



## Functions and Expressions in AGROBASE Generation II® July, 2010



AGROBASE Generation II® has a powerful set of functions and expressions which can filter data, make data transformations, and help generate complex reports. This document shows the functions and expressions useful within Generation II, along with examples. Please see the online tutorials from our Learning Centre at [www.agronomix.com](http://www.agronomix.com), entitled "Data Transformations I and II", which will give the context for the data manipulations shown below. One need not resort to Excel®, for example, to complete data transformations – much can be done completely within Generation II.

### Text or Character Functions

**"x"\$ Y** – Simple search function. Is "x" found in Y? If so, return .T., otherwise .F.

**Example:** "AC" \$ "AC Baton" returns .T., whereas "AC" \$ "Accolade/Comanche" returns .F. Unless another function is used to put all to upper case, like "AC" \$ UPPER("Accolade/Comanche"), which returns .T., the search will fail. This simple function is used often.

**ALLTRIM(x)** – trim all blanks, whether before or after the text field.

This function is very useful when adding other text or converted numeric data to a text field that has trailing blanks. If you don't use this function, you won't see the new values simply by using "+" to concatenate. You will also need this when designing barcodes with text data, to prevent overly long barcodes that simply barcode a lot of blanks.

#### Examples:

Adding a selection number to a text field:  
`ALLTRIM(SELECTION)+STR(SEL,3)`

Changing a Pedigree by adding a location code:  
`ALLTRIM(PEDIGREE)+"-"+LOCCODE`

Removing blanks from an expression which will be added to text:  
`"VAR-"+ALLTRIM(STR(ENTRY,3))` would create "VAR-1", "VAR-2", etc.

**AT(x, y)** – find the first occurrence of "x" in text field "y", and return the character position.

**Example:** AT("Baton",NAME) returns 4 when NAME is "AC Baton", returns 0 if NAME is "AC BATON". The function is case sensitive, and useful to replace text specifically somewhere in a text string.

You could also use this as a search tool, where a parental name, for example, should be found in a pedigree. So, AT("COKER",PEDIGREE)>0, would show all records where COKER is in a pedigree. Or, AT("COKER",PEDIGREE)=2 would search for all records where COKER is at least twice in the PEDIGREE.

**AT(x, y, i)** – find the  $i^{\text{th}}$  occurrence of “x” in text field “y”, and return the character position. If the  $i^{\text{th}}$  occurrence is not found, returns 0. This is a more complex version of the previous expression.

**Example:** AT(“Baton”,PEDIGREE,2) returns 23 when PEDIGREE is “AC Baton/Coker-62//AC Baton/Araphoe”. This function is useful if you plan to replace a specific occurrence of a parent found in a backcross or a 3 or 4-way cross – perhaps the second crossing had used the wrong parent.

**ATC(x, y, i)** – find the  $i^{\text{th}}$  occurrence of “x” in text field “y”, and return the character position, as above, but with *no regard for case sensitivity*.

**Example:** AT(“Baton”,PEDIGREE,2) returns 23 when PEDIGREE is “AC BATON/Coker-62//AC BATON/Araphoe”.

**CHR(i)** – return the character associated with the ASCII code sequence from 0 to 255.

**Example:** You want to include a Greek character, or a character from another language with diacritical marks. CHR(230) returns  $\mu$ . You would need to become familiar with the range of ASCII codes. Perhaps you want to add a certain symbol in a cross or pedigree designation – this is the best way to add that character.

**CHRTRAN(x, y, z)** – character translation -- replace or translate the part of x with z when y is found in x.

**Example:** In PEDIGREES where “Coker21” is found, replace with “Coker-21”. CHRTRAN(“Arigans/Merans//Coker21”,“Coker21”,“Coker-21”) would give you the expression in “Arigans/Merans//Coker-21”. This could also be used for changing a group of names, where part of a name’s spelling is incorrect.

**GETWORDCOUNT(x)** – calculates the number of words in a character string. This would also be a good way to determine how many spaces there are in a text like pedigree or selection.

**Example:** GETWORDCOUNT(“COKER BEZO ARIMA”) results in “3”.

**LEFT(x, i)** – return the  $i^{\text{th}}$  leftmost number of characters from “x”, a character field or expression. This is a most useful function when you need to “chop” or strip off only a certain number of characters from the left side of a character field or expression.

**Example:** LEFT(“AAA-BB”,4) returns “AAA-”.

**LEN(x)** – returns the length of the character expression.

**Example:** You want to reduce the size of a character field, like COMMENTS, but don’t want to lose any data. You could make a filter to see if you have a larger comment than, say, 50 characters: LEN(ALLTRIM(COMMENTS))>50. If no records appear in that logical expression, then trim down the field size.

**LIKE(x, y)** – compare if “x” is the same as “y”, character for character, but allowing for possible wildcards like “\*” or “?”. One would use “?” as a wildcard if “\*” is in the text. This would be useful in more complex name or pedigree or selection comparisons or filtering. Instead of an *exact* comparison or match, you are looking for an *approximate*.

**Example:** To compare two text strings with several characters that may or may not match. A string comparison such as LIKE(“ABC\*\*FG\*J\*33”,“ABCDEFGHJ233”) would return .T., whereas LIKE(“ABC\*\*FG\*J\*34”,“ABCDEFGHK233”) would return .F.

**LTRIM(x)** – trim only any leading blanks, more specific than ALLTRIM().

This function is useful when you want to re-align text in a character field to the far left, maybe after a transformation that had unintended results.

**Example:** LTRIM(" AAABB ") returns "AAABB".

**MAX(x,y,z)** – returns the maximum, according to the ASCII sorting sequence, of 2 or more character fields or values. This is also a numeric function, as below.

**Example:** MAX("ARAP","BANJO","ZOROTOS") would return "ZOROTOS".

**MIN(x,y,z)** – returns the minimum, according to the ASCII sorting sequence, of 2 or more character fields or values. This is also a numeric function, as below.

**Example:** MAX("ARAP","BANJO","ZOROTOS") would return "ARAP".

**Example:** MAX("ARAP","ARAC","ZOROTOS") would return "ARAC".

**OCCURS(x,y)** – return the number of occurrences of the text 'x' in the text or character field 'y'.

This function scans character (text) data "y" for the number of occurrences of a text constant, "x". This function is much more powerful than the simple "\$" function, like "Coker" \$ PEDIGREE, which just tests for the occurrence of text in a string, and thus is equal to OCCURS('Coker',PEDIGREE)>0.

**Examples:** You want to see how often the variety name 'Coker' appears in the pedigree of a group of varieties. You would use: OCCURS('Coker',PEDIGREE). For the pedigree "Arapaho/Coker//Yuma 51/Coker", the function would return '2'. This result could then be stored in a numeric field, and you could sort that field as well to group occurrences for selection.

You want to determine how many varieties have at least one backcross by looking for "^" in the pedigree via OCCURS("^",PEDIGREE). This assumes the back-cross symbol is "^", it could be "\*" or another symbol.

How many varieties have one backcross AND Coker at least twice: OCCURS("^",PEDIGREE)=1 AND OCCURS('Coker',PEDIGREE)>1. Note: this will require a Logical field if you want to store the result.

**PROPER(x)** – put in "proper" case any text, like you would spell a person's last name.

**Example:** PROPER("ARAPAHOE") returns "Arapahoe", as does Proper("Arapahoe"). This could be useful in renaming varieties or pedigrees when incorrectly capitalized or imported under different case sensitive names. A real timesaver! Note: When importing names, for example, you might want to use the PROPER() function first on the file as an external dbase file so as to reduce any import errors.

**REPLICATE(x, i)** – replicate, or repeat, character string "x" i times.

**Example:** REPLICATE("\*\*\*\*",5) results in "\*\*\*\*\*". This could be true for a numeric field as well, like REPLICATE("\*\*\*\*",FN) where based on the FN level, there would be "\*\*\*\*" added or imbedded in a character field.

**RIGHT(x, i)** – return the ith right-most number of characters from "x", a character field or expression. This function is especially useful since you don't have to determine the length of a text field, or find the end of the text.

**Example:** RIGHT("AAA-BB",3) returns "-BB". If you use RIGHT() with a field that has trailing blanks, you must trim off the blanks first, otherwise you might just return blanks. Therefore, use RIGHT(ALLTRIM(PEDIGREE),4), for example. This might be especially relevant when building a string for barcoding.

**RTRIM(x)** – trim only any trailing blanks, on the ‘right-side’, more specific than ALLTRIM(). This could be useful in reports where you want to pad some blanks first, but have no trailing banks.

Example: RTRIM(“ Morells ”) results in “ Morells”.

**SOUNDEX(x)** – alphanumeric sound equivalent of a character field. A possible use for simple encryption, where you want variety names to appear “scrambled” in a listing or field book, to hide their true identity.

Example: SOUNDEX(“AC BATON”) results in “A213”.

**SPACE(i)** – create a text expression consisting of just some blanks, or spaces. Useful when exact spacing in a text string is important.

Example: SPACE(5) will create “ ”.

**STR(n, i, j )** - create a text expression from a numeric value, along with the given size and number of decimals.

This is required when you want to add ‘numeric’ data to a character field or expression. For example, entry numbers for dummy variety names or a selection number to a selection field, or when renumbering population names. *This is a very useful function, and often needed for barcodes.*

Example: “VAR-“+STR(ENTRY,3) – you want to import an experiment from an external source, and the entry names are not present, although you might get them later. If you open as an external DBF file, the above expression would create “VAR- 1”, “VAR- 2”, etc.

Or, you might want to create an experiment with “dummy entries” since you don’t know yet what will be planted therein

**STRTRAN(x, y, z)** – replace or translate each occurrence of the text ‘y’ with the text ‘z’ whenever found in ‘x’. Effectively, a string translation. Instead of just searching, now you can *replace values* in a text field. A very powerful function, especially to correct naming errors from imported text data.

#### Examples:

Several hundred lines were imported with incorrect names or pedigrees, or you have generated populations and you want to alter the name or pedigrees or selection field.

STRTRAN(SELECTION,“LT”,“LE”) – this would replace “LT” for the Lethbridge location, where the user had wanted “LE” instead.

STRTRAN(SELECTION,“LT”) – this would replace “LT” for the Lethbridge location with a null – effectively removing it from wherever it is found in the text. The expression, STRTRAN(SELECTION,“LT”,“”) would give the same result.

STRTRAN(NAME, “IA”,“YA”) – replacement for incorrect names like “BEZOSTAIA” which should be “BEZOSTAYA”.

When the replacement should be carried out for a specific number of occurrences, you can use the expanded form of STRTRAN as below.

**STRTRAN(x, y, z, i, j)** – replace or translate each occurrence of the text ‘y’ with the text ‘z’ whenever found in ‘x’, but beginning from the i-th occurrence and for j-th occurrences thereafter. If “y” is not found in “x”, then simply return “x” without any changes.

Example: STRTRAN(PEDIGREE,“Veery”,“Vee”,2,3) – replace the second to fifth, if found, occurrences of Veery with Vee.

**STUFF(x, i, j, y)** – replace a specified number of characters from a specific position in the text field, “x”, with the contents of “y”. No searching here, just a direct replacement. Maybe the wrong location code was added to a selection field, and needed to be changed.

**Example:** STUFF(“AAA-BBB”,5,2,“CC”) would result in “AAA-CCB”.

**SUBSTR(x, i, j,)** – take out a substring from a specified position in a text field.

**Example:** SUBSTR(SELECTION, 7, 8) - after several rounds of selections, a breeder wants to extract the text for the last two selections which have 3 character positions each. Unlike STRTRAN, the original field is left unchanged. If SELECTION is “WX2006-12T-04L”, the result is “-12T-04L”. With SUBSTR(SELECTION, 7, 4), the result is “-12T”.

**Example:** SUBSTR(STR(RECNO()+1000,6),4,3) would give “456” when record number 2456 is encountered.

## Numeric Functions

**ABS(j)** – returns the absolute value of a number.

**CEILING(i)** – returns the next highest integer value. The opposite is FLOOR().

**Example:** CEILING(12.1) gives 13. CEILING(-12.1) gives -12.

**FLOOR(i)** – returns the next lowest integer value.

**Example:** FLOOR(12.1) gives 12. FLOOR(-12.1) gives -13.

**INT(j)** – returns the integer portion of a number. This is useful to determine if a number has a decimal fraction or not, and can be readily used in an expression to determine every second or fifth, say, record in a report and then print a thicker line or change color, for example.

**Example:** IIF(RECNO()/5=INT(RECNO()/5) would return a .T. for every multiple of 5.

**MAX(i,j,k,l,m,n)** – returns the maximum value from a series of numbers as values or fields.

**Example:** MAX(RSC1,RSC2, RSC3,RSC4) – rust data was taken from 4 different environments for the same population. The breeder wants to store the maximum score in another field, to indicate the score under maximum disease pressure.

**MIN(i,j,k,l,m,n)** – returns the minimum value from a series of numbers as values or fields.

**Example:** MIN(YLD1,YLD2, YLD3,YLD4) – yield data was taken from 4 different environments for the same hybrids. The breeder wants to store the minimum value in another field, to indicate the lowest yield in the most stressed environments.

**MOD(i/j)** – Returns the modulus of a division. That is, any remainder when not an exact division resulting in an integer.

**Example:** MOD(RECNO()/5) returns 2, 4, 6, 8 for 1 top 4, but 0 for 5. This could be another logical alternative to using INT() and RECNO() to print every second or fifth page, for example .

**RAND( )** – returns random deviates from the (0,1) uniform pseudo-random distribution. No parameter is needed, the values range from 0 to 1. This is helpful for generating “random” numbers, but see the NORML(I,j) AGROBASE function below.

**Example:** RAND()\*100 generates random number from 0 to 100.

**RECNO( )** – returns a record number as numeric value. No parameter needed. This function returns the record number for any open table for browsing, such as an experiment, trial, or group of experiments. The function returns integers from 1 to the maximum number of records currently open. This function would also work with an external, open dBase or FoxPro file. This function does not require an argument between the brackets, the '( )' remain empty and simply identify it as a FoxPro function instead of a field or variable called 'RECNO'. When an argument is included, it would be an integer for the work area open – but this is irrelevant for Gen. II users.

**Example:** You want to enter an alternate set of plot numbers, starting from 1001 upwards. Since the first record returns 1, adding 1000 results in 1001. You would use: RECNO()+1000.

Those values could be stored in a new calculated field, or you can replace the existing plot numbers for an experiment or nursery through **Experiment > Modify Experiment/Nursery > Plot Renumbering > Transformation.**

**RECCOUNT()** – returns the maximum record count as a numeric value.

This function is similar to RECNO(), except that it returns the maximum number of records available for any open table for browsing, such as an experiment, trial, or group of experiments, or for an external dBase or FoxPro file.

**ROUND(i, j)** – rounds up a numeric field to the nearest number of j digits. This gives certain control over rounding and control of precision in reporting.

**Example:** Round(TWT,1) for 45.39 results in “45.4”.

**SECONDS()** – seconds from midnight, an internal FoxPro system function. This could be a way to computer elapsed time in calculations.

**VAL(x)** – returns the numeric value for a character string. This is important when you want to convert numbers stored as text, or found in a text string, to numeric type data. VAL() will ignore leading blanks and zeroes, and will start converting until a non-integer value is found.

**Example:** VAL(“23.5”) results in the number, 23.5. VAL(“23TY”) results in the number 23. VAL(“A23.5”) results in 0.

**Others numerical expressions are:** SQRT(), SIN(), ARCSIN(), COS(), ARCOS(), LOG(), LOG10(), EXP(), TAN(), ATAN2(), PI(), DTOR() – degrees to radians, RTOD() – radians to degrees.

## Date Functions

Date expressions are used to show dates on a report, or calculate number of days elapsed, for example. Important date expressions are:

**CTOD(char)** – converts a character expression to a date expression.

**Example:** Given a dd/mm/yy format, an expression like CTOD(“11/29/2006”) would generate the corresponding date expression and could then be used in other date expressions. This is also useful when you have an entire field or column of dates stored as character in date text format, and must be converted to use in date expression calculations.

**DATE()** – returns the current system date as a date expression, often used in other date expression. This function is used without any arguments.

**DMY(date)** – this will generate character text for s string in day, month and year format.

**Example:** In reports to generate the current date, use: DMY(DATE()), to give, for example “29 November 2006”.

**DAY(date)** – returns the numeric day for the date expression. For today’s days in the month, use: DAY(ATE()). If the current date is 29 November 2006, the expression returns “29” as a numeric.

**Example:** DAY(CTOD(‘29/11/2006’)), wherein CTOD generates a valid date expression.

**DOW(date)** – returns the day of week as a numeric value.

**CDOW(date)** – returns the day of week as text. So if the current date is 30 November, 2006, CDOW(ATE()) returns “Thursday”.

**MONTH(date)** – returns the numeric month for a date. If the current date is 29 November 2006, the expression returns “11” as a numeric.

**YEAR(date)** – returns the numeric year for a date. If the current date is 29 November 2006, the expression returns “2006” as a numeric.

### Special Note: Computing Days Elapsed

This can be done via two date expressions, which when subtracted, give a numeric value.

**Example:** Emergence date was May 15, 2006. Then to compute the days elapsed from a stored date of “09/11/2006” in MDATE (maturity date), use the expression: MDATE – CTOD(‘06/15/2006’).

You can also do such computations using a SYS function. SYS(11,ATE()) would return the character string “2454070” for the date of November 30, 2006. To make this a numeric value, use VAL(SYS(11,ATE())). Two such expressions subtracted also would give the number of days elapsed.

## Logical Functions

Logical functions may be used in reports, searches, data display, and in many places. Here are some examples, with explanations of the functions following.

<u>Expression</u>	<u>Explanation</u>
ENTRY<6 AND BLOCK=1 !(ENTRY<6 BLOC=1)	Entries 1 through 5, block one only. All entries except those in block 1, and entry numbers 1 through 5. The “!”, which is equal to NOT, simply reverses the condition.
‘VEE’ \$ NAME ‘VEE’ \$ NAME AND !‘CKR’ \$NAME ‘VEE’ \$ NAME AND !‘CMX’ PEDIGREE	All varieties with ‘VEE’ in the NAME field. All varieties with ‘VEE’ in the NAME field, but not ‘CKR’ in the NAME field. All varieties with ‘VEE’ in the NAME field but not with ‘CMX’ in the PEDIGREE field.
‘VEE’ \$ UPPER(NAME)	All varieties with ‘VEE’ found in the NAME field, taken to upper case via the UPPER function. UPPER converts character text from any case to upper case.
‘vee’ \$ LOWER(NAME) ‘Vee’ \$PROPER(NAME)	Same as the previous example, but all with lower case. All varieties with ‘Vee’ found in NAME, but in the proper text with a first upper case character, and the remaining in lower case, like you would spell a person’s last name.
ENTRY=1 OR ENTRY=3 RECNO()/10=INT(RECNO()/10)	All varieties with entry numbers 1 or 3. Every tenth record, whereby a record number (returned by RECNO()) is equal to the integer portion (as returned by the INT function), that is an exact multiple of 10. This function is helpful if browsing every tenth record in a file, like when every tenth record is a check plot.
BETWEEN(ENTRY,3,19)	All varieties with entry numbers in ENTRY between 3 and 19. This is an easier way to write the expression ENTRY >2 AND ENTRY <20
INLIST(ENTRY,11,45,6,34,22,67)	All records with ENTRY equal to either 11, 45, 6, 34, 22, or 67. The list of numbers can be as long as fits in the expression builder.
INLIST(LOCCODE,‘BN’,‘SK’,‘LT’) ISDIGIT(SUBSTR(ID,2,1))	All records where LOCCODE (location code) is equal to BN, or SK, or LT. All records where the second character, (width of 1, so 2,1) as returned by the digit function, is a digit – that is, numeric.

ISUPPER(SUBSTR(NAME,3,1))	All records where the third character in the name field is upper case.
ISLOWER(SUBSTR(NAME,3,1))	All records where the third character in the name field is lower case
LEFT(NAME,5)='COKER'	All records with the first five characters in NAME equal to "COKER".
!(INLIST(NAME,"AC Avonlea","AC Baton"))	Excluding two varieties from a series of treatment names in a table:
IIF(YIELD=-9,-9,YIELD/CALC("A","YIELD","checks>0")*100)	Calculating the percent of checks unless a missing value is encountered.
FINDPLOT("PLOT","ENTRY","-",".F.)	Finding all plot numbers, separated with "-" for entries of an experiment, leaving out the plot number of the active record.

**==** - an exact match, sometimes required when equating possibly empty text fields that should be then equal, but do not evaluate as such.

**!** or **.NOT**. The negator to reverse any logical condition.

### **AND, NOT, OR – logical connectors.**

**Example:** YIELD>20 AND MAT<90 OR SRUST<2. To remove ambiguity, or to allow for even more complex expressions, use parentheses. (YIELD>20 AND MAT<90) OR SRUST<2.

**BETWEEN(x, i, j)** Includes all records that have a numeric value for field "x" between the range of i and j.

**Example:** BETWEEN(ENTRY, 3, 19) would select all varieties with entry numbers in ENTRY between 3 and 19. This is an easier way than writing the expression ENTRY >2 AND ENTRY <20. You could also have "BETWEEN(ENTRY,3, 19) OR BETWEEN(ENTRY,14,17)" to select varieties in those two entry number ranges. More complex still would be to select all varieties except those, which could easily be done with the statement "!(BETWEEN(ENTRY,3, 19) OR BETWEEN(ENTRY,14,17))". Notice the extra set of brackets to ensure the correct result.

**Example:** To view only the varieties that have NAMES starting with "A" to "P", or from "AW" to "DS". You can use BETWEEN here as well to select records across the ASCII sorting character range. You would use "BETWEEN(NAME,"A","P")".

**EMPTY(x)** – returns .T. if a character field is empty, or just equal to blanks. This is simpler than seeing if ALLTRIM(x)="".

**EMPTY(i)** – returns .T. if a numeric field is equal to zero. This is simpler than seeing if FIELD=0.

**IIF(logical expression, result when .T., result when .F.)**. This is an expression used a lot in reports and in searches. For each record, a logical condition is evaluated, and if true, the first result is returned, otherwise the second result is returned.

**Example:** IIF(MATURITY < 80, "EARLY", MATURITY). If MATURITY is less than 80 days, a text field, such as MATGRP (maturity group) is given a value "EARLY", otherwise the current value is retained. The user could have first assigned "MEDIUM" to all genotypes, then this expression, then IIF(MATURITY>95, "LATE",MATURITY), for example.

**INLIST(x, i, j, k, l)** Instead of including records in a range, you want to select an arbitrary list of entries that do not fit any logical condition. You want entries "in your list". So, use INLIST for field X, and then list as many as you want to include in your list.

**Example:** INLIST(ENTRY,11,45,6,34,22,67) includes all records with ENTRY equal to either 11, 45, 6, 34, 22, or 67. The list of numbers can be as long as fits in the expression builder. There is really no simpler way, other than possibly creating another field and flagging those records and then building the condition on that new field, but those are more steps!

**Example:** INLIST(LOCCODE,"BN","LT","WG") would only select records where "BN", "LT", or "WG" are found in the location codes. This is a fast way to filter out locations. Note: this must be an exact match, it cannot be a text search.

**ISALPHA(x).** Check and see if the left-most character in a text expression is an alphabetic character. This is much simpler than checking if it is found in the string “abcde ...”. You might use this to check for character data in a field where there could be some digits. This function returns a .T. or .F.

**ISDIGIT(x).** Check and see if a character value is a digit. This is much simpler than checking if it is found in the string “0123456789”. You might use this to check for numeric data in a field, or confirm the absence thereof. This function returns a .T. or .F.

**Example:** ISDIGIT(“a”) returns a .F., whereas ISDIGIT(“1”) returns a .T. If scanning text, try ISDIGIT(SUBSTR(XX,1,1)) to see if the first character in the text field “XX” is a digit. ISDIGIT(SUBSTR(NAME,6,2)) will check if the sixth character is a digit, regardless whether the seventh is a digit or not. This can also be very useful in germplasm searches.

**ISUPPER(x).** Determine if the character is in upper case.

**Example:** ISUPPER(SUBSTR(NAME,3,1)). That would find all records where the third character in the NAME field is upper case. This could be useful in standardizing names.

**ISLOWER(x).** Determine if the character is in lower case.

**Example:** ISLOWER(SUBSTR(NAME,3,1)). That would find all records where the third character in NAME is in lower case.

**TYPE(x).** Determine the type (numeric, character, data, logical) of a field, and even if it is present. This is very useful in developing reports, where a field might not always be in a given experiment or nursery.

**Example:** TYPE(“TWT”). This would return “U” if not present, “N” if present and numeric, “C” if present and character, “L” if present and logical, “D” if present as a date field type.

**Example:** IIF(TYPE(“TWT”)='N',TWT,0). This expression would check if TWT is indeed present, then if it is numeric, return the value to possibly the report generator, otherwise return a zero.

## AGROBASE Generation II Functions

A number of functions have been developed uniquely for users of Generation II, reflecting the needs of agronomic research.

**CALC(“calc”,“field”,“condition”).** In addition to the report calculation functions, you might want to make other calculations. By using the arguments “A” for averaging, “T” for summing, “S” for standard deviation, “V” for variance, “C” for count, and “U” to find the number of unique values, you can make such calculations on a field, or store the results in another field, also with any condition. This permits the calculation of values relative to others when used with a condition, or a constant for other data manipulations.

**Example:** CALC(“A”,“YIELD”,“ENTRY=1”) would compute the average yield for Entry 1. YIELD/CALC(“A”,“YIELD”,“ENTRY=1”)\*100 would compute the percentage of yield relative to the first entry.

**EXGMEAN(“field”)** Returns the experiment grand mean for the trait passed as “field”, if the experiment has been analyzed. Note that if a strip trial or GLM analysis has been done, there might be a different means value stored in the database as a mean for that trait, rather than simply computing the mean afresh via CALC using “A”. For large datasets, this would faster than re-computing using all the data from all open records once again.

**Example:** YIELD-EXGMEAN(“YIELD”) would subtract the stored means from yield from each entry, and show the results as plus and minus values.

**EXCMEAN("field")** Returns the experiment check mean for the trait passed as "field", if the experiment has been analyzed. The stored value for the check(s) mean will be used from the database.

**Example:** YIELD-EXCMEAN("YIELD") would subtract the stored check mean from yield for each entry, and show the results as plus and minus values.

**FINDPLOT("i","j","x",.L.)** – generate a character string of plot (i-field) numbers for each entry (j-field) across as many reps or blocks, with "x" as the separator, and include or exclude the first occurrence via the logical expression. You might use this to print the plot numbers for each entry in each plot in a field book.

**Example:** FINDPLOT("PLOT","ENTRY","-",".F.) might return "27-64-76" for Entry number 4 in a four replication experiment. However, you can use the right-click for an experiment on the research tree to display and print a randomization table.

**MY\_AVG("\*\*X\*\*")** – compute averages for traits which have the same "root" name, such as CUT1, CUT2, and CUT3 which have "CUT" as the same root. *This works only as a calculated trait.*

**Example:** MY\_AVG("\*\*CUT\*\*") would compute the averages for all traits with that root name. This would be simpler than selecting all the fields and making an expression for the simple averages.

**PCEGM("i")** to compute data in numeric field "i" as a percentage of the Experiment's grand mean. This is very useful to compute the percentage of a trait such as yield, to "standardize" it in that sense and save it to a new trait, and later use COLLECT or HEAD-to-HEAD to make comparisons on relative yield, or any other trait.

**Example:** PCEGM("YIELD") to compute all data as a percent of the experiment mean.

**PCECM("i")** to compute data in numeric field "i" as a percentage of the Experiment's checks, whether one check or the average of several checks. Again, this standardizes all data as a percent of the checks. For later COLLECT or HEAD-to-HEAD comparisons.

**Example:** PCECM("YIELD") would compute all data as a percent of the check mean.

**PCTGM("i")** and **PCTCM("j")** will compute similarly on an entire trial basis.

**NORML(i,j)** – generate normal distribution values with a mean of "i" and a standard deviation of "j". As many values are generated as there are open records. This is invaluable in giving an intended analysis a "dry run" when no actual data is available. Perhaps you have randomized a lattice square for the first time, or want to analyze data with a specific linear model, and you want to be sure about analysis when the data is entered an reports are due soon thereafter.

**Example:** NORML(30,3) would generate normally distributed data around a mean of 30 and with a standard deviation of 3.

**RTOTAL(i)** – generate a running total from a numeric field.

**Example:** RTOTAL"PLOT") would generate a sequence of 1, 3, 6, 10, ... assuming plot numbers 1, 2, 3, 4, 5, ...

**STDScore("i")** – generate the standard scores (standardized data) for a numeric trait. This function, accessible within TRANSFORM and **Edit->Transform Field Data** alike, will subtract the mean from a dataset, and divide by the standard deviation. The resulting values will then express the data as standard deviations above or below the mean. This could be helpful when comparing data across locations and years irrespective of the grand mean of an experiment. This will be computed down the open column or field, for all records not filtered out.

Thus, the breeder could select the entries at least say 1.25 standard deviations above the mean, and maybe then build an index for the same criteria for several traits. This would also allow for an expression of a trait(s) irrespective of the units of measurement or plot size and conversion to kg/ha, tons/ha, etc. This capability should be used when Trial means are open, and you are building indexes to select varieties for the next year of testing.

**STEP(i, j, k).** Create a set of numbers starting at i, repeat j times, then add k, repeat j times, and continue as long as there are open records.

**Example:** STEP(1, 5, 4) would give 1,1,1,1,1,5,5,5,5,5,9,9,9,9,9,13,13,13,13,13, etc.

**MYPEDIGREE(i).** (*SQL only*) Used in reports and labels to output the complete pedigree if the pedigree string is longer than 254 characters. In most cases, use 1 as the parameter. For some factorial and split-plot designs, use the level at which the pedigrees are relevant (if treatments with pedigrees are found in factor 2, use 2). This function works in Experiments, Nurseries, Trials, and the main nodes for Treatments, Parents, and Populations.

**Example:** MYPEDIGREE(1) will output the complete pedigree for the treatments found in the experiment or nursery.

**TRGMEAN("field")** Returns the trial mean grand mean for the trait passed as "field", if the trial has been analyzed.

**Example:** YIELD-TRGMEAN("YIELD") would subtract the stored trial mean from yield for each entry, and show the results as plus and minus values.

## Memory Variables

Variables in memory, like some FoxPro system variables, and special Generation II memory variables, can be used in reports and even some expressions and calculations. Note that memory variables typically begin with a "\_" so they will never be confused with a field in an open database file. The ones to note are:

### FoxPro System Memory Variables

**\_PAGENO** – holds the current page number for a report.

**\_PAGETOTAL** – holds the current page number for a report. So, you can develop a "Page 1 of 8" type of page heading.

In a report, you could have an expression generate something like "PAGE 1 of 5", for example.

### AGROBASE Memory Variables

**\_experimentname** – holds the experiment/nursery name for the loaded experiment.

**\_experimentinfo** – holds the experiment/nursery description for the loaded experiment.

**\_experimentlocation** – holds the experiment/nursery location for the loaded experiment.

**\_experimentyear** – holds the experiment/nursery year for the loaded experiment.

**\_groupname** – the name of the group the experiment/nursery/trial is found in.

**\_reporttitle** – when activated, it can be used to print a report with the same title across experiments or nurseries.