AGRESSO Report Writer Language Reference
Version 2.4

Compiled From Various Sources
by Mike Schofield
Technical Consultant
UNIT4 Business Software Limited

Contents

Introduction.....5
History.....5
Command-Line Tool.....5
Database Independence.....5
Language Independence.....5
Multilingual Reports.....5
Current Usage.....5
Syntax Conventions Used In This Manual.....6
The Statements.....7
Report Setup Statements.....7
The .NAME Statement.....7
The .SETUP Statement.....7
Comments.....7
The .ENCODING Statement.....7
The .OUTPUT Statement.....8
The .QUERY Statement.....8
The .SORT Statement.....8
The .BREAK Statement.....8
The .DECLARE Statement.....8
The .INCLUDE Statement.....9
The .DEF Statement.....9
The .CALL Statement.....9
The .TRANSLATE Statement.....9
The .EXPORT Statement.....9
The .CLEANUP Statement.....10
Report Structure Statements.....10
The .HEADER Statement.....10
The .DETAIL Statement.....11
The .FOOTER Statement.....11
Page Layout and Control Statements.....11
The .FORMFEEDS Statement.....11
The .NOFORMFEEDS Statement.....11
The .NEED Statement.....11
The .NEWPAGE Statement.....11
Column Statements.....12
The .FORMAT Statement.....12
The .POSITION Statement.....12
Text Positioning Statements.....12

The .LINESTART Statement 12
The .LEFT Statement 13
The .CENTER Statement 13
The .RIGHT Statement 13
The .TAB Statement 13
The .NEWLINE Statement 13
Print Statements 13
The .PRINT Statement 13
The .PRINTADDR Statement 14
The .PRINTSTRINGADDR Statement 14
The .IMAGE Statement 14
The .DOCUMENT Statement 15
The .MAIL Statement 15
Miscellaneous Statements 15
The .IF Statement 16
The .THEN Statement 16
The .ELSEIF Statement 16
The .ELSE Statement 16
The .ENDIF Statement 16
The .LET Statement 16
ARW Functions 18
ASQL Functions 18
ARW Body “ASQL” Functions 18
Attribute Functions 18
The GETDESCRIPTION Function 18
The GETTEXT Function 18
The MAILADDRESS Function 19
Condition Functions 19
The BREAK Function 19
The ISNULL Function 19
Image Functions 19
The GETBLOBREF Function 19
The GETBLOBTRANSREF Function 20
Number Functions 20
The ABS Function 20
The COUNT Function 21
The CUMULATIVE Function 21
The MOD Function 21
The SUM Function 22

String Functions 22
The LEFT Function 22
The LENGTH Function 22
The LOCATE Function 22
The LOWER Function 23
The RIGHT Function 23
The SUBSTRING Function 23
The UPPER Function 24
Conditions 25
Format Specification 26
Character Formatting 26
Date Formatting 26
Numeric Formatting 27
Default Format Specifications 27
Process Parameters 28
Report Writer Variables 29
ARW-File Storage 30
From ABW 5.5 sp2 Onwards 30
Prior to ABW 5.5 sp2 30
Command Line 31

Introduction

History

On versions one to three of AGRESSO we only used the Ingres database. This database had a tool for creating reports called 'Ingres Report-Writer'. We used this tool to create all reports in AGRESSO. On version four of AGRESSO we started using other databases such as ORACLE and SYBASE on which Ingres Report-Writer is not available; so we created Agresso Report Writer (ARW). ARW is based on Ingres Report-Writer and works the same way, but we left out some functionality that we didn't need and added some AGRESSO specific functionality. In AGRESSO Business World (ABW) ARW has been given a new lease of life, see below

Command-Line Tool

Agresso Report Writer is a command-line tool originally written to run as a batch process, at ABW 5.5 sp2 it was enhanced so that it could also run over the Web as a part of Visualizer[1]. The commands used to create a report are stored as plain-text in an arw-file; the arw-file can be stored in a number of ways. The arw-file is read by a command-line program which interprets these commands to create the output report

Database Independence

The same arw-file can potentially work on all the database platforms we use

Language Independence

If its titles are stored in the database the same arw-file can produce reports in different languages

Multilingual Reports

The language used can be dependent on the data being printed and can vary from line to line

Current Usage

Because ARW doesn't have the inbuilt ability to print images, graphics and "font effects" it is rarely used to produce batch reports, however it's still a useful tool for producing complex output files for data transfer, e.g.:

- Files with multiple record types e.g. header or footer records
- EDI messages
- HTML files
- XML files

The fact that ARW can produce HTML output is the reason why it is one of the renderers for Visualizer

[1] Visualizer is the "printing from the Web" sub-system released in ABW 5.5 sp2

Syntax Conventions Used In This Manual

Upper case words in **BOLD**

Key words but can be written in upper or lower case. Statements start with a full-stop (.), functions do not

Lower case words in *italic*

Variable arguments.

[Value1]

'Value1' is optional

{Value1|Value2}

Either 'Value1' or 'Value2' may be used

…

Repeat as required

The Statements

Report Setup Statements

These commands set up the overall report environment:

The .NAME Statement

.NAME *report_name*

report_name	is the unique name of this report

This command is used to name your report.
The given name is used in standard AGRESSO reports to retrieve language specific titles for the report.

By adding your own entries to the asysrepref table you can implement language independence in your own reports. As always when you modify an “asys…” table you need to create a SQL script that the customer can use to reapply the changes you made after the customer has carried out an ABW upgrade

The .SETUP Statement

.SETUP *sql*

sql	is a single valid ASQL statement

This statement defines a SQL statement (DDL or DML) that will be run at the beginning of the report before the **.QUERY** statement's SQL.

The SQL must be written in ASQL[2] syntax and must not generate a result set, i.e. it must not be a SELECT statement. Process parameters may be used in the SQL in the usual way

Comments

/* *comment_text* ***/**

comment_text	is text that will be ignored during report processing

Comments can be placed anywhere in your arw-file except inside strings. A comment can span multiple lines

The .ENCODING Statement

.ENCODING {ANSI | UNICODE | UTF8}

ANSI	More correctly known as ASCII[3]. This scheme uses one byte per character and is based on the English alphabet. It defines codes for 128 characters: 33 are non-printing, mostly obsolete control characters that were used to affect how text was processed, and 95 are printable characters
UNICODE	This scheme uses a variable number of bytes per character and its aim is to encode text expressed in any of the world's writing systems. It consists of a repertoire of about 10,000 characters
UTF8	This scheme uses between one and four bytes per character and is a **U**nicode **T**ransformation **F**ormat which is backwardly compatible with the 128 characters in the ASCII scheme

This statement can only be used from ABW 5.5 sp2 onward

Set the character encoding scheme that will be used in the output file of your report.

[2] Agresso SQL (ASQL) is a proprietary dialect of SQL – See the ‘ASQL Language Reference’ for details
[3] **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange

If the **.ENCODING** statement is not present in your arw-file then the value (**ANSI**, **UNICODE** or **UTF8**) of the process parameter 'rep_encoding' will be used.

If both the **.ENCODING** statement and process parameter 'rep_encoding' are not present then the value (**ANSI**, **UNICODE** or **UTF8**) of the system parameter 'DEF_FILE_ENCODING' will be used

The .OUTPUT Statement

{**.OUTPUT**|**.OUT**} "*file_name*"

file_name	is the path of a file to which the output will be written instead of the normal ".lis" file

When you use this command a .lis file will not be created.

If *file_name* does not contain a path then the file will be created in the ‘Bin’ folder (‘Server’ folder prior to ABW 5.5), similarly relative paths will start from this folder.

See the **-f** command line option.

Unfortunately neither a variable nor a parameter can be used here

The .QUERY Statement

.QUERY [**DATABASE**] *sql*

DATABASE	the SELECT statement is interpreted in native syntax, if omitted then ASQL is used
sql	is a valid SELECT statement

Defines the SQL that retrieves the data used by the report, it can be written in native or ASQL syntax.

If you are going to use the **.SORT** statement then your SQL must not have an “ORDER BY” clause

The .SORT Statement

.SORT *col_name*[**:d**] [, *col_name*[**:d**]…]

col_name	is a data column present in the SELECT clause of the **.QUERY** statement
:d	indicates a descending sort

The purpose of the **.SORT** statement is to sort the **.QUERY** statement’s result set and to define break columns for use in **.HEADER** and **.FOOTER** statements and the **CUMULATIVE** function. The break columns can be overridden by the **.BREAK** statement. You can sort the result of the query on one or more columns.

You cannot use the **.SORT** statement if the SQL in the **.QUERY** statement includes an “ORDER BY” clause

The .BREAK Statement

.BREAK *col_name* [, *col_name*…]

col_name	is a data column present in the SELECT clause of the .QUERY statement

Specifies which data columns can be used in **.HEADER** and **.FOOTER** statements and the **CUMULATIVE** function, if it is present it overrides this aspect of the **.SORT** statement's functionality

The .DECLARE Statement

.DECLARE *variable_name* = *data_type*

variable_name	is the name of the variable to declare
data_type	is a valid data type: {**INTEGER**|**INT**}

	Will hold numeric values with no decimal places
	FLOAT
	Will hold numeric values with decimal places
	CHAR
	Will hold variable length strings of characters

This statement declares a variable for use in your arw-file, once declared variables can have values assigned to them using the **.LET** statement and can be used in the same way as a column name from the **.QUERY**.

Examples:
.DECLARE iTotalFlag = integer
.DECLARE fSumTotAmount = float
.DECLARE cVariantText = char

The .INCLUDE Statement

.INCLUDE "*file_name*"

file_name	is the name of the file you want to include

This statement includes a file with procedures (defined with the **.DEF** statement) in your arw-file so that you can **.CALL** them

The .DEF Statement

.DEF *procedure_name*

procedure_name	is the name of the procedure you want to define

This statement specifies a section of statements that forms a procedure. The procedure can be defined locally or in a separate file. If the procedure is defined in a separate file then that file must be included in your arw-file using the **.INCLUDE** statement. The procedure is started using the **.CALL** statement

The .CALL Statement

.CALL *procedure_name*

procedure_name	is the name of the procedure you want to call

This statement calls an already defined procedure. The procedure is created using the **.DEF** statement either locally or in a file included using the **.INCLUDE** statement

The .TRANSLATE Statement

.TRANSLATE *from = to*

from	is the old character value
to	is the new character value

Both *from* and *to* can be either an ASCII code or a character enclosed in single quotes (e.g. 97 or ‘a’)

The .TRANSLATE statement translates one character to another. It must be defined in the section of a ‘Report Structure Statement’ and once it has been executed the *from* character will always be converted to the *to* character in subsequent ‘Print Statements’

The .EXPORT Statement

.EXPORT {*col_name* | *variable_name*} [,{*col_name* | *variable_name*}…]

col_name	is a data column present in the SELECT clause of the .QUERY statement
variable_name	is a variable defined by the **.DECLARE** statement

The .EXPORT statement provides a way in which your report can send data back to its calling process, in order for this to work the calling process must be a bespoke program.

The .EXPORT statement writes the named columns and variables and their values back to the report's parameter file, destroying that file's original content. Because the .EXPORT command removes your report's run parameters it should only be called once, at the end of the report. Once your report has finished the calling process can then read the parameter file to get the values.

Example:

```
.DECLARE test_par = CHAR
.DECLARE tot_amt = FLOAT
.HEADER report
        .LET test_par = "Hello World"
        .LET tot_amt = 0
.DETAIL
        .LET tot_amt = tot_amt + ABS(amount)
.FOOTER report
        .EXPORT test_par, tot_amt
```

After the report has run the parameter file will contain:

```
test_par=Hello World
tot_amt=1335072.580000
```

The exact value of 'tot_amt' is data dependent

The .CLEANUP Statement

.CLEANUP *sql*

sql	is a single valid ASQL statement

This statement defines a SQL statement (DDL or DML) that will be run at the end of the report. The SQL must be written in ASQL syntax and must not generate a result set, i.e. it must not be a SELECT statement. Process parameters may be used in the SQL in the usual way

Report Structure Statements

These commands control the level of detail printed, headers, footers and totalling. As the report executes a number of "events" occur, statements can be linked to these events as follows:

Report start	**.HEADER REPORT**
Page start	**.HEADER PAGE**
"Break column" start	**.HEADER** *col_name*
New row in the .QUERY statement's result set	**.DETAIL**
"Break column" end	**.FOOTER** *col_name*
Page end	**.FOOTER PAGE**
Report end	**.FOOTER REPORT**

A "break column" is enabled by listing that column's name in either the **.SORT** or **.BREAK** statement

Each of these statements introduces a block of statements to be run. All subsequent statements until the next 'Report Structure Statement' (or the end of the arw-file) belong to that block

The .HEADER Statement

{**.HEADING**|**.HEADER**|**.HEAD**} {**REPORT** | **PAGE** | *col_name*}

REPORT	Run this header once at the beginning of the report
PAGE	Run this header at the top of every page
col_name	Run this header at the beginning of each new value of this column, which must be listed in the **.SORT** or **.BREAK** statement

This statement introduces a block of statements that will be run when the specified event occurs

The .DETAIL Statement

.DETAIL

The **.DETAIL** statement has no parameters

This statement introduces a block of statements that will be run once for each row in the **.QUERY** statement's result set.

The .FOOTER Statement

{**.FOOTING|.FOOTER|.FOOT**} {**REPORT** | **PAGE** | *col_name*}

REPORT	Run this header once at the end of the report
PAGE	Run this header at the bottom of every page
col_name	Run this header at the end of each new value of this column, which must be listed in the **.SORT** or **.BREAK** statement

This statement introduces a block of statements that will be run when the specified event occurs

Page Layout and Control Statements

These commands control when the report starts a new "page" and whether form feed characters are sent to the printer or not.

The .FORMFEEDS Statement

{**.FORMFEEDS|.FFS**}

The .FORMFEEDS statement has no parameters

If this is present in the **.HEADER PAGE** section then the **.NEWPAGE** statement does insert a form feed character {ASCII(12)} in the output file
See the **-b** command line option

The .NOFORMFEEDS Statement

{**.NOFORMFEEDS|.NOFFS**}

The .NOFORMFEEDS statement has no parameters

If this is present in the **.HEADER PAGE** section then the **.NEWPAGE** statement does not insert a form feed character {ASCII(12)} in the output file
See the **-b** command line option

The .NEED Statement

.NEED *n*

n	is the number of lines needed to print the text block

This statement ensures that a specified number of lines are printed together on the same page.
If the number of remaining lines on the current page is less than those needed for printing the given number of lines, a page break is generated and printing will continue on the next page. It is typically used to ensure that all the lines in a section are printed together

The .NEWPAGE Statement

{**.NEWPAGE|.NP**} [[{+ | -}] *n*]

{+|-}	indicates a relative change. If no value of *n* is given the default is +1 If {+|-} is absent then *n* is an absolute page number
N	is the number to set/change the page number to.

The .NEWPAGE statement initiates an immediate page break with an optional change of page number. See the **.FORMFEEDS** and **.NOFORMFEEDS** statements

Column Statements

You can use column statements to set up an explicit print position and format for the values contained in the named database column. For instance, you can use **.POSITION** to assign the starting print position for a column, which is used in conjunction with Text Positioning Statements (see the next section). If you do not specify column print positions with **.POSITION**, then you must use explicit column numbers on the Text Positioning Statements. If you do not specify column formats with the **.FORMAT** statement and do not supply a format specification on **.PRINT** statements then ARW will use default formats: see the 'Default Format Specifications' section.

Use the following column statements:

The .FORMAT Statement

.FORMAT *col_name* [, *col_name…*] *format*
[, *col_name* [, *col_name…*] *format…*]

col_name	is a data column present in the SELECT clause of the .QUERY statement
format	is a valid format specification for the column's datatype

The .FORMAT statement can be used to specify a column format used whenever the data in the column is printed on the report. This data format controls the width of the output and also in which form the output should be printed (where to print the minus sign, how to print the elements in a date column, etc)
Format specifications given explicitly in the **.PRINT** statement will be used instead of specifications given in the .FORMAT statement.
If a *col_name* does not appear in a .FORMAT statement and no format is supplied on its .PRINT statement then ARW uses default formatting.

Valid format specifications and defaults are described later

The .POSITION Statement

{**.POSITION**|**.POS**} *column* [, *column…*] **(***n***)** [, *column* [, *column…*] **(***n***)**…]

column	is a name of the position (e.g. the name of a print column)
n	is a print column number between 0 and the width of the report page

ARW does not calculate default positions for print columns, all named print columns used by other ARW statements must explicitly be given a position in the **.POSITION** statement

Text Positioning Statements

These commands define where the next item to be printed will appear, either in absolute terms or relative to the last print position. Most of these statements accept the name of a column set with the **.POSITION** statement or a numeric value. See Column Statements.

Use the following text positioning statements:

The .LINESTART Statement

{**.LINESTART**|**.LNSTART**}

The .LINESTART statement has no parameters

This statement left justifies to the left of page the next text to be printed.
It is equivalent to a **.LEFT** statement with no parameter

The .LEFT Statement

.LEFT [{*n* | *column*}]

n	is the print column number at which the text is to be left justified
column	is a print column named by the .POSITION statement

This statement left justifies the next text to be printed.
If a parameter is not supplied the text is left justified to the left of page, making it equivalent to the **.LINESTART** statement

The .CENTER Statement

.CENTER [{*n* | *column*}]

n	is the print column number at which the text is to be centred
column	is a print column named by the .POSITION statement

This statement centres the next text to be printed.
If a parameter is not supplied the text is centred around the centre of the page

The .RIGHT Statement

.RIGHT [{*n* | *column*}]

n	is the print column number at which the text is to be right justified
column	is a print column named by the .POSITION statement

This statement right justifies the next text to be printed.
If a parameter is not supplied the text is right justified to the right of the page

The .TAB Statement

.TAB {[{+|-}]*n* | *column* | *x,y*}

[{+|-}]*n*	is a print column number. The use of **+** or **-** indicates relative positioning
column	is a print column named by the .POSITION statement
x,y	are a column and line number respectively

The **.TAB** statement specifies the position at which the next text is to be printed.
Using the *x,y* option it can be used to move both in vertical and horizontal direction

The .NEWLINE Statement

{**.NEWLINE**|**.NL**} [*n*]

n	is the number of lines to advance. The default is 1 (advance to the next line)

The **.NEWLINE** statement moves the current position marker to the left margin after a given number of new lines

Print Statements

Use these statements to print text or data values in a report:

The .PRINT Statement

{**.PRINT**|**.PR**|**.P**} {*expression*|*title*[**[***language***]**]} [*format*] [,{*expression*|*title*[**[***language***]**]} [*format*]…]

expression	is a literal, data column, variable, report parameter, result of an operation or function call
title	is the name of a ‘title_id’ for this report from asysrepref
[*language***]**	An optional language code enclosed in square brackets […]

format	is a valid format specification

Printing an *expression* simply prints its value

All literals must be given in speech marks or apostrophes e.g. "text" or 'text'; speech marks are the preferred option, but apostrophes can be used when the text includes speech marks.

Printing a *title* is more complex.

If a title_id is used without a **[***language***]** qualifier then the language in report parameter $lg is used

If a title_id is used with a **[***language***]** qualifier then that language is used, this can be a literal or a data column

If present the format specification on a **.PRINT** statement overrides that data column's **.FORMAT** statement entry, if not present a data column's **.FORMAT** statement entry is used, if the column has no **.FORMAT** statement entry then default formats are used. Valid format specifications and defaults are described below.

Functions **GetDescription** and **GetText** can be used to get descriptions and texts, respectively, from ABW.

Example:
.PRINT column1
.PR "Page: ", page_number {"zzn"}
.PR "Client: ", $client
.PR SUM(column2)
.PR '<Co xmlns="">', client, "</Co>"

The .PRINTADDR Statement

{**.PRINTADDRESS|.PRINTADDR**} *addr* [, *n*]

addr	is a data column which holds the address to print
n	is the max number of lines used to print the address.

This statement is used to print addresses. The address can be printed on more than one line and optionally restricted to a given number of lines using the *n* argument.
The **maxaddrline** report writer variable holds the number of lines printed

The .PRINTSTRINGADDR Statement

.PRINTSTRINGADDR *addr*

addr	is a data column which holds the address to print

This statement is allegedly used to print old AGRESSO 4 addresses. I have never tested this so cannot vouch for either its syntax or operation

The .IMAGE Statement

.IMAGE [*blob_id*, *interface*]

blob_id	is a unique number which is the index to aimblob
interface	is a value of the ABW attribute IMAGE (GQ), this determines the kind of image being stored as a BLOB, e.g. text file, MSWord document, etc. Because an ARW can only create plain text this will have a value of "D4"

This statement was replaced by **.DOCUMENT** in ABW 5.5 sp2; in prior releases it made it possible to send the result or part of the result from your ARW file to the document archive in ABW. The result generated between this statement with arguments and the next occurrence of this statement with or without arguments will be written to a file and the stored in the aimblob table.
To get a *blob_id* you must use function **GetBlobRef** or **GetBlobTransRef**

Examples:

```
.IMAGE
        GetBlobRef(
                "HS", "C0", "87010101", 1, 0, "ARW Demonstration", "PC", "EN",
                "sysen"),
        "D4" /* interface */
.PR "This text will be stored against RESNO 87010101"
.IMAGE
        GetBlobTransRef(
                "VP", voucher_no, sequence_no, line_no, ext_inv_ref, $doc_type,
                $client, $user_id),
        "D4"
.PR "This text will be in the transaction blob"
.IMAGE
.PR "This text will not be in a blob"
```

The .DOCUMENT Statement

.DOCUMENT [*title*, *description*, *doc_type*, *client*, *entity*]

title	A unique title for the document
description	A description of the document
doc_type	A valid document type defined in the **Agresso Common ► Document archive ► Fixed registers ► Document type** screen (DS10)
client	The ABW company in which to store the document
entity	The entity against which to store the document

This statement replaces **.IMAGE** in ABW 5.5 sp2. It made it possible to send the result or part of the result from your ARW file to the document archive in ABW. The result generated between this statement with arguments and the next occurrence of this statement with or without arguments will be stored in the ABW document archive

Examples:

```
.DOCUMENT
        t_sales_order_copy + invoice_id,
        t_sales_order_copy_cust + " " + apar_id,
        $doc_type,
        $apar_client,
        apar_id
.PR "This text will be stored against the customer"
.DOCUMENT
.PR "This text will not be in the document archive"
```

The .MAIL Statement

.MAIL [*email_ address*]

email_address	is the e-mail address you want the result to be sent to

This statement makes it possible to send the result or part of the result from your ARW as an e-mail. The result generated between this statement with arguments and the next occurrence of this statement will be written to the specified e-mail address. The e-mail address can be written using double quotes or it can be found using the function **MailAddress**.

Miscellaneous Statements

Use these statements to specify alternative blocks of statements and to assign values to variables:

The .IF Statement

.IF *condition* **.THEN** [*statements*]
[**.ELSEIF** *condition* **.THEN** [*statements*] [**.ELSEIF** *condition* **.THEN** [*statements*]…]]
[**.ELSE** [*statements*]]
.ENDIF

condition	One or more comparisons between one expression (literal, data column, variable, report parameter, result of an operation or function call) and another. Round brackets (…) can be used to group comparisons when a mixture of **AND**s and **OR**s are used
statements	are one or more ARW statements

.IF statements are used to specify blocks of *statements* which are only to be executed if the specified *condition*(s) are true (**.THEN** block) or false (**.ELSE** block)

The .THEN Statement

.THEN [*statements*]

statements	are one or more ARW statements

All statements up to the next **.ELSEIF**, **.ELSE** or **.ENDIF** are executed.
See the **.IF** statement

The .ELSEIF Statement

.ELSEIF *condition* **.THEN** [*statements*]

condition	One or more comparisons between one expression (literal, data column, variable, report parameter, result of an operation or function call) and another. Round brackets (…) can be used to group comparisons when a mixture of **AND**s and **OR**s are used
statements	are one or more ARW statements

All statements up to the next **.ELSEIF**, **.ELSE** or **.ENDIF** are executed.
See the **.IF** statement

The .ELSE Statement

.ELSE [*statements*]

statements	are one or more ARW statements

All statements up to the next **.ENDIF** are executed.
See the **.IF** statement

The .ENDIF Statement

.ENDIF

	The .ENDIF statement has no parameters

See the **.IF** statement

The .LET Statement

.LET *variable_name = expression*
Where

variable_name	is a variable defined by the .DECLARE statement
expression	is a literal, data column, variable, report parameter, result of an operation or function call

This statement assigns the value of an expression to a variable

Example:

```
.LET totals = 0
.LET totals = totals + column
```

ARW Functions

You can use six types of functions in an arw-file:

- ASQL functions
- Attribute Functions
- Condition Functions
- Image Functions
- Number Functions
- String Functions

ASQL Functions

These are only available for use in the **.CLEANUP**, **.QUERY** and **.SETUP** sections

ARW Body “ASQL” Functions

There are functions that can be used in the body of an ARW that have the same names, and similar functionality, to some ASQL functions, these include:

Function	Section
ABS	Number
COUNT	Number
LEFT	String
LENGTH	String
LOWER	String
MOD	Number
RIGHT	String
SUM	Number
UPPER	String

These are dealt with individually under the section indicated.

Attribute Functions

These functions provide information about attribute values:

The GETDESCRIPTION Function

GETDESCRIPTION(*attribute_id***,** *dim_value***,** *client***)**

attribute_id	The code of the attribute to retrieve the description of
dim_value	The attribute's value
client	The client to retrieve the description from

GetDescription returns the description of an attribute value from agldescription.

Example:
.PR GetDescription("C0", resource_id, $client) {c25}

The GETTEXT Function

GETTEXT(*text_type***,** *variant***,** *language***)**

text_type	The type of text (see below)
variant	Various meanings
language	The language in which the text is written

GetText returns a message text from the table 'acrtexts'. This table contains various items such as reminder texts, statement messages etc.

Example:
.PR GetText("3", $variant, lg) {cf0.72}

The MAILADDRESS Function

MAILADDRESS(*client***,** *attribute_id***,** *dim_value***,** *addr_type***)**

client	The client to retrieve the address from
attribute_id	The code of the attribute holding the address
dim_value	The value of the attribute
addr_type	The ADDRTYPE value for the address

The function MailAddress returns an email address from table agladdress

Example:
.MAIL MailAddress (client, "C0", resource_id, "1")

Condition Functions

These functions can be used in *condition* in the **.IF** and **.ELSEIF** statements:

The BREAK Function

BREAK(*col_name***)**

col_name	is a data column present in the **SELECT** clause of the **.QUERY** statement

Break is used in conditions and returns TRUE when the value of *col_name* changes.

The ISNULL Function

ISNULL(*col_name***)**

col_name	is a data column present in the SELECT clause of the .QUERY statement whose type is DATE or CHAR

IsNull is used in conditions and returns TRUE when the column's value is blank

Image Functions

These functions must only be used in conjunction with the **.IMAGE** statement. Both these functions and the **.IMAGE** statement carry out updates to the ABW database and to do one without the other will damage the integrity of the database. BLOB stands for Binary Large Object and is data that is stored either:

- in a form that is not plain text, such as a MS Word document or a scanned image
- or is an arbitrarily large amount of text

The GETBLOBREF Function

GETBLOBREF (*trans_type*, *attribute_id*, *dim_value*, *sequence_no*, *line_no*, *description*, *image_type*, *client*, *user_id***)**

trans_type	is an ABW module code
attribute_id	The code of the attribute that the BLOB is being held against
dim_value	The value of the attribute that the BLOB is being held against
sequence_no	Unknown
line_no	A single attribute value can have more than one BLOB held against it with the same *trans_type*, each one must have a unique *line_no*
description	to be displayed on the ‘Document’ tab on ABW screens

image_type	Unknown
client	is the ABW company in which the BLOB is being stored
user_id	is an ABW user id, this is for audit trail purposes, in addition the ‘Updated’ column will be set automatically

GetBlobRef inserts a row in the 'aimblobref' table and returns the BLOB id used for the **.IMAGE** statement to use in its update of aimblob

Example

```
.IMAGE
        GetBlobRef(
                "HS", "C0", "87010101", 1, 0, "ARW Demonstration", "PC", "EN",
                "sysen"),
        "D4" /* interface */
.PR "This text will be stored against RESNO 87010101"
```

The GETBLOBTRANSREF Function

GETBLOBTRANSREF(*trans_type*, *voucher_no*, *sequence_no*, *line_no*,
description, *image_type*, *client*, *user_id***)**

trans_type	is an ABW module code
voucher_no	is the ‘TransNo’ of the transaction that the BLOB is being held against
sequence_no	is the ‘#’ of the transaction that the BLOB is being held against
line_no	A single transaction row can have more than one BLOB held against it with the same *trans_type*, each one must have a unique *line_no*
description	to be displayed on the ‘Document’ tab on ABW screens
image_type	Unknown
client	is the ABW company in which the BLOB is being stored
user_id	is an ABW user id, this is for audit trail purposes, in addition the ‘Updated’ column will be set automatically

GetBlobTransRef inserts a row in the 'aimtransblobref' table and returns the BLOB id used for the **.IMAGE** statement to use in its update of aimblob

Example:

```
.IMAGE
        GetBlobTransRef(
                "VP", voucher_no, sequence_no, line_no, ext_inv_ref, $doc_type, $client,
                $user_id),
        "D4"
.PR "This text will be in the transaction blob"
```

Number Functions

The ABS Function

ABS(*expression***)**

expression	is a literal, data column, variable, report parameter, result of an operation or function call

This function returns the absolute (i.e. unsigned) value of *expression*

Examples:

```
.LET fExample = ABS(-1000.50)
```

fExample will be set to 1000.50

.LET fExample = ABS(1000.50)
fExample will be set to 1000.50

The COUNT Function

COUNT(*expression***)**

expression	is a literal, data column, variable, report parameter, result of an operation or function call

This function will, by default, return the number of values, including duplicates, of *expression* in the break column group that it appears in. For example if “COUNT(voucher_no)” appears in a ‘.FOOTER account’ section then the number of ‘voucher_no’ values for the current account will be returned. This default behaviour can be overridden by using the **CUMULATIVE** function.

Be aware that if *expression* is not an integer it appears to return an incorrect result.

Example:
.PR COUNT(voucher_no) {"zzz,zzz,zzn"}

The CUMULATIVE Function

{**CUMULATIVE**|**CUM**}[**(***col_name***)**] {**COUNT(**…**)** | **SUM**(…**)**}

col_name	Reset the value at the beginning of each new value of this column, which must be listed in the **.SORT** or **.BREAK** statement. If omitted then the value is only reset at the beginning of the report

CUMULATIVE can only be used in conjunction with the **COUNT** or **SUM** functions in your arw-file and when it is used it modifies the scope of their operation.

Example:
```
.DETAIL
        .TAB brk_tot     .PR amount
        .TAB run_tot     .PR CUM(account) SUM(amount)
.FOOTER account
        .TAB brk_tot     .PR SUM(amount)
        .TAB run_tot     .PR CUMULATIVE SUM(amount)
```
The first **.PRINT** statement in the **.DETAIL** section will print the amount on this line; the second will print a running total for this account. The first **.PRINT** statement in the **.FOOTER** section will print the total for this account; the second a running total for the report so far

The MOD Function

MOD(*expression, divisor***)**

expression	is a literal, data column, variable, report parameter, result of an operation or function call
divisor	is a literal, data column, variable, report parameter, result of an operation or function call

This function returns the remainder when *expression* is divided by *divisor*

Examples:
.LET iExample = MOD(100, 25)
iExample will be set to 0

.LET iExample = MOD(102, 25)
iExample will be set to 2

The SUM Function

SUM(*expression***)**

expression	is a literal, data column, variable, report parameter, result of an operation or function call

This function will, by default, return the sum of the values of *expression* in the break column group that it appears in. For example if “SUM(amount)” appears in a ‘.FOOTER account’ section then the sum of the ‘amount’ values for the current account will be returned. This default behaviour can be overridden by using the **CUMULATIVE** function.

Example:
.PR SUM(amount) {"zzz,zzz,zzn.nn"}

String Functions

These functions allow in-report manipulation of character strings.

The LEFT Function

LEFT(*expression*, *num_chars***)**

expression	is a literal, data column, variable, report parameter, result of an operation or function call
num_chars	The number of characters to extract

This function returns the first *num_chars* characters of *expression*. If there are less than *num_chars* in *expression* then an error will occur when the report is run, see the LENGTH function.

If *expression* is a literal string it must be enclosed in apostrophes not speech marks, e.g. LEFT('abcd') not LEFT("abcd").

Example:
```
.IF LENGTH('AbCd') >= 2 .THEN
        .LET sExample = LEFT('AbCd', 2)
.ENDIF
```
sExample will be set to “Ab”

The LENGTH Function

LENGTH(*expression***)**

expression	is a literal, data column, variable, report parameter, result of an operation or function call

This function returns the number of characters in *expression*

If *expression* is a literal string it must be enclosed in apostrophes not speech marks, e.g. LENGTH('abcd') not LENGTH("abcd").

Example:
.LET iExample = LENGTH('abcd')
iExample will be set to 4

The LOCATE Function

LOCATE(*source*, *target***)**

source	is a literal, data column, variable, report parameter, result of an operation or function call
target	is a literal, data column, variable, report parameter, result of an operation or function call

This function returns the position of *target* in *source*. If *target* is not present in *source* then zero is returned

Example:
.LET iExample = Locate("Ab,Cd", ",")
iExample will be set to 3

The LOWER Function

LOWER(*expression***)**

expression	is a literal, data column, variable, report parameter, result of an operation or function call

This function returns *expression* converted to lower-case characters

If *expression* is a literal string it must be enclosed in apostrophes not speech marks, e.g. LOWER('ABCD') not LOWER("ABCD").

Example:
.LET sExample = LOWER('AbCd')
sExample will be set to “abcd”

The RIGHT Function

RIGHT(*expression*, *num_chars***)**

expression	is a literal, data column, variable, report parameter, result of an operation or function call
num_chars	The number of characters to extract

This function returns the last *num_chars* characters of *expression*. If there are less than *num_chars* in *expression* then an error will occur when the report is run, see the LENGTH function.

If *expression* is a literal string it must be enclosed in apostrophes not speech marks, e.g. RIGHT('abcd') not RIGHT("abcd").

Example:
```
.IF LENGTH('AbCd') >= 2 .THEN
        .LET sExample = RIGHT('AbCd', 2)
.ENDIF
```
sExample will be set to “Cd”

The SUBSTRING Function

SUBSTRING(*source*, *start*, *length***)**

source	is a literal, data column, variable, report parameter, result of an operation or function call
start	A data column present in the SELECT clause of the **.QUERY** statement or an expression
length	A data column present in the SELECT clause of the **.QUERY** statement or an expression

Substring returns the fragment of *source* defined by *start* and *length*

Examples:
.LET sExample = SUBSTRING("AbCd", 2, 2)
sExample will be set to “bC”

.LET sAddr2 = SUBSTRING(address, 41, 40)

sAddr2 will be set to the second address line stored in column ‘address’

The UPPER Function

UPPER(*expression*)

expression	is a literal, data column, variable, report parameter, result of an operation or function call

This function returns *expression* converted to upper-case characters

If *expression* is a literal string it must be enclosed in apostrophes not speech marks, e.g. UPPER('abcd') not UPPER("abcd").

Example:
.LET sExample = UPPER ('AbCd')
sExample will be set to “ABCD”

Conditions

The following are available for use in *condition* in the **.IF** and **.ELSEIF** statements:

Operators for comparing two expressions (literal, data column, variable, report parameter, result of an operation or function call) to make a condition:

=	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

Boolean functions are conditions on their own:

- **ISNULL(…)**
- **BREAK(…)**

Operators for combining conditions:

- **NOT**
- **AND**
- **OR**
- **(…)** for grouping conditions

Example:

```
.IF (client = 'EN' OR client = '44') AND IsNull(last_update) .THEN
```

Format Specification

Format strings are used by the **.FORMAT** and **.PRINT** statements and define how data should be presented on the report.
They are always enclosed in braces {…}

- **C** formats are for strings
- **D** formats are for dates
- **N** formats are for numbers

N.B.

Although the "C", "D" and "N" are shown in uppercase in the syntax definitions they **MUST** be used in lowercase as shown in the examples

Character Formatting

[{+ | -}] **C**[**F***n*[.*w*]]

+	Right justifies the text
-	Left justifies the text (the default)
F	Folds the text of more than one line
n	Sets the total number of characters to print. 0 means print as is: no truncation or padding
w	Sets the maximum number of characters to print on each line

Examples
The following formats are applied to "This is a string":

• {c0}	"This is a string"
• {c12}	"This is a st"
• {c20}	"This is a string~~bbbb~~"
• {+c20}	"~~bbbb~~This is a string"
• {c12.7}	"This is" "~~b~~a st"
• {c16.7}	"This is" "~~b~~a stri" "ng"
• {cf16.7}	"This is" "a" "string"

Where ~~b~~ is a single space character

Date Formatting

D"*template string*"

The following characters can be used in *template*:

%b	Abbreviated month name
%B	Full month name
%d	Day of the month
%H	Hour of the day (00-23)
%I	Hour of the day (01-12)
%j	Day of the year
%m	Month of the year

%M	Minute of the hour
%S	Second of the minute
%y	Year without century (00-99)
%Y	Year with century

Examples
The following formats are applied to "20050829 14:15:23":

• {d"%d-%B-%Y"}	"29-August-2005"
• {d"%y%m%d"}	"050829"
• {d"%d %b %Y, %H-%M-%S"}	"29 Aug 2005, 14-15-23"

Numeric Formatting

[{+ | -}] **N***w*[.*d*]
Or
[{+ | -}] "*template_string*"

+	right justifies the text
-	left justifies the text (the default)
w	is the maximum field width
d	is the number of decimals
template_string	defines "special" formatting options, see below

Examples
The following formats are applied to -1250.79:

• {n10}	"-1250~~bbbbbb~~"
• {n10.2}	"-1250.79~~bbb~~"
• {+n10.2}	"~~bbb~~-1250.79"
• {"nnnnn"}	"01250"
• {"zzzzz.zz"}	"~~b~~1250.79"
• {"z,zzz,zzn.nn"}	"~~bbbb~~1,250.79"
• {"z,zzz,zzn.nn-"}	"~~bbbb~~1,250.79-"
• {"-,---,--n.nn"}	"~~bbbb~~-1,250.79"

Where ~~b~~ is a single space character

When a template contains no "n" characters a value of zero will be completely suppressed

Default Format Specifications

In the absence of a format specification the following defaults apply:

- *Character* - Left justified, prints all characters in the string
i.e. {c0}
- *Date* - The default date format is defined by environment variable
AGR_DATE_FORMAT.
If this variable is not set, the ISO date format is used i.e. {d”%y%m%d”}
- *Float/Money* - Left justified, prints all decimals
- *Integer* - Left justified.
WARNING on Oracle ARW treats integers as floats

Process Parameters

ARW reads its parameters from parameter file which can contain an unlimited number of parameter values stored as follows:

```
parameter1=”value”,’type’
parameter2=”value”,’type’
```

When running a report from ABW (i.e. ordering it from the menu), the ABW server will create a temporary parameter file containing some standard parameters and any additional parameters that:

- On new reports: you have defined in the ‘User defined reports’ screen (AG35).
- In customised standard reports: are defined in the ‘Report variants’ screen (AG25)

The standard parameters are:

Parameter	Meaning
CLIENT	The client from which the report was ordered
CMT	The report’s description from the menu
LG	The language of user **USER_ID**
NO	The abbreviation for “no” in language **LG**, e.g. “N”
REPNAM	The report’s name (usually a 4 or 5 character code)
USER_ID	The ABW user who ordered the report
YES	The abbreviation for “yes” in language **LG**, e.g. “Y”

To refer to a parameter’s value in your arw-file you must prefix its name with a “$”, e.g. “.PR $user_id”. When referring to a parameter whose value is a string in SQL (in the **.CLEANUP**, **.QUERY** or **.SETUP** sections) you must enclose it in apostrophes, e.g. “… client = ‘$client’…”

Examples:
…WHERE client LIKE '$client'… (in the **.QUERY** section)

.PR $period_from

Report Writer Variables

You can use the following variables in your arw-file without declaring them:

• COLPOS	Current column position
• CURRENT_DATE	Run date
• CURRENT_TIME	Run time
• LINENO	Current line number
• PAGE_NUMBER	Current page number
• maxaddrline	Number of lines printed by the **.PRINTADDR** statement

Example:

```
.PR page_number {"zzn"}
```

ARW-File Storage

Where and how an arw-file is stored depends on:

- Whether it is an unmodified standard report or not
- What version of ABW you are on

From ABW 5.5 sp2 Onwards

The unmodified versions of the standard reports are held in 'Bin\agrreportresource.exe'. If you wish to customise these reports or write new ones these can be stored as:

- ".arw" files in the 'Customised Reports' folder
- blobs imported into the aagvisualizerfile table, using the ABW Management Console

If your report is going to be used in the Self-Service (Web) Client then it must be stored in aagvisualizerfile.

You can extract a standard report from 'Bin\agrreportresource' by running it at the command-line:

```
cd AGRESSO 5.5\Customised Reports
..\Bin\agrreportresource ReportName.arw
```

Where *ReportName* is the name of the report to extract. This creates a new file called "*ReportName*.arw" containing the report's source code.

When running a report ABW searches for it as follows:

- blob in the aagvisualizerfile table
- arw-file in the 'Customised Reports' folder
- arw-file in the 'Report Writer' folder - BUT DO NOT DO THIS!
- resource in the agrreportresource.exe file in the 'Bin' folder

Prior to ABW 5.5 sp2

The unmodified versions of the standard reports are held in a series of report library files (extension ".rep") stored in the 'Report Writer' folder. If you wish to customise these reports or write new ones these should be stored as ".arw" files in the 'Customised Reports' folder, both of these folders are on the Business Server.

You can extract a standard report from its library file by:

- Opening the ".rep" file in a text editor and copying the relevant report's source to the clipboard and then pasting this into a new file. Make sure that you ignore the line with non-printing characters on it and start at the line containing the **.NAME** command
- By using the 'Server\agrxrep' command\a150line program:

  ```
  agrxrep Library ReportName
  ```

  Where *Library* is the name of the report library file containing the report to be extracted and *ReportName* is the name of the report to extract, This creates a new file called "*ReportName*.arw" containing the report's source code. You will then have to move this file from 'Report Writer' to the 'Customised Reports' folder

When running a report ABW searches for it as follows:

- arw-file in the 'Customised Reports' folder
- arw-file in the 'Report Writer' folder - BUT DO NOT DO THIS!
- entry in a library file in the 'Report Writer' folder

Command Line

ARW can be run manually from the 'Command Prompt' or from within some text editors[4].

<u>From ABW 5.5 sp2 onwards the command to run is:</u>

Bin\RunRep *OrderNumber ReportName* [**-q***ServerQueue*] *DataSource*

Where:

OrderNumber	Is the order number of the report (used to get its parameters)
ReportName	Is the name of your arw-file
ServerQueue	Is the name of the queue on which to run the report e.g. "DEFAULT"
DataSource	Is the ABW data source name, as defined in the Management Console

You can also type

Bin\RunRep

for on-screen instructions

<u>Prior to ABW 5.5 sp2 the command was:</u>

C:\agresso\server\agrrep *parameters DataSource*

Where

DataSource	Is the ABW data source name, as defined in the Management Console

The following *parameters* could be given on the command line after the name of the 'agrrep' exe file:

-b	Switch off the formfeed functionality (see the **.FORMFEEDS** and **.NOFORMFEEDS** statements)
-i*arw-file*	Name of your arw-file
-f*lst-file*	Where to write the report results (see the **.OUTPUT** statement)
-l*n*	Page width (default is 132). This is a lower-case L
-p*par-file*	Name of the file containing the process parameter values
-v*n*	Page length (default is 66)
-c*lang*	Language code

You could also type:

"**agrrep** *DataSource*"

for on-screen instructions

[4] In order to be able to run your arw-file from within an editor it must have the ability to run external programs. Editors that have this facility include:

- Program File Editor (PFE)
- NotePad++
- TextPad