QUOTE(DATE_CREATED)>> <<Q_QUOTE(TOTAL)>> <<Q_QUOTE(TOTALNET)>> <<Q_QUOTE(FREIGHT)>> <<Q_QUOTE(CURRENCY)>> <<Q_QUOTE(CURRENCY_SIGN)>> <<Q_QUOTE(TAX)>> <<Q_QUOTE(VALID_UNTIL)>> <<Q_QUOTE(BASE_TOTALNET)>> <<Q_QUOTE(REVISION)>></p> <p><<Q_SHIPTO(FIRSTNAME)>> <<Q_SHIPTO(LASTNAME)>> <<Q_SHIPTO(EMAIL)>> <<Q_SHIPTO(COMPANY)>> <<Q_SHIPTO(ADDRESS1)>> <<Q_SHIPTO(ADDRESS2)>> <<Q_SHIPTO(CITY)>> <<Q_SHIPTO(STATE)>> <<Q_SHIPTO(PROVINCE)>> <<Q_SHIPTO(ZIPCODE)>> <<Q_SHIPTO(COUNTRY)>> <<Q_SHIPTO(PHONE)>> <<Q_SHIPTO(FAX)>></p> <p><<Q_COMP(NAME)>> <<Q_COMP(EMAIL)>> <<Q_COMP(ADDRESS1)>> <<Q_COMP(ADDRESS2)>> <<Q_COMP(CITY)>> <<Q_COMP(STATE)>> <<Q_COMP(ZIPCODE)>> <<Q_COMP(COUNTRY)>> <<Q_COMP(PHONE)>> <<Q_COMP(FAX)>></p>	<p><<Q_BILLTO(FIRSTNAME)>> <<Q_BILLTO(LASTNAME)>> <<Q_BILLTO(EMAIL)>> <<Q_BILLTO(COMPANY)>> <<Q_BILLTO(ADDRESS1)>> <<Q_BILLTO(ADDRESS2)>> <<Q_BILLTO(CITY)>> <<Q_BILLTO(STATE)>> <<Q_BILLTO(PROVINCE)>> <<Q_BILLTO(ZIPCODE)>> <<Q_BILLTO(COUNTRY)>> <<Q_BILLTO(PHONE)>> <<Q_BILLTO(FAX)>> <<Q_BILLTO(CUSTOMERNUMBER)>></p> <p><<Q_USER(NAME)>> <<Q_USER(USERNAME)>> <<Q_USER(EMAIL)>> <<Q_USER(COMPANY)>> <<Q_USER(ADDRESS1)>> <<Q_USER(ADDRESS2)>> <<Q_USER(CITY)>> <<Q_USER(STATE)>> <<Q_USER(ZIPCODE)>> <<Q_USER(COUNTRY)>> <<Q_USER(PHONE)>> <<Q_USER(FAX)>></p> <p><<Q_ENDUSER(FIRSTNAME)>> <<Q_ENDUSER(LASTNAME)>> <<Q_ENDUSER(EMAIL)>> <<Q_ENDUSER(COMPANY)>> <<Q_ENDUSER(ADDRESS1)>> <<Q_ENDUSER(ADDRESS2)>> <<Q_ENDUSER(CITY)>> <<Q_ENDUSER(STATE)>> <<Q_ENDUSER(PROVINCE)>> <<Q_ENDUSER(ZIPCODE)>> <<Q_ENDUSER(COUNTRY)>> <<Q_ENDUSER(PHONE)>> <<Q_ENDUSER(FAX)>></p>
--	---



<<Q_QP(Quote property name)>>	Quote property name as defined by label field in scParamDefn
<<Q_QP_FILE(Quote property name)>>	Quote property name as defined by label field in scParamDefn, assumes returned value is a file name and inserts it (file has to be located in /quotation/user_templates/ folder
<<Q_QP_IMAGE(Quote property name)>>	Quote property name as defined by label field in scParamDefn, assumes returned value is an image name and inserts it (file has to be located in /quotation/user_templates/ folder
<<Q_FILE(filename.doc)>>	Inserts file "filename.doc" in the any place of the Word Document – including Header and Footer. filename.doc can have any tag and all tags from filename.doc will be processed after inserting and has to be located in /quotation/user_templates/ folder
<<ST_PTYPE_PRICE(Product_Type)>>	Total discount price (Desired Price times Quantity) for the product type "Product_Type"
<<ST_PTYPE_NETPRICE(Product_Type)>>	Net Price for the product type "Product_Type"
<<ST_PTYPE_LISTPRICE(Product_Type)>>	Extended Price (Quantity times Base Price), before applying a discount for the product type "Product_Type"
<<ST_USERFILE(Section)>>	Inserts User File from section. User Files has to be located into folder /user_files/
<<ST_USERFILE_IMG(Section)>>	Inserts Image from User File. User File (Image) has to be located into folder /user_files/
<<ST_GROUP_SUBTOTAL(Group_Name)>>	Inserts Subtotal form particular Group
<<ST_GROUP_LABEL(Group_Name)>>	Inserts Label of particular Group
<<ST_GROUP_DESCRIPTION(Group_Name)>>	Inserts Description of particular Group

All Word Quote Tags supports Conditions.

General form of the Conditions:

<<TAG, Conditions=Action if TAG satisfies the Condition|Action if TAG doesn't satisfy any of the Conditions>>

Available Actions:

- D - delete the line
- d - delete only the last added content
- P - insert page break after the last added content
- p - insert page break before the last added content
- B - bold the line that the content is on
- b - bold the last added content
- DR - if on <<C_..>> inside the <<CT...>> tag (table) it will delete the row that content is in



DC - delete the contents of the cell where the last added content was in
DELETE_TABLE – delete table

Condition supports wildcard "*".

For **Example**,

<<C_DESC,W*com*nc=B|>>

will bold the line that says Webcom, Inc (of course, if “Webcom Inc” is in Description field)

Inserted Word and Excel files are "pre processed".

This means that now is first created a copy of the file to be inserted, process all the tags inside it, save it and insert the processed file.

Repeatable Cart Tags

- Textual

<<C>> tag

Begin with **<<C>>**.....**<<C_END>>** - lists only Main Items

or

Begin with **<<C_LI>>**.....**<<C_END>>** - lists all Items

or

Begin with **<<C_LI_PTYPE(Product_Type)>>**.....**<<C_END>>**
- shows only Items that are of certain Product Type

or

Begin with **<<C_GROUP_Group_Name>>**
- shows only Items that are of certain group created in the Shopping Cart.
Groups can be “A”, B” “C” or “D”.

It is necessary to use <<C_STOP>> right before <<C_END>>

<<C2>> tag

<<C2>> tag can separate Main Items and Line Items:

<<MAIN>> ... **<<MAIN_END>>** - shows Main Items

< ... **<<LI_END>>** - shows Line Items

For Example:

<<C2>>

MAIN tag - for Main Items: **<<MAIN>>**

Tags for Main Items

<<C_STOP>>

<<MAIN_END>>

LI tag - for Line Items: **<**

Tags for Line Items

<<C_STOP>>



<<LI_END>>

<<C_END>>

<<C2_PTYPE(Product_Type)>> tag – works on the same way as <<C2>> tag, but shows only Items that are of certain Product Type.

For **Example**, <<C2_PTYPE(Hardware)>> will show only items that are from product type “Hardware”

<<C2_GROUP_Group_Name>> tag – works on the same way as <<C2>> tag, but shows only Items that are of certain group created in the Shopping Cart. Group can be “A”, “B”, “C” or “D”.

For **Example**, <<C2_GROUP_A>> will show only items that are from group “A” created in the Cart.

- Table

- first row has headers

- second row must have <<CT>> or <<CT_LI>> in the first column. <<CT_LI>> will list Main items and Line Items. <<CT>> will list only the Main Items.

- third row must have tags defining what values will go to in that column. Column can be blank, have a mix of characters and tags or only have tags.

- there must be a fourth row. It can be empty or have total or whatever else, but it has to be there or else the code will fail

- Table with Tabs

- first row has headers

- second row must have <<CT_LI_TAB>> in the first column.

- third row must have <<C_TAB_NAME>> - display Tab's Name

- fourth row must have tags defining what values will go to in that column. Column can be blank, have a mix of characters and tags or only have tags.

- fifth row must have <<C_TAB_SUM>>. First Column will have a sum for that tab.

- sixth row - It can be empty or have total or whatever else, but it has to be there or else the code will fail

<<CT2>>Table

<<CT2>>Tables works on the same way as “normal” Tables but it can separate Main Items and Line Items.

- first row has headers

- second row must have <<CT2>> in the first column.

- third row has tags for Main Items (can be Empty if we don't want to show Main Items)

- fourth row has tags For Line Items (can be Empty if we don't want to show Line Items)

- there must be a fifth row. It can be empty or have total or whatever else, but it has to be there or else the code will fail

<<CT2_GROUP_Group_Name>> tag can be placed in the second row instead of <<CT2>> tag and shows only Items that are of certain group created in the Shopping Cart. Group can be “A”, “B”, “C” or “D”.

For **Example**, <<CT2_GROUP_B>> will show only items that are from group “B” created in the cart.



<<C_BOM(Selection Chart Name)>>	This tag will call BOM for each item in the cart and get results for Selection chart specified. Results will be parsed using the product object and if there is a <<FILE(fname)>> or <<IMAGE(imagename)>> tag that file or image will be inserted in place of that BOM tag else BOM result text will be inserted. For now, files have to be in /quotation/templates/ folder and images have to be in /images/ folder.
<<C_ID>>	Item ID
<<C_QTY>>	Item Quantity
<<C_PTYPE>>	Item Product Type
<<C_PATTR(<*PRODUCTINFO(TYPE)*>>>	Item Product Type, but only for Main Items
<<C_DESC>>	Item Description
<<C_PNUM>>	Part Number, also known as Catalog Code, Catalog Number, SKU, etc
<<C_DESIRED>>	Unit price after applying a discount
<<C_BASE>>	Unit price before applying a discount
<<C_LIST>>	Extended Price (Quantity times Base Price), before applying a discount
<<C_DISCOUNT>>	Discount Percentage
<<C_USER_DESC>>	Description for Main Items, added by User
<<C_EMPTY>>	Does nothing just to be used with conditions – for Example: <<C>> <<C_ID>> <<C_PNUM>> <<C_EMPTY, P>> <<C_STOP>> <<C_END>> This Code Shows every Main Item on the New Page (inserts Page Break in between the Main Items)
<<C_PRICE>>	Total discount price (Desired Price times Quantity)
<<C_PIMAGE>>	Insert product image as defined in product administration
<<C_PNAME>>	Inserts product name
<<C_PATTR(...)>>	it can call any standard CPQ tag (i.e. <*Value(...)*>
<<C_PATTR_FILE(...)>>	it can call any standard CPQ tag (i.e. <*Value(...)*>, assumes returned value is a file name and inserts it (file has to be located in /quotation/user_templates/ folder)



<<C_PATTR_IMAGE(...)>>	it can call any standard CPQ tag (i.e. <*<Value(...)*>, assumes returned value is an image name and inserts it (file has to be located in /quotation/user_templates/ folder)
<<C_PATTR_EXCEL(...)>>	this will insert an Excel file based on the value of the attribute (hint. since (...) is really a parsable string, you can just enter a file name here and it will try to insert the file - the file should be in /quotation//user_templates/ folder). Excel Document inserted using C_PATTR_EXCEL can have Tags too. All regular tags should be supported (no conditional processing on them though) and for Items only use <<CT_LI(n)>> where n is number of columns.
<<MAIN>>.....<<MAIN_END>>	See <<C2>> tag
<<SEPARATE>>	If <<C>> Tag is before the Table, it is possible to be inserted into the Table (instead of before the table). To avoid this, use this Tag.
[CONDITION]	If the Condition is True, then executes the Tags inside [CONDITION,...], [CONDITION_END] . Example: [CONDITION,<<.....>>=123 456 789] If expression inside "<<... >>" is "123" or "456" or "789" this tags will be processed. (Tags goes here) [CONDITION_END] Or [CONDITION, <<.....>>!=123 456 789] If expression inside "<<... >>" is NOT "123" or "456" or "789" this tags will be processed (Tags goes here) [CONDITION_END]
<<ST_GI(...)>>	Insert GI value
<<ST_GI_FILE(...)>>	find GI value, assumes returned value is a file name and inserts it (file has to be located in /quotation/user_templates/ folder)
<<ST_GI_IMAGE(...)>>	find GI value, assumes returned value is an image name and inserts it (file has to be located in /quotation/user_templates/ folder)
<<ST_LOGO()>>	Finds a logo for current users company and inserts the image
<<ST_GI(foo),xxx=D P>>	If value of foo is "xxx" it will delete that whole line, else for any other values it will insert a page break after the value



<<ST_GI(foo),xxx=D >>	If value of foo is "xxx" it deletes the whole line, else doesn't do anything (keep in mind that "]" is required in the end)
<<ST_GI(foo),=D >>	If foo is blank/doesn't have a value, it will delete the line, else it doesn't do anything
<<ST_GI(),xxx=D P>>	Inserts a Page Break
<<ST_USERFILE_IMG(section_name)>>	Inserts the image chosen in the upload Form, image is located in the folder /quotation/user_files/userid/
<<ST_USERFILE(section_name)>>	Inserts the file chosen in the upload Form, file is located in the folder /quotation/user_files/userid/
<<ST_PTYPE_DISCOUNT(Product Type)>>	Displays the Discount of certain Product Type
[FORMAT_NUMBER(any tag that returns a number,currencyFlag,decimalPoints,leadingZero,includeCommas)]	<p>This will format any number and: If currencyFlag is Y it will prefix it with the currency sign decimalPoints indicates how many decimal points to have (number will be rounded - so 3.789 will be 3.79 if decimalPoints is 2) If leadingZero is Y, .7 will be prefixed with 0 so it will be 0.7 If includeCommas is Y it will show commas - 3,123.00</p> <p>Examples: [FORMAT_NUMBER(<<C_PATTR(<*Value(Next Annual Renewal Maintenance)*>>),Y,2,Y,Y)] - input is 5678.91 result will be \$5,678.91 for example</p> [FORMAT_NUMBER(<<C_DISCOUNT>>,N,2,Y,Y)] - input is 9.123456789, result will be 9.12

XLS Templates

XLS Templates Works on the same way as Word Templates but with some limitations:

- Only Columns A - H can be used for tags
- In Tables, instead of <<CT_LI>> we have to use <<CT_LI(n)>>, where "n" is the number of Columns in this table.
- Output of XLS Template is also XLS file
- First XLS tab has to be "Quote"
- CONDITION tag will not work in XLS Templates



Consolidate Tags

Consolidate functionality adds ability to group the items based on a common attribute or value.

Consolidate is a part of <<C2>> tag and should always be enclosed within <<C2>>....<<C_END>>

Typical consolidate tag implementation looks like this:

```
<<C2>>
[CONSOLIDATE(<<C_PATTR(<*Value(Color)*>>>)]
Items color: [UNIQUE(<<C_PATTR(<*Value(Color)*>>>)]
Material: [APPEND(<<C_PATTR(<*Value(Material)*>>>)]
Items shape is: [UNIQUE(<<C_PATTR(<*Value(Shape)*>>>)]
[STOP(<<C_STOP>>)]
[CONSOLIDATE_END]
<<C_END>>
```

As an example lets assume to have 5 items in the quote, each having the following attributes:

Item 1 – Color: Blue;	Material:Plastic;	Shape:Circle
Item 2 – Color: Blue;	Material:Plastic;	Shape:Square
Item 3 – Color: Red;	Material:Metal;	Shape:Circle
Item 4 – Color: Blue;	Material:Glass;	Shape:Square
Item 5 – Color: Red;	Material:Metal;	Shape:Triangle

The beginning of consolidate starts with **[CONSOLIDATE(value to consolidate on)]** and ends with **[CONSOLIDATE_END]** . As a part of pre processing, application will first go through all items in the quote, identify the value enclosed within (...) and group the items that have the same value. For example, **[CONSOLIDATE(<<C_PATTR(<*Value(Color)*>>>)]** will group items that have Blue color together and items that have Red color together, etc. It will process the content between **CONSOLIDATE** and **CONSOLIDATE_END** once for each group.

There are two tags specific to consolidation:

[APPEND(...)] – it finds a value for ... and inserts it for each item in the group, separated by “,”. For example, **[APPEND(<<C_PATTR(<*Value(Material)*>>>)]** will list Plastic,Plastic,Metal for each items that is Blue.

[UNIQUE(...)] – it displays a single value that is unique to all items in the group (otherwise it will display the value of the first item in the group). It only makes sense to use the same value as is used in **CONSOLIDATE** statement, otherwise the uniqueness cannot be guaranteed. For example, **[UNIQUE(<<C_PATTR(<*Value(Color)*>>>)]** will return value Blue.

[UNIQUE(<<C_PATTR(<*Value(Shape)*>>>)] should return Circle, but the result is not guaranteed because Shape is not unique for all items within the group.

[STOP(<<C_STOP>>)] – this is a static identifier that signals where the processing should stop and should be used as is and always be in front of **[CONSOLIDATE_END]**.



611 N. Broadway Suite 102
Milwaukee, WI 53202
sales@webcominc.com
(414) 273-4442
www.webcominc.com

The output should look like this:

Items color: Blue
Material: Plastic,Plastic,Glass
Items shape is: Circle

Items color: Red
Material: Metal,Metal
Items shape is: Circle



Word Quote Generation Examples

EXAMPLE 1:

```
<<C>>
<<C_PATTR(<*Value(Software Description)*>)>>
<<C_END>>
```

This code will write right exactly Software Descriptions as we have items in the cart – for each item one Software Description – Software Description is Attribute and contain descriptions.

Output can look like this:

Description of Item 1

Description of Item 2

Description of Item 3

Description of Item 4

Description of Item 5

Description of Item X – in real situation, this is description of item (“128 USB Flash Memory Card”, for Example).

EXAMPLE 2:

Table 1 below will produce the output as shown on Figure 43-8

Item	Part Number	Description	Quantity	Price	Extended Amount
<<C_T_LI>>					
<<C_ID>>	<<C_PNUM>>	<<C_DESC>>	<<C_QTY>>	<<C_DESIRE>>	<<C_PRICE>>
					<<Q_QUOTE(TOTAL)>>

Table 1 Example 2



Item	Part Number	Description	Quantity	Price	Extended Amount
1	DT-A-C-W	Business Desktop. 0.5 GB PC3200 (400 MHz) memory, 200GB Ultra HDD, Amtel 9 3GHz Processor, with 96x/64x/96x CD-RW Drive and . Windows Home 2003 operating system.	1	\$1,210.00	\$1,210.00
2	DT-3-C-W	Business Desktop. 2x512 MB Dual Channel PC3200 memory, 400GB Ultra HDD, PMD 3 3GHz Processor, with 96x/64x/96x CD-RW Drive and . Windows Home 2003 operating system.	1	\$1,420.00	\$1,420.00
3	DT-8-D-W	Business Desktop. 2x512 MB Dual Channel PC3200 memory, 800GB Ultra HDD, PMD 8 8GHz Processor, with 64x DVD-ROM Drive and . Windows Home 2003 operating system.	1	\$1,990.00	\$1,990.00
					\$4,630.00

Table 2 Example 2

EXAMPLE 3:

Use the following tags in order to insert a Page Break after every included file:

```
<<C_LI>>
Detailed Documentation for Item <<C_ID>>,a1234=|P>>
<<C_PATTR_FILE(<*Value(Processor_Docs_d)*>),a1234=|P>>
<<C_PATTR_FILE(<*Value(memory_docs_d)*>),a1234=|P>>
<<C_STOP>>
<<C_END>>
```

For example, the condition checked is “a1234”. When there is no file named “a1234” the condition will evaluate to False and action P (in this case insert Page Break) will be always executed.

EXAMPLE 4:

Use the following Example to avoid inserting blank pages in a document:

```
<<C_LI>>
[CONDITION,<<C_PATTR(<*Value(HDD_Docs_d)*>)>>!] ]
<<C_PATTR_FILE(<*Value(HDD_Docs_d)*>),a1234=|P>>
[CONDITION_END]
<<C_STOP>>
<<C_END>>
```

For example, use a CONDITION tag: IF there is no Value for HDD_Docs_d Attribute then this file will not be inserted, because Condition is False and all tags (in this case tag <<C_PATTR_FILE(<*Value(HDD_Docs_d)*>),a1234=|P>>) will not be executed and blank page



611 N. Broadway Suite 102
Milwaukee, WI 53202
sales@webcominc.com
(414) 273-4442
www.webcominc.com

will not be inserted.

EXAMPLE 5:

Use the following tag to display only items from a certain Product type.

```
<<C2_PTYPE(Hardware)>>  
...Tags for this product type...  
<<C_STOP>>  
<<C_END>>
```

For example, only Items from Product Type "Hardware" will be displayed.