ACL Scripting Reference

Version: 14.2

Published Wednesday, July 15, 2020

© 2020 ACL Services Ltd. dba Galvanize

Table of contents

Table of contents **3**
Getting started 12
Analytics scripting basics 13
Comments 18
Data types 20
Expressions 21
Defining computed fields with expressions 23
Functions 25
Variables 27
Control structures 29
Grouping and looping 32
Top 30 Analytics functions 40
Commands **51**
Import and export data 52
ACCEPT command 54
ACCESSDATA command 59
ACTIVATE command 67
AGE command 69
APPEND command 73
ASSIGN command 81
BENFORD command 84
CALCULATE command 88
CLASSIFY command 90
CLOSE command 95
CLUSTER command 97
COMMENT command 100
COUNT command 102
CREATE LAYOUT command 105
CROSSTAB command 107
CVSEVALUATE command 111

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CVSPREPARE command 115
CVSSAMPLE command 120
DEFINE COLUMN command 123
DEFINE FIELD command 126
DEFINE FIELD . . . COMPUTED command 132
DEFINE RELATION command 138
DEFINE REPORT command 141
DEFINE TABLE DB command 142
DEFINE VIEW command 145
DELETE command 147
DIALOG command 151
DIRECTORY command 159
DISPLAY command 164
DO REPORT command 169
DO SCRIPT command 170
DUMP command 172
DUPLICATES command 174
ESCAPE command 180
EVALUATE command 182
EXECUTE command 186
EXPORT command 193
EXTRACT command 203
FIELDSHIFT command 208
FIND command 210
FUZZYDUP command 212
FUZZYJOIN command 217
GAPS command 223
GROUP command 227
HELP command 233
HISTOGRAM command 234
IF command 238
IMPORT ACCESS command 240
IMPORT DELIMITED command 242

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT EXCEL command 250
IMPORT GRCPROJECT command 258
IMPORT GRCRESULTS command 264
IMPORT LAYOUT command 272
IMPORT MULTIDELIMITED command 274
IMPORT MULTIEXCEL command 281
IMPORT ODBC command 288
IMPORT PDF command 291
IMPORT PRINT command 300
IMPORT SAP command 308
IMPORT XBRL command 315
IMPORT XML command 319
INDEX command 323
JOIN command 326
LIST command 332
LOCATE command 335
LOOP command 338
MERGE command 340
NOTES command 344
NOTIFY command 346
OPEN command 349
OUTLIERS command 352
PASSWORD command 360
PAUSE command 362
PREDICT command 364
PRINT command 367
PROFILE command 369
QUIT command 371
RANDOM command 372
RCOMMAND command 375
REFRESH command 382
RENAME command 386
REPORT command 388

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RETRIEVE command ... 392
SAMPLE command ... 394
SAVE command ... 402
SAVE LAYOUT command ... 404
SAVE LOG command ... 408
SAVE TABLELIST command ... 410
SAVE WORKSPACE command ... 412
SEEK command ... 413
SEQUENCE command ... 415
SET command ... 418
SIZE command ... 428
SORT command ... 433
STATISTICS command ... 438
STRATIFY command ... 442
SUMMARIZE command ... 447
TOP command ... 454
TOTAL command ... 455
TRAIN command ... 457
VERIFY command ... 462
Functions ... **465**
ABS( ) function ... 466
AGE( ) function ... 467
ALLTRIM( ) function ... 473
ASCII( ) function ... 475
AT( ) function ... 477
BETWEEN( ) function ... 480
BINTOSTR( ) function ... 488
BIT( ) function ... 490
BLANKS( ) function ... 492
BYTE( ) function ... 494
CDOW( ) function ... 496
CHR( ) function ... 499
CLEAN( ) function ... 501

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CMOY( ) function 503
COS( ) function 506
CTOD( ) function 508
CTODT( ) function 515
CTOT( ) function 521
CUMIPMT( ) function 525
CUMPRINC( ) function 527
DATE( ) function 529
DATETIME( ) function 533
DAY( ) function 537
DBYTE( ) function 540
DEC( ) function 542
DHEX( ) function 545
DICECOEFFICIENT( ) function 547
DIGIT( ) function 553
DOW( ) function 555
DTOU( ) function 558
EBCDIC( ) function 561
EFFECTIVE( ) function 563
EOMONTH( ) function 565
EXCLUDE( ) function 568
EXP( ) function 570
FILESIZE( ) function 572
FIND( ) function 574
FINDMULTI( ) function 578
FREQUENCY( ) function 582
FTYPE( ) function 584
FVANNUITY( ) function 587
FVLUMPSUM( ) function 591
FVSCHEDULE( ) function 594
GETOPTIONS( ) function 596
GOMONTH( ) function 598
HASH( ) function 601

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

HEX( ) function 606
HOUR( ) function 608
HTOU( ) function 610
INCLUDE( ) function 612
INSERT( ) function 614
INT( ) function 616
IPMT( ) function 617
ISBLANK( ) function 619
ISDEFINED( ) function 621
ISFUZZYDUP( ) function 623
LAST( ) function 627
LEADING( ) function 629
LENGTH( ) function 631
LEVDIST( ) function 633
LOG( ) function 637
LOWER( ) function 639
LTRIM( ) function 641
MAP( ) function 643
MASK( ) function 647
MATCH( ) function 649
MAXIMUM( ) function 656
MINIMUM( ) function 659
MINUTE( ) function 662
MOD( ) function 665
MONTH( ) function 667
NOMINAL( ) function 670
NORMDIST( ) function 672
NORMSINV( ) function 674
NOW( ) function 675
NPER( ) function 676
OCCURS( ) function 679
OFFSET( ) function 681
OMIT( ) function 683

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PACKED( ) function 687
PI( ) function 689
PMT( ) function 691
PPMT( ) function 694
PROPER( ) function 696
PROPERTIES( ) function 698
PVANNUITY( ) function 702
PVLUMPSUM( ) function 705
PYDATE( ) function 708
PYDATETIME( ) function 710
PYLOGICAL( ) function 712
PYNUMERIC( ) function 714
PYSTRING( ) function 716
PYTIME( ) function 718
RAND( ) function 720
RATE( ) function 722
RDATE( ) function 725
RDATETIME( ) function 728
RECLEN( ) function 731
RECNO( ) function 732
RECOFFSET( ) function 734
REGEXFIND( ) function 736
REGEXREPLACE( ) function 743
REMOVE( ) function 751
REPEAT( ) function 754
REPLACE( ) function 756
REVERSE( ) function 759
RJUSTIFY( ) function 760
RLOGICAL( ) function 761
RNUMERIC( ) function 764
ROOT( ) function 767
ROUND( ) function 769
RSTRING( ) function 771

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RTIME( ) function 774
SECOND( ) function 777
SHIFT( ) function 779
SIN( ) function 781
SOUNDEX( ) function 783
SOUNDSLIKE( ) function 787
SPLIT( ) function 790
STOD( ) function 794
STODT( ) function 798
STOT( ) function 802
STRING( ) function 806
SUBSTR( ) function 810
TAN( ) function 813
TEST( ) function 815
TIME( ) function 817
TODAY( ) function 822
TRANSFORM( ) function 823
TRIM( ) function 825
UNSIGNED( ) function 827
UPPER( ) function 829
UTOD( ) function 831
VALUE( ) function 835
VERIFY( ) function 838
WORKDAY( ) function 840
YEAR( ) function 844
ZONED( ) function 846
ZSTAT( ) function 850
Analytics **853**
Analytic scripts 854
Developing analytic scripts 857
Working with analytic headers 863
Analytic development best practices 871
Packaging analysis apps 877

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Sample analytic scripts (analysis app) ... 880
Analytic headers and tags ... 884
ANALYTIC ... 887
FILE ... 889
TABLE ... 892
FIELD ... 894
PARAM ... 896
PASSWORD ... 908
DATA ... 911
RESULT ... 915
PUBLISH ... 920
Appendix ... **921**
System requirements ... 922
Installing ACL for Windows ... 923
Unicode versus non-Unicode editions ... 926
Converting analytic scripts to Unicode ... 927
Checking for Unicode compatibility ... 930
Running R scripts on AX Server ... 932
Running Python scripts on AX Server ... 936
Analytic engine error codes ... 940
Variables created by Analytics commands ... 947
Reserved keywords ... 951

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Getting started

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics scripting basics

ACLScript is a command language that allows you to program and automate Analytics commands. The structure and components of ACLScript are simple yet powerful.

> **Note**
>
> If you are completely new to scripting, consider visiting the Academy for some basic training before jumping into this content. For courses on scripting and using Analytics, visit www.highbond.com.

Commands

Every line in a script executes an ACLScript command and starts with the command name. A command is an instruction to execute an operation in Analytics.

The command name is followed by one or more parameters that are specified as *parameter_name parameter_value*.

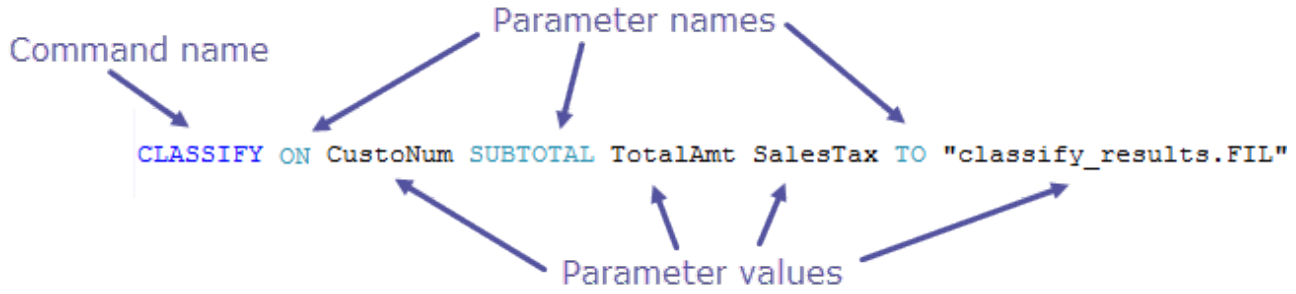
> **Tip**
>
> Depending on the command, some parameters are required and some are optional. You do not need to specify optional parameters. If they are omitted, the command executes without them. However, if you omit a required parameter, Analytics uses the default value for that parameter.

Example using the CLASSIFY command

The following example shows the CLASSIFY command along with following parameters:

- ON - specifies which field of the table to classify on
- SUBTOTAL - specifies optional fields to subtotal in the output
- TO - specifies the table to write the results of the CLASSIFY command to

Notice how each parameter is followed by one or more parameter values:



Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Important command syntax notes

- the first word in a script line must be a command name
- for most commands, the order of parameters that follow the command name does not matter
- most commands require that you open the target table before executing the command, precede these commands with OPEN *table_name*

Comments

Like any scripting language, you can add comments in ACLScript With the COMMENT keyword. Use comments to make your code easier to understand and to communicate with anyone who may try to read, use, or understand your script. ACLScript supports two types of comments:

- **single line comments** – all text following COMMENT is ignored until the end of the line is reached
- **multiple line comment blocks** – begin with COMMENT and each subsequent line is ignored until the END keyword, or a blank line, is reached

For more information and examples, see "Comments" on page 18.

Data types

ACLScript supports four basic data types:

- **logical** – the simplest data type. Logical data expresses a truth value of either true or false
- **numeric** – contain digits from 0 to 9 and, optionally, a negative sign and a decimal point
- **character** – a series of one or more characters
- **datetime** – a date, datetime, or time value expressed in a supported format

Each data type is treated differently by Analytics and can be used in different commands and functions. For more information about data types, see "Data types" on page 20.

Expressions

An expression is any statement that has a value. The simplest form of expression is a literal such as 2 or "test", however expressions usually appear as calculations and can be as complex as any valid combination of operators, conditions, functions, and values that you can imagine:

```
((2 + (3 - 2)) * 2) > ROOT(9,0)
```

Expressions are typically used in Analytics to populate computed fields or as input for conditional logic. For more information about expressions, see "Expressions" on page 21.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Functions

Functions are built-in routines that accept a given number of parameters and return a single value. Use functions to manipulate field contents and variables that are used in commands.

> **Note**
>
> Functions do not modify field data, functions generate and return a new value based on a calculation or algorithm that uses field data or variables as input. Use the value the function returns as input for a command.

Functions start with the function name followed directly by an opening parenthesis, a comma-separated list of 0 or more values that are passed into the function as arguments, and a closing parenthesis.

Example

The BETWEEN(*value*, *min*, *max*) function takes three arguments and returns true if the value falls within the range or false if it falls outside the range:

- value - the expression or field to test
- min - the minimum of the range
- max - the maximum of the range

```
BETWEEN(amount, 500, 5000)
```

For more information about functions, see "Functions" on page 25.

Variables

A variable is temporary storage location used to hold a value. Variables have an associated identifier that lets you reference and work with the value stored in your computer's memory.

ACLScript uses the ASSIGN command to create a variable and assign it a value at the same time:

```
ASSIGN v_age_in_years = 3
```

For simplicity you can omit the ASSIGN keyword, however ASSIGN is implicitly used and the same command runs:

```
v_age_in_years = 3
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> ACLScript does not support null values. All variables must have an associated value of one of the supported data types. The script interpreter evaluates the data type using the data format and qualifier you use to assign the value. For more information, see "Data types" on page 20.

Using variables

Once a variable is created, you can reference it anywhere you reference field names or variables. You can also reassign it a new value using the ASSIGN command.

```
EXTRACT RECORD TO 'result.fil' IF age > v_age_in_years
v_age_in_years = 5
```

You can also use string interpolation, or variable substitution, to include a variable in a string literal by wrapping the variable name in % characters. When Analytics encounters the substituted variable, it replaces the placeholder with its corresponding value:

```
ASSIGN v_table = erp_data
OPEN %v_table%
```

For more information about variables, see "Variables" on page 27.

Control structures

A control structure is a component of a script that decides which direction to take based on given parameters. ACLScript provides both conditional logic and looping structures.

Conditional logic

ACLScript implements conditional logic as an IF command and as an optional parameter on many commands in the language.

> **Tip**
>
> You use the IF command to control if a command runs or not while you use the IF parameter to decide which records in a table a command runs against.

IF command

```
IF v_counter > 10 CLASSIFY ON customer_no
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IF parameter

```
CLASSIFY ON customer_no IF state = 'NY'
```

Looping

The LOOP command provides the looping control structure in ACLScript. This command processes the statements inside the loop for as long as the control test expression evaluates to true.

For more information about control structures, see "Control structures" on page 29

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Comments

Like an scripting language, you can add comments in ACLScript With the COMMENT keyword. Use comments to make your code easier to understand and to communicate with anyone who may try to read, use, or understand your script.

Comment types

ACLScript supports two types of comments:

- **single line comments** – all text following COMMENT is ignored until the end of the line is reached
- **multiple line comment blocks** – begin with COMMENT and each subsequent line is ignored until the END keyword, or a blank line, is reached

Single line comments

Use single line comments to describe individual steps in your script or to describe variables:

```
COMMENT *** the start date for the analysis period
v_Start_Date = `20150101`
```

Multiple line comment blocks

Use multiple line comment blocks to describe scripts or script sections.

```
COMMENT
**********************
** This section of the script prepares data for import
**********************
END
```

Header comment blocks

It is good practice to include a header comment block that contains key script information at the start of each script:

```
COMMENT
************************************************************
*** Script Name:   {App_ID}{Script name}
*** Parameters:    {Detailed description}
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
*** Output:      {Describe parameters}
*** Written By:   {Name}, ABC Corporation, {Month YYYY}
*** Modified By:  {Name}, ABC Corporation, script purpose and logic
*** Version:     1.1.1 {app_ver.script_ver.defect.fix}
************************************************************
END
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Data types

ACLScript supports four basic data types: logical, numeric, character, and datetime.

Type	Description	Limit	Qualifier	Examples
Character	A series of one or more characters.	32,767 bytes	Single quotation marks, or double quotation marks	◦ 'John Doe' ◦ "John Doe"
Numeric	Numeric values contain digits from 0 to 9 and, optionally, a negative sign and a decimal point.	22 digits	No qualifier	◦ 100 ◦ -5 ◦ 5.01 ◦ 22222.1232
Datetime	A date, datetime, or time value expressed in a supported format.	◦ Minimum = 1900-01-01 ◦ Maximum = 9999-12-31	◦ Backquotes ◦ Leading 't', or a single blank space, for time values	◦ `20160101` ◦ `141231` ◦ `t2359` ◦ `20141231T23595-9` ◦ `20141231 235959`
Logical	The simplest data type. Logical data expresses a truth value of either true or false. Comparison operators such as '=', '>', and '<' return logical values.	◦ T ◦ F	No qualifier	`ASSIGN v_truth = 5 > 4` evaluates to T

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Expressions

An expression is any statement that has a value. The simplest form of expression is a literal, however expressions can be as complex as any legal combination of operators, conditions, functions, and values you can imagine.

Expression components

Literal values

A literal value is a value written exactly as it is meant to be interpreted, such as the character literal 'my value'. For information about literals, see "Data types" on the previous page.

Operators

Operators are symbols that tell the script interpreter to perform arithmetic, string, comparison, or logical evaluation of the specified values:

Operator type in order of precedence	Operators in order of precedence	Examples
Parenthesis	◦ () specifies precedence ◦ () function operator	(5 + 3) * 2
Unary	◦ NOT logical ◦ - negation	v_truth = NOT (3 < 2)
Arithmetic	◦ ^ exponentiation ◦ * multiplies, / divides ◦ + adds, - subtracts **Note** Multiplicative operators have equal precedence with each other and evaluate from left to right. Additive operators have equal precedence with each other and evaluate from left to right.	1 + 5 - 3 * 2
String	+ concatenates	"This is" + " my script"
Comparative	◦ < less than ◦ > greater than ◦ = equality ◦ >= greater than or equal to	IF amount <> 100

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Operator type in order of precedence	Operators in order of precedence	Examples
	◦ <= less than or equal to ◦ <> not equal **Note** Comparative operators have equal precedence with each other and evaluate from left to right.	
Binary logical	◦ AND or & ◦ OR or |	IF amount > 5 AND amount < 10

Functions

Expressions are evaluated using the values returned by functions. Functions execute with the highest precedence of any expression component. For more information about functions, see "Functions" on page 25.

Example expressions

Evaluates to 6

```
(2 + (3 - 2)) * 2
```

Evaluates to true

```
((2 + (3 - 2)) * 2) > ROOT(9,0)
```

Evaluates to 'ACLScript tutorial'

```
'AC' + 'LScri' + 'pt ' + 'tutorial'
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Defining computed fields with expressions

Use **computed fields** to create additional data fields in the currently open table with an expression. A computed field is a field that is appended to the open table and populated with the value of the specified expression.

Computed field syntax

```
DEFINE FIELD name COMPUTED expression
```

- **name** – the name of the computed field to generate
- **expression** – the computation or calculation used to generate the value for the field

Example computed field

```
DEFINE FIELD c_full_name COMPUTED first_name + ' ' + last_name
```

> **Tip**
>
> Prefix computed field names with `c_` to identify them as computed data rather than original source data.

Defining conditional computed field values

You can also use conditions with computed fields to define the value for different cases:

```
DEFINE FIELD c_total COMPUTED

 amount * ca_tax_rate IF state = 'CA'
 amount * ny_tax_rate IF state = 'NY' OR state = 'NJ'
amount * general_rate
```

When the first conditional expression evaluates to true, the value specified for that case is used. In this example, `amount * general_rate` is the default value used when neither of the conditional expressions evaluate to true.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> You must add an empty line between the line command and the conditions unless you include the IF , WIDTH, PIC, or AS parameters on the DEFINE FIELD command. For more information, see "DEFINE FIELD . . . COMPUTED command" on page 132.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Functions

Functions are built-in routines that accept a given number of parameters and return a single value. Use functions to manipulate field contents and variables that are used in commands.

> **Note**
>
> Functions do not modify field data, functions generate and return a new value based on a calculation or algorithm that uses field data or variables as input. Use the value the function returns as input for a command.

Function syntax

Functions start with the function name followed directly by an opening parenthesis, a comma-separated list of 0 or more values that are passed into the function as arguments, and a closing parenthesis.

Example

The BETWEEN(*value*, *min*, *max*) function takes three arguments and returns true if the value falls within the range or false if it falls outside the range:

- value - the expression or field to test
- min - the minimum of the range
- max - the maximum of the range

```
BETWEEN(amount, 500, 5000)
```

Function arguments

An argument of a function is a specific input value passed into the function.

Function arguments are passed to functions via an argument list. This is a comma-delimited list of literal values, variables, or expressions that evaluate to values of the parameter data type. For more information about working with data types, see "Data types" on page 20.

> **Note**
>
> If your project works with European number formats, or if you are writing scripts that are portable across regions, separate function arguments with a space character instead of a comma unless you are passing in a signed numeric value. Functions accepting signed numeric values require an explicit delimiter.

Functions vs commands

The distinction between commands and functions is subtle but critical to using ACLScript:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Functions	Commands
Use fields, values, or records as input and generate a new value that is returned.	Use tables as input and generate new records and tables.
Used in expressions, computed fields, command parameter values, variables, and filters to assist and modify command execution.	Used to analyze data, import data, and produce results.
Cannot be an independent step in a script.	Can be an independent step in a script.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Variables

A variable is temporary storage location used to hold a value. Variables have an associated identifier that lets you reference and work with the value stored in your computer's memory.

How variables work in ACLScript

Creating a variable and assigning a value

ACLScript uses the ASSIGN command to create a variable and assign it a value at the same time:

```
ASSIGN v_age_in_years = 3
```

For simplicity you can omit the ASSIGN keyword, however ASSIGN is implicitly used and the same command runs:

```
v_age_in_years = 3
```

> **Note**
>
> ACLScript does not support null values. All variables must have an associated value of one of the supported data types. The script interpreter evaluates the data type using the data format and qualifier you use to assign the value. For more information, see "Data types" on page 20.

Using variables

Once a variable is created, you can reference it anywhere you reference field names or variables. You can also reassign it a new value using the ASSIGN command.

```
EXTRACT RECORD TO 'result.fil' IF age > v_age_in_years
v_age_in_years = 5
```

You can also use string interpolation, or variable substitution, to include a variable in a string literal by wrapping the variable name in % characters. When Analytics encounters the substituted variable, it replaces the placeholder with its corresponding value:

```
ASSIGN v_table = erp_data
OPEN %v_table%
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Types of variables

Analytics uses the following types of variables:

- **system-generated variables** – automatically created after executing a command
- **permanent variables** – remain in your computer's memory until you delete them and persist after closing the Analytics project

  > **Note**
  >
  > To define a permanent variable, prefix the identifier with an '_': `_v_company_name = 'Acme'`.

- **session variables** – remain in your computer's memory until you delete them or until the Analytics project is closed

Variable identifiers

Variable identifiers are case-insensitive and follow certain conventions related to the type of variable:

- system-generated variable identifiers use all caps: `OUTPUTFOLDER`
- permanent variable identifiers must have a '_' prefix: `_v_permanent`
- session variable identifiers use the format `v_`*varname* by convention but you are not restricted to this naming convention

Viewing variable values

During script development or while debugging, it can be useful to track variable values as the script executes. To capture variable values in the script log file, use the `DISPLAY` command:

```
DISPLAY v_age_in_years
```

When the script encounters this command, it writes the command to the log file. To view the variable value at that stage of script execution, click the entry in the log.

> **Tip**
>
> You can also use variables to help debug by inserting breakpoints in your script and inspecting the variable values on the **Variables** tab of the **Navigator**.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Control structures

A control structure is a component of a script that decides which direction to take based on given parameters. ACLScript provides both conditional IF logic and looping structures.

Conditional logic using IF

ACLScript implements conditional logic as an IF command and as an optional parameter on many commands in the language:

- **command** - controls whether or not a command runs
- **parameter** - decides which records in a table to execute the command against

The IF command

When using the IF command, you specify a conditional expression followed by the command to execute if the expression evaluates to true:

```
IF v_counter > 10 CLASSIFY ON customer_no
```

This conditional structure controls which code executes, so you can use the IF command when you want to process an entire table based on the test expression. If the expression evaluates as true, the command is run against all records in the table. For more information about the IF command, see "IF command" on page 238.

IF parameter

Many commands accept an optional IF parameter that you can use to filter which records the command is executed against:

```
CLASSIFY ON customer_no IF state = 'NY'
```

When this statement executes, the script classifies all records in the table where the value of the state field is 'NY'.

Looping

The LOOP command

The LOOP command provides the looping control structure in ACLScript.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> The LOOP command must execute within the GROUP command, it cannot standalone.

This command processes the statements inside the loop for as long as the specified WHILE expression is true:

```
ASSIGN v_counter = 10
GROUP
 LOOP WHILE v_counter > 0
  v_total = v_total + amount
  v_counter = v_counter - 1
 END
END
```

This structure iterates 10 times and adds the value of the amount field to the variable v_total. At the end of each iteration, the v_counter variable is decremented by 1 and then tested in the WHILE expression. Once the expression evaluates to false, the loop completes and the script progresses.

When the loop completes, v_total holds the sum of the 10 records' amount fields.

For more information about looping, see "LOOP command" on page 338.

LOOPING with a subscript

Sometimes the LOOP command does not provide the exact looping functionality you may require. In this case, you can also call a separate Analytics script to execute a loop using the DO SCRIPT command: DO SCRIPT *scriptName* WHILE *conditionalTest*.

You can use one of the following common methods to control when your loop ends:

- **flag** – the loop continues until the logical flag variable is set to FALSE
- **counter** – the loop continues until an incrementing or decrementing variable crosses a conditional threshold

For more information about calling subscripts, see "DO SCRIPT command" on page 170.

Example

You need to import all the CSV files in the `C:\data` folder into your project. You can use the DIRECTORY command to get a list of files from the folder, however you cannot use the IMPORT command inside the GROUP structure. You need an alternative way of looping through the table that DIRECTORY creates.

To achieve this, you create a main script that:

1. Executes the DIRECTORY command and saves the results to a table.
2. Gets the number of records in the table to use as a counter.
3. Calls a subscript once per record in the table to execute the IMPORT command against the current record.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Main script

```
COMMENT Main script

DIRECTORY "C:\data\*.csv" TO T_Table_To_Loop
OPEN T_Table_To_Loop
COUNT
v_Num_Records = COUNT1
v_Counter = 1
DO SCRIPT Import_Subscript WHILE v_Counter <= v_Num_Records
```

Import subscript

```
COMMENT Import_Subscript

OPEN T_Table_To_Loop
LOCATE RECORD v_Counter

COMMENT code to import CSV file in record goes here...

ASSIGN v_Counter = v_Counter + 1
```

Variables are shared among all scripts that run in the project, so the main script calls the subscript until the value of `v_Counter` exceeds the value of `v_Num_Records`. Each time the subscript executes, it increments `v_Counter`.

This structure allows you to call the `IMPORT` command against each record while looping through the table. When the main script completes, you have imported all CSV files from the `C:\data` folder.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Grouping and looping

The GROUP and LOOP commands provide two ways to execute a series of commands repeatedly. GROUP performs a single iteration of one or more commands against each record. LOOP performs multiple iterations of a series of commands against a single record, and can only be used inside a GROUP block.

A simple example of GROUP

You have a table of invoice data called **Ap_Trans**. Using this data, you need to calculate a running total of invoice amounts:

Vendor_ Number	Vendor_Name	Vendor_ City	Invoice_ Number	Invoice_ Date	Invoice_ Amount	Quantity	Unit_ Cost
11663	More Power Industries	Los Angeles	5981807	2000-11-17	618.30	90	6.87
13808	NOVATECH Wholesale	Des Moines	2275301	2000-11-17	6705.12	976	6.87
12433	Koro International	Sheveport	6585673	2000-11-17	7955.46	1158	6.87

To calculate this amount, you use the GROUP command. Inside each iteration of GROUP, you:

1. Calculate the running total as of the current record.
2. Extract the invoice number, amount, date, and running total to a results table.

```
OPEN Ap_Trans

COMMENT set the initial value of running total to zero
ASSIGN v_running_total = 0.00

COMMENT iterate over each record in the table and then calculate and extract the running total
GROUP
 ASSIGN v_running_total = v_running_total + Invoice_Amount
 EXTRACT Invoice_Number, Invoice_Amount, Invoice_Date, v_running_total AS "Running total" TO
results1
END
```

When the script runs, the commands inside the GROUP block are processed against each record in the table, from top to bottom, and the running total is calculated and extracted. If we could walk through GROUP as it runs, this is how it would look:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

First iteration of GROUP: running total = 0.00 + 618.30

The GROUP adds the invoice amount of the first record to the initial running total of 0.00 and extracts the fields to the results table:

Vendor_ Number	Vendor_Name	Vendor_ City	Invoice_ Number	Invoice_ Date	Invoice_ Amount	Quantity	Unit_ Cost
11663	More Power Industries	Los Angeles	5981807	2000-11-17	618.30	90	6.87
13808	NOVATECH Wholesale	Des Moines	2275301	2000-11-17	6705.12	976	6.87
12433	Koro International	Sheveport	6585673	2000-11-17	7955.46	1158	6.87

Second iteration of GROUP: running total = 618.30 + 6705.12

The GROUP block adds the invoice amount of the second record to the new running total of 618.30 and extracts the fields to the results table:

Vendor_ Number	Vendor_Name	Vendor_ City	Invoice_ Number	Invoice_ Date	Invoice_ Amount	Quantity	Unit_ Cost
11663	More Power Industries	Los Angeles	5981807	2000-11-17	618.30	90	6.87
13808	NOVATECH Wholesale	Des Moines	2275301	2000-11-17	6705.12	976	6.87
12433	Koro International	Sheveport	6585673	2000-11-17	7955.46	1158	6.87

Third iteration of GROUP: running total = 7323.42 + 7955.46

The GROUP block adds the invoice amount of the third record to the new running total of 7323.42 and extracts the fields to the results table:

Vendor_ Number	Vendor_Name	Vendor_ City	Invoice_ Number	Invoice_ Date	Invoice_ Amount	Quantity	Unit_ Cost
11663	More Power Industries	Los Angeles	5981807	2000-11-17	618.30	90	6.87

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Vendor_ Number	Vendor_Name	Vendor_ City	Invoice_ Number	Invoice_ Date	Invoice_ Amount	Quantity	Unit_ Cost
13808	NOVATECH Wholesale	Des Moines	2275301	2000-11-17	6705.12	976	6.87
12433	Koro International	Sheveport	6585673	2000-11-17	7955.46	1158	6.87

Final results table

After GROUP has processed the final record in the table, you have the following results table:

Invoice_Number	Invoice_Amount	Invoice_Date	Running_total
5981807	618.30	2000-11-17	618.30
2275301	6705.12	2000-11-17	7323.42
6585673	7955.46	2000-11-17	15278.88

Handling different cases using GROUP IF

Using the same **AP_Trans** table as above, you now need to calculate running totals for three types of invoices:

- High value (greater than or equal to 1000.00)
- Medium value (between 100.00 and 1000.00)
- Low value (less than 100.00)

The GROUP command provides an IF/ELSE structure to handle different cases. You provide the conditional expressions to test, and if a record evaluates to true, then the commands inside the block run.

How cases are tested

Cases are tested **from top to bottom**, and a record can only be processed by one IF/ELSE block. The first case that evaluates to true for the record is the one that processes the record:

1. When GROUP processes the first record, it tests it against the first IF condition (`Invoice_Amount >= 1000`). If this evaluates to true, then the code for this case runs and no other cases are tested.
2. If the first case evaluates to false, then the next ELSE IF condition (`Invoice_Amount >= 100`) is tested. Likewise, if this test evaluates to true, then the code for this case runs and no other cases are tested.
3. Finally, if none of the IF or ELSE IF cases evaluate to true, then the default case in the ELSE block processes the record.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> If a record evaluates to true for more than one case, the record is only processed by the first IF/ELSE block that tests it. Records are never processed by more than one IF/ELSE block in a GROUP command.

```
OPEN Ap_Trans

COMMENT set initial values for running totals
ASSIGN v_running_total_hi = 0.00
ASSIGN v_running_total_med = 0.00
ASSIGN v_running_total_low = 0.00

COMMENT use GROUP IF to run different ASSIGN and EXTRACT commands depending on invoice
amount
GROUP IF Invoice_Amount >= 1000
 ASSIGN v_running_total_hi = v_running_total_hi + Invoice_Amount
 EXTRACT Invoice_Number, Invoice_Amount, Invoice_Date, v_running_total_hi AS "Running total"
TO results_hi
ELSE IF Invoice_Amount >= 100
 ASSIGN v_running_total_med = v_running_total_med + Invoice_Amount
 EXTRACT Invoice_Number, Invoice_Amount, Invoice_Date, v_running_total_med AS "Running total"
TO results_med
ELSE
 ASSIGN v_running_total_low = v_running_total_low + Invoice_Amount
 EXTRACT Invoice_Number, Invoice_Amount, Invoice_Date, v_running_total_low AS "Running total"
TO results_low
END
```

When the script runs, the GROUP command tests the invoice amount for each record. Depending on the amount, the record is used to update one of three running totals (low, medium, high) and three result tables are produced.

LOOP inside a GROUP

When using GROUP to process the records in a table, you can use a LOOP command to execute a series of commands on a single record multiple times. LOOP is a second iteration that happens inside the iteration of GROUP, and it runs until a test condition that you specify evaluates to false.

Using the LOOP to split a field

You have the following table containing invoice data and you need to isolate specific information for invoice amounts per department. One invoice may be related to more than one department, and department codes are stored in comma-delimited format in the table:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Vendor_Number	Invoice_Number	Invoice_Date	Invoice_Amount	Department_Code
11663	5981807	2000-11-17	618.30	CCD,RDR
13808	2275301	2000-11-17	6705.12	CCD
12433	6585673	2000-11-17	7955.46	CCD,LMO,RDR

To extract the invoice amounts per department, you:

1. Use a GROUP command to process the table record by record.
2. Calculate the number of departments (*n*) associated with each record.
3. Use the LOOP command to iterate *n* times over the record to extract data for each department associated with the record.

> **Note**
>
> You must increment the v_counter variable inside LOOP. If you do not, the WHILE test always evaluates to true and the script enters an infinite loop. You can include a SET LOOP command in your scripts to guard against infinite loops. For more information, see "SET command" on page 418.

```
COMMENT
use GROUP to count commas in each department code field as a way of identifying how many depart-
ments are associated with the record
"LOOP" over each record for each code in the field, extracting each code into its own record
END
GROUP
 v_department_count = OCCURS(Department_Code,',')
 v_counter = 0
 LOOP WHILE v_counter <= v_department_count
  v_dept = SPLIT(Department_Code, ',', (v_counter + 1))
  EXTRACT FIELDS Invoice_Number, Invoice_Amount, v_dept AS "Department" TO result1
  v_counter = v_counter + 1
 END
END
```

When the script runs, the commands inside the GROUP block are processed against each record in the table, from top to bottom. For each record, the LOOP command iterates over the record once per department code in the comma-delimited list and then extracts a record. If we could walk through GROUP and LOOP as they run, this is how it would look:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

First iteration of GROUP: 2 iterations of LOOP

Vendor_Number	Invoice_Number	Invoice_Date	Invoice_Amount	Department_Code
11663	5981807	2000-11-17	618.30	CCD,RDR
13808	2275301	2000-11-17	6705.12	CCD
12433	6585673	2000-11-17	7955.46	CCD,LMO,RDR

For the first record in the table, the value of v_department_count is 1, so LOOP iterates twice:

1. For the first iteration of the LOOP:
   - v_counter = 0
   - v_depart = CCD

   The following record is extracted and the value of v_counter is incremented to 1, therefore LOOP iterates again:

5981807	618.30	CCD

2. For the second iteration of LOOP:
   - v_counter = 1
   - v_depart = RDR

   The following record is extracted and the value of v_counter is incremented to 2, therefore LOOP does not iterate again and GROUP proceeds to the next record:

5981807	618.30	RDR

Second iteration of GROUP: 1 iteration of LOOP

Vendor_Number	Invoice_Number	Invoice_Date	Invoice_Amount	Department_Code
11663	5981807	2000-11-17	618.30	CCD,RDR
13808	2275301	2000-11-17	6705.12	CCD
12433	6585673	2000-11-17	7955.46	CCD,LMO,RDR

For the second record in the table, the value of v_department_count is 0, so LOOP iterates once:

- v_counter = 0
- v_depart = CCD

The following record is extracted and the value of v_counter is incremented to 1, therefore LOOP does not iterate again and GROUP proceeds to the next record:

2275301	6705.12	CCD

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Third iteration of GROUP: 3 iterations of LOOP

Vendor_Number	Invoice_Number	Invoice_Date	Invoice_Amount	Department_Code
11663	5981807	2000-11-17	618.30	CCD,RDR
13808	2275301	2000-11-17	6705.12	CCD
12433	6585673	2000-11-17	7955.46	CCD,LMO,RDR

For the third record in the table, the value of v_department_count is 2, so LOOP iterates three times:

1. For the first iteration of LOOP:
   - v_counter = 0
   - v_depart = CCD

   The following record is extracted and the value of v_counter is incremented to 1, therefore LOOP iterates again:

6585673	7955.46	CCD

2. For the second iteration of LOOP:
   - v_counter = 1
   - v_depart = LMO

   The following record is extracted and the value of v_counter is incremented to 2, therefore LOOP iterates again:

6585673	7955.46	LMO

3. For the third iteration of LOOP:
   - v_counter = 2
   - v_depart = RDR

   The following record is extracted and the value of v_counter is incremented to 3, therefore LOOP does not iterate again and GROUP reaches the end of the table:

6585673	7955.46	RDR

Final results table

After GROUP has processed each record in the table, and LOOP has iterated though all the department codes, you have the following results table:

Invoice_Number	Invoice_Amount	Department
5981807	618.30	CCD
5981807	618.30	RDR

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Invoice_Number	Invoice_Amount	Department
2275301	6705.12	CCD
6585673	7955.46	CCD
6585673	7955.46	LMO
6585673	7955.46	RDR

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Top 30 Analytics functions

The top 30 functions in ACLScript are useful across a number of different tasks. Use these functions regularly to help you prepare, parse, convert, and harmonize data in your scripts.

Removing leading and trailing space

Character fields in Analytics tables often contain leading or trailing spaces because the field widths are fixed length. When you need to perform an operation using the data from a character field, you can remove these spaces so that the string only contains the actual data.

ALLTRIM( )

Returns a string with leading and trailing spaces removed from the input string.

> **Note**
>
> It is good practice to use ALLTRIM() on any character field that you are using as input for another function so that no leading or trailing spaces affect the returned value.

Example

The Vendor_Number field contains the value " 1254". You need to remove this extra space from Vendor_Number so that you can harmonize the field with data in another table.

```
COMMENT returns "1254"
ALLTRIM(Vendor_Number)
```

Synchronizing alphabetic case

String comparison in Analytics is case-sensitive, therefore it is useful to synchronize the casing of all data in a field before you perform any comparisons, joins, or relations using the data.

UPPER( )

Returns a string with alphabetic characters converted to uppercase.

Example

The Last_Name field contains the value "Smith". You need to make this value uppercase to compare with an uppercase value from another table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
COMMENT returns "SMITH"
UPPER(Last_Name)
```

LOWER( )

Returns a string with alphabetic characters converted to lowercase.

Example

The Last_Name field contains the value "Smith". You need to make this value lowercase to compare with an lowercase value from another table.

```
COMMENT returns "smith"
LOWER(Last_Name)
```

PROPER( )

Returns a string with the first character of each word set to uppercase and the remaining characters set to lowercase.

Example

The Last_Name field contains the value "smith". You need to display it as a proper noun in your output.

```
COMMENT returns "Smith"
PROPER(Last_Name)
```

Calculating and separating strings

When you need to extract a segment of data from a longer string, or test some information about the string such as its length or contents, use these functions.

SUBSTR( )

Returns a specified substring from a string.

Example

The GL_Account_Code field contains the value "001-458-873-99". You need to extract the first three bytes, or characters, from the string.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
COMMENT returns "001"
ASSIGN v_start_pos = 1
ASSIGN v_length = 3
SUBSTR(GL_Account_Code, v_start_pos, v_length)
```

LAST( )

Returns a specified number of characters from the end of a string.

Example

The GL_Account_Code field contains the value "001-458-873-99". You need to extract the last two bytes, or characters, from the string.

```
COMMENT returns "99"
ASSIGN v_num_chars = 2
LAST(GL_Account_Code, v_num_chars)
```

SPLIT( )

Returns a specified segment from a string.

Example

The GL_Account_Code field contains the value "001-458-873-99". You need to extract the second segment of the code from the string.

```
COMMENT returns "458"
ASSIGN v_delimiter = "-"
ASSIGN v_segment_num = 2
SPLIT(GL_Account_Code, v_delimiter, v_segment_num)
```

AT( )

Returns a number specifying the starting location of a particular occurrence of a substring within a character value.

Example

The GL_Account_Code field contains the value "001-458-873-99". You need to determine the starting byte position of the value "458" to test whether the GL code's second segment is "458" (start position "5").

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
COMMENT returns "5"
ASSIGN v_occurrence = 1
ASSIGN v_substring = "458"
AT(v_occurrence, v_substring, GL_Account_Code)
```

OCCURS( )

Returns a count of the number of times a substring occurs in a specified character value.

Example

The GL_Account_Code field contains the value "001-458-873-99". You need to determine that the GL code is correctly formatted by ensuring the data contains three hyphen characters.

```
COMMENT returns "3"
ASSIGN v_substring = "-"
OCCURS(GL_Account_Code, v_substring)
```

LENGTH( )

Returns the number of characters in a string.

Example

The GL_Account_Code field contains the value "001-458-873-99". You need to determine that the GL code is correctly formatted by ensuring the data contains 14 characters.

```
COMMENT returns "14"
LENGTH(GL_Account_Code)
```

Converting data types

Depending on the data source and import statements that produced the Analytics table, you may need to convert values in a field from one data type to another so that an operation is possible. For example, to perform arithmetic on data that was imported as character ("12345"), you must convert it to numeric.

STRING( )

Converts a numeric value to a character string.

Example

The Invoice_Amount field contains the value 12345.67. You need to convert this to character data.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
COMMENT returns "12345.67"
ASSIGN v_str_length = 8
STRING(Invoice_Amount, v_str_length)
```

VALUE( )

Converts a character string to a numeric value.

> **Tip**
>
> VALUE( ) is often used with ZONED( ) to add leading zeros.

Example

The Invoice_Amount field contains the value "12345.67". You need to convert this to numeric data.

```
COMMENT returns 12345.67
VALUE(Invoice_Amount, 2)
```

CTOD( )

Converts a character or numeric date value to a date. Can also extract the date from a character or numeric datetime value and return it as a date. Abbreviation for "Character to Date".

Example

The Submission_Date field contains the value "April 25, 2016". You need to convert this to datetime data.

```
COMMENT returns `20160425`
ASSIGN v_date_format = "mmm dd, yyyy"
CTOD(Submission_Date, v_date_format)
```

DATE( )

Extracts the date from a specified date or datetime and returns it as a character string. Can also return the current operating system date.

Example

The Submission_Date field contains the value `20160425`. You need to convert this to character data.

```
COMMENT returns "04/25/2016"
ASSIGN v_date_format = "MM/DD/YYYY"
DATE(Submission_Date, v_date_format)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Adding leading zeros

Convert numeric data to character data and adds leading zeros to the output when you need to harmonize fields that require leading zeros.

ZONED( )

Converts numeric data to character data and adds leading zeros to the output.

Example

The Employee_Number field contains the value "254879". You need to convert the value to a 10-digit string with leading zeros.

> **Tip**
>
> You must use the VALUE() function to convert the character to numeric data before using the numeric as input for ZONED().

```
COMMENT returns "0000254879"
ASSIGN v_str_length = 10
ASSIGN v_num_decimals = 0
ZONED(VALUE(Employee_Number, v_num_decimals), v_str_length)
```

BINTOSTR( )

Returns Unicode character data converted from ZONED or EBCDIC character data. Abbreviation for "Binary to String".

> **Note**
>
> Unicode edition only. For non-Unicode editions, see ZONED() above.

Example

The Employee_Number field contains the value "254879". You need to convert the value to a 10-digit string with leading zeros.

> **Tip**
>
> You must use the VALUE() function to convert the character to numeric data before using the numeric as input for ZONED(). You then use BINTOSTR() to convert the ASCII data returned from ZONED() to Unicode.

```
COMMENT returns "0000254879"
ASSIGN v_str_length = 10
ASSIGN v_num_decimals = 0
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
ASSIGN v_str_type = "A"
BINTOSTR(ZONED(VALUE(Employee_Number, v_num_decimals), v_str_length), v_str_type)
```

Extracting datetime parts

Use these functions to isolate and extract specific components of a datetime value.

MONTH( )

Extracts the month from a specified date or datetime and returns it as a numeric value (1 to 12).

Example

The Transaction_Date field contains the value `20160815 100252`. You need to extract the month as character data with a leading zero.

```
COMMENT returns "08"
ASSIGN v_str_length = 2
ZONED(MONTH(Transaction_Date), v_str_length)
```

DAY( )

Extracts the day of the month from a specified date or datetime and returns it as a numeric value (1 to 31).

Example

The Transaction_Date field contains the value `20160815 100252`. You need to extract the day as character data.

```
COMMENT returns "15"
ASSIGN v_str_length = 2
STRING(DAY(Transaction_Date), v_str_length)
```

YEAR( )

Extracts the year from a specified date or datetime and returns it as a numeric value using the YYYY format.

Example

The Transaction_Date field contains the value `20160815 100252`. You need to extract the year as a numeric value.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
COMMENT returns 2016
YEAR(Transaction_Date)
```

HOUR( )

Extracts the hour from a specified time or datetime and returns it as a numeric value using the 24-hour clock.

Example

The `Transaction_Date` field contains the value `` `20160815 100252` ``. You need to extract the hours as a numeric value.

```
COMMENT returns 10
HOUR(Transaction_Date)
```

MINUTE( )

Extracts the minutes from a specified time or datetime and returns it as a numeric value.

Example

The `Transaction_Date` field contains the value `` `20160815 100252` ``. You need to extract the minutes as a numeric value.

```
COMMENT returns 2
MINUTE(Transaction_Date)
```

SECOND( )

Extracts the seconds from a specified time or datetime and returns it as a numeric value.

Example

The `Transaction_Date` field contains the value `` `20160815 100252` ``. You need to extract the seconds as a numeric value.

```
COMMENT returns 52
SECOND(Transaction_Date)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CDOW( )

Returns the name of the day of the week for a specified date or datetime. Abbreviation for "Character Day of Week".

Example

The Transaction_Date field contains the value `20160815 100252`. You need to extract the name of the day as character data.

```
COMMENT returns "Mon"
CDOW(Transaction_Date, 3)
```

CMOY( )

Returns the name of the month of the year for a specified date or datetime. Abbreviation for "Character Month of Year".

Example

The Transaction_Date field contains the value `20160815 100252`. You need to extract the name of the month as character data.

```
COMMENT returns "Aug"
CMOY(Transaction_Date, 3)
```

Manipulating strings

Remove or replace segments of character fields using these functions.

INCLUDE( )

Returns a string that includes only the specified characters.

Example

The Address field contains the value "12345 ABC Corporation". You need to extract the address number and exclude the name of the company.

```
COMMENT returns "12345"
ASSIGN v_chars_to_return = "0123456789"
INCLUDE(Address, v_chars_to_return)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EXCLUDE( )

Returns a string that excludes the specified characters.

Example

The Address field contains the value "12345 ABC Corporation". You need to extract the name of the company and exclude the address number.

```
COMMENT returns "ABC Corporation"
ASSIGN v_chars_to_exclude = "0123456789"
EXCLUDE(Address, v_chars_to_exclude)
```

REPLACE( )

Replaces all instances of a specified character string with a new character string.

Example

The Address field contains the value "12345 Acme&Sons". You need to replace the "&" character with the word " and ".

```
COMMENT returns "12345 Acme and Sons"
ASSIGN v_target_char = "&"
ASSIGN v_replacement_char = " and "
REPLACE(Address, v_target_char, v_replacement_char)
```

OMIT( )

Returns a string with one or more specified substrings removed.

Example

The Address field contains the value "12345 Fake St". You need to extract the address without the street suffix.

```
COMMENT returns "12345 Fake"
ASSIGN v_chars_to_omit = "St"
OMIT(Address, v_chars_to_omit)
```

REVERSE( )

Returns a string with the characters in reverse order.

Example

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

The Report_Line field contains the value "001 Correction 5874.39 CR ". You need to reverse the value and omit any leading or trailing spaces.

```
COMMENT returns "RC 93.4785 noitcerroC 100"
REVERSE(ALLTRIM(Report_Line))
```

BLANKS( )

Returns a string containing a specified number of blank spaces.

Example

You need to create a computed field for a region name based on a value in the region_code field. You must ensure that the default value you specify at the end of the command is at least as long as the longest input value.

```
COMMENT BLANKS returns a string of 8 " " chars
ASSIGN v_length = 8
DEFINE FIELD region COMPUTED

"Southern" IF region_code = 1
"Northern" IF region_code = 2
"Eastern" IF region_code = 3
"Western" IF region_code = 4
BLANKS(v_length)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Commands

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Import and export data

The import commands let you import data from a variety of different data sources.

Depending on the data source, you also define the source data as part of importing it. Defining data means specifying attributes such as field names, field lengths, and field data type.

The export command lets you export data to a variety of different file formats, or to Results in HighBond.

Data access by Analytics is read-only

When connecting to any data source, or importing from any data source, Analytics is strictly read-only. Analytics cannot add, update, or delete data in a data source, or modify a data source in any way. This restriction applies to all data sources accessible by Analytics: file-based data sources, databases, and cloud data services.

Analytics data files (.fil) created from imported data are also treated as read-only by Analytics. Analytics cannot alter .fil files, with the exception of refreshing the file from the data source.

.fil files are completely separate from the data source used to create them. Deleting a .fil file has no effect on the data source.

Command descriptions

Command	Description
ACCESSDATA	Imports data from a variety of ODBC-compliant data sources. The command takes the form ACCESSDATA64 or ACCESSDATA32 depending on whether you are using a 64-bit or 32-bit ODBC driver.
DEFINE TABLE DB	Defines an Analytics server table by connecting to a database table using AX Connector. You can connect to a Microsoft SQL Server, Oracle, or DB2 database.
EXPORT	Exports data from Analytics to the specified file format, or to HighBond Results.
IMPORT ACCESS	Creates an Analytics table by defining and importing a Microsoft Access database file.
IMPORT DELIMITED	Creates an Analytics table by defining and importing a delimited text file.
IMPORT EXCEL	Creates an Analytics table by defining and importing a Microsoft Excel worksheet or named range.
IMPORT GRCPROJECT	Creates an Analytics table by importing a HighBond Projects table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Command	Description
IMPORT GRCRESULTS	Creates an Analytics table by importing a HighBond Results table or interpretation.
IMPORT MULTIDELIMITED	Creates multiple Analytics tables by defining and importing multiple delimited files.
IMPORT MULTIEXCEL	Creates multiple Analytics tables by defining and importing multiple Microsoft Excel worksheets or named ranges.
IMPORT ODBC	Creates an Analytics table by defining and importing data from an ODBC data source. ODBC stands for Open Database Connectivity, a standard method for accessing databases.
IMPORT PDF	Creates an Analytics table by defining and importing an Adobe PDF file.
IMPORT PRINT	Creates an Analytics table by defining and importing a Print Image (Report) file.
IMPORT SAP	Creates an Analytics table by importing data from an SAP system using *Direct Link*.
IMPORT XBRL	Creates an Analytics table by defining and importing an XBRL file.
IMPORT XML	Creates an Analytics table by defining and importing an XML file.
RETRIEVE	Retrieves the result of a *Direct Link* query submitted for background processing.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ACCEPT command

Creates a dialog box that interactively prompts users for one or more script input values. Each input value is stored in a named character variable.

> **Note**
>
> Using the ACCEPT command to enter passwords is not secure. You should use the "PASSWORD command" on page 360 instead.
>
> The ACCEPT command is not supported in AX Server analytics.
>
> You can create a more advanced interactive dialog box with the "DIALOG command" on page 151.

Syntax

```
ACCEPT {message_text <FIELDS project_item_category> TO variable_name} <...n>
```

Parameters

Name	Description
message_text	The label displayed in the dialog box used to prompt for input. Must be a quoted string or a character variable. When entering multiple prompts, you can separate them with commas. Using commas improves script readability, but it is not required: `ACCEPT "Specify a start date:" TO v_start_date, "Specify an end date:" TO v_end_date`
FIELDS *project_item_category* optional	Creates a drop-down list of project items for user input instead of a text box. The user can select a single project item, field, or variable from the list. *project_item_category* specifies which item types to display in the list. For example, specifying xf displays all the project tables in the list. Enclose *project_item_category* in quotation marks: `FIELDS "xf"` For the codes used to specify categories, see "Codes for project item categories" on page 57. You can specify more than one code in the same prompt, but you cannot mix project items, fields, or variables.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
TO *variable_name*	The name of the character variable to use to store the user input. If the variable does not exist, it is created. If the variable already exists, its current value is displayed in the dialog box as the default value. **Note** You cannot use non-English characters, such as é, in the names of variables that will be used in variable substitution. Variable names that contain non-English characters will cause the script to fail. The ACCEPT command creates character variables only. If you need input of another data type, you must convert the character variable to the required type in subsequent processing in a script. For more information, see "Input data type" on page 57.

Examples

Prompting the user to select the Analytics table to open

You require a dialog box that prompts the user to select the name of the table to open. The script then opens the table the user selects:

```
ACCEPT "Select the table to open:" FIELDS "xf" TO v_table_name
OPEN %v_table_name%
```

The percent signs are required because they indicate that the table name to open is stored in the *v_table_name* variable. If the percent signs are omitted, the script attempts to open a table called "v_table_name".

Using multiple dialog boxes to gather required input

You want to create a separate dialog box for each value that the script user must enter.

You use a single prompt string in each instance of the ACCEPT command. The script generates separate dialog boxes for specifying each of the following:

- a table name
- a field on which to sample
- a sampling interval
- a random start value

```
ACCEPT "Enter the name of the table to analyze" TO v_table_name
OPEN %v_table_name%
ACCEPT "Select the field to sample" FIELDS "N" to v_field_to_sample
ACCEPT "Enter the sampling interval" TO v_sampling_interval
ACCEPT "Enter the random start value" TO v_random_start_value
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
SAMPLE ON %v_field_to_sample% INTERVAL v_sampling_interval FIXED v_random_start_value
RECORD TO Sample_output OPEN
```

When the script runs

1. The first dialog box prompts for the table name.
2. The second dialog box, with `FIELDS "N"` , prompts for a field selection from a drop-down list of numeric fields.
3. The third dialog box prompts for the interval value.
4. The fourth dialog box prompts for the random start value.

Using a single dialog box with multiple prompts to gather required input

You want to create a single dialog box for all values that the script user must enter.

You use multiple prompts separated by commas in the ACCEPT command to ask the user for multiple input values. The same dialog box contains prompts for the start date and the end date of a date range:

```
ACCEPT "Specify a start date:" TO v_start_date, "Specify an end date:" TO v_end_date
```

Remarks

Interactivity

Use ACCEPT to create an interactive script. When the ACCEPT command is processed, the script pauses and a dialog box is displayed that prompts the user for input that Analytics uses in subsequent processing.

You can create separate dialog boxes that prompt for one item at a time, or you can create one dialog box that prompts for multiple items.

DIALOG versus ACCEPT

The DIALOG command allows you to create a more advanced interactive dialog box that can have one or more of the following types of controls:

- text box
- check box
- radio buttons
- drop-down list of customized values
- project item list

You also have the flexibility to customize the layout of the dialog box. For more information, see "DIALOG command" on page 151.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Codes for project item categories

Use the following codes to specify the category of project item to display in a drop-down list.

Project categories

Code	Category
xf	Tables
xb	Scripts
xi	Indexes
xr	Views and reports
xw	Workspaces

Field categories

Code	Category
C	Character fields
N	Numeric fields
D	Datetime fields
L	Logical fields

Variable categories

Code	Category
c	Character variables
n	Numeric variables
d	Datetime variables
l	Logical variables

Input data type

ACCEPT stores the user input in one or more character variables. If you need numeric or datetime input, you can use the VALUE( ) or CTOD( ) functions to convert the contents of a character variable to a numeric

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

or datetime value:

```
SET FILTER TO BETWEEN(%v_date_field%, CTOD(%v_start_date%), CTOD(%v_end_date%))
```

In the example, the start and end dates for this filter are stored as character values. They must be converted to date values in order to be used with a date field that uses a Datetime data type.

Enclosing the variable name in percent signs (%) substitutes the character value contained by the variable for the variable name. The CTOD( ) function then converts the character value to a date value.

Position of the ACCEPT command

It is good practice to place all ACCEPT commands at the beginning of a script, if possible. If you ask for all input at the beginning, the script can then run unimpeded once the user enters the necessary information.

> **Note**
>
> You cannot use the ACCEPT command inside the GROUP command.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ACCESSDATA command

Imports data from a variety of ODBC-compliant data sources.

The command takes the form ACCESSDATA64 or ACCESSDATA32 depending on whether you are using a 64-bit or 32-bit ODBC driver.

Syntax

```
{ACCESSDATA64 | ACCESSDATA32} {CONNECTOR | ODBC {"Driver"|"Dsn"|"File"}} NAME value
<USER user_id> <PASSWORD num | PROMPT_PASSWORD> TO table_name CHARMAX max_
field_length MEMOMAX max_field_length <ALLCHARACTER> SOURCE (connection_settings)
<HASH(salt_value, fields)>
SQL_QUERY
(SQL_syntax)
END_QUERY
```

Parameters

Name	Description
CONNECTOR | ODBC {"Driver"|"Dsn"|"File"}	The type of ODBC connection you want to make: ◦ **CONNECTOR** - connect using a native Analytics data connector ◦ **ODBC "Driver"** - connect using a Windows ODBC driver installed on your computer ◦ **ODBC "Dsn"** - connect using a saved DSN (data source name) on your computer ◦ **ODBC "File"** - connect using a file DSN (a saved .dsn file)
NAME *value*	The name of the Analytics data connector, the ODBC driver, or the DSN. For example: ◦ `NAME "Amazon Redshift"` ◦ `NAME "Microsoft Access Driver (*.mdb, *.accdb)"` ◦ `NAME "My Excel DSN"` ◦ `NAME "excel.dsn"`
USER *user_id* optional	The user ID for data sources that require a user ID.
PASSWORD *num* | PROMPT_PASSWORD optional	For data sources that require a password: ◦ **PASSWORD num** - use the numbered password definition ◦ **PROMPT_PASSWORD** - displays a password prompt The password prompt also allows the *user_id* to be changed. If you use PASSWORD *num*, you must specify a previously created password definition.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	For more information, see "PASSWORD command" on page 360 and "SET command" on page 418. **Tip** Using the PASSWORD command with PASSWORD *num* is similar to using PROMPT_PASSWORD. Both approaches prompt the user for a password. PROMPT_PASSWORD has the benefit of allowing updating of the *user_id*.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
CHARMAX *max_field_length*	The maximum length in characters for any field in the Analytics table that originates as character data in the source from which you are importing. The default value is 50. Data that exceeds the maximum field length is truncated when imported to Analytics.
MEMOMAX *max_field_length*	The maximum length in characters for text, note, or memo fields you are importing. The default value is 100. Data that exceeds the maximum field length is truncated when imported to Analytics.
ALLCHARACTER optional	Automatically assign the Character data type to all imported fields. Once the data is in Analytics, you can assign different data types, such as Numeric or Datetime, to the fields, and specify format details. **Tip** ALLCHARACTER is useful if you are importing a table that contains numeric ID values. You can use ALLCHARACTER to prevent Analytics automatically assigning the Numeric data type to values that should use the Character data type.
SOURCE *connection_settings*	The connection settings (connection string) required to connect to the data source.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
HASH(*salt_value*, *fields*) optional	Imports the specified fields as cryptographic hash values. Hash values are one-way transformations and cannot be decoded after you import the fields: ◦ ***salt_value*** - an alphanumeric string that is concatenated with the source data values to strengthen the hashing of the values in the fields. Enter the hash value as a quoted string. The salt value is limited to 128 characters. Do not use any of the following characters: ( ) " ◦ ***fields*** - a list of one or more fields to hash. Enter the fields as a quoted string and separate each field with a comma. You must specify the field name you see in the Data Access window preview and staging area, not the physical field name in the data source. **Note** The field name that is shown in the Data Access window preview is the field alias value in the SQL query ("field_name" AS "alias"). You must use the alias value to reference fields. `HASH("QZ3x7", "SSN_NO, CC_NO, Last_Name")` For information about comparing values hashed during import to values hashed in ACLScript, see "Comparing data hashed with ACCESSDATA to data hashed with the ACLScript HASH( ) function" on page 65.
SQL_QUERY (*SQL_syntax*) END_QUERY	The SQL import statement. Everything inside the parentheses is part of the SQL query and must be valid SQL.

Examples

Importing data using a native Analytics data connector

You need to import data from the Amazon Redshift cloud data service. To do so, you use the Analytics Amazon Redshift data connector:

```
ACCESSDATA64 CONNECTOR NAME "Amazon Redshift" USER "ACL_user" PROMPT_
PASSWORD TO "Entitlement_History.FIL" CHARMAX 50 MEMOMAX 100
 SOURCE( boolsaschar-
=0;cache-
size-
=100;dat-
abase-
=usage-
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
;de-
clarefetch-
mode=0;maxbytea=255;maxlongvarchar=8190;maxvarchar=255;port=5439;servername=acl_
test.high-
bond.-
com;sin-
glerowmode=1;sslmode=require;textaslongvarchar=0;usemultiplestatments=0;useunicode=1)
 SQL_QUERY(
  SELECT
   "entitlement_history"."organization" AS "organization",
   "entitlement_history"."user_email" AS "user_email",
   "entitlement_history"."plan_id" AS "plan_id",
   "entitlement_history"."date_from" AS "date_from",
   "entitlement_history"."date_to" AS "date_to"
  FROM
   "prm"."entitlement_history" "entitlement_history"
) END_QUERY
```

Importing data using a Windows ODBC driver

You need to import data from a Microsoft Access database. To do so, you use a Windows ODBC driver to connect to MS Access and complete the import:

```
ACCESSDATA32 ODBC "Driver" NAME "Microsoft Access Driver (*.mdb)" TO "Invoices.FIL"
CHARMAX 50 MEMOMAX 100
 SOURCE( dbq=C:\Users\lachlan_murray\Documents\ACL Data\Sample Data Files\Sample.m-
db;defaultdir=C:\Users\lachlan_murray\Documents\ACL Data\Sample Data Files;driv-
erid=281;fil=MS
Access;maxbuf-
fersize=2048;maxscanrows=8;pagetimeout=5;safetransactions=0;threads=3;usercommitsync=Yes)
 SQL_QUERY(
  SELECT
   `Customer`.`CustID` AS `CustID`,
   `Customer`.`Company` AS `Company`,
   `Customer`.`Address` AS `Address`,
   `Customer`.`City` AS `City`,
   `Customer`.`Region` AS `Region`,
   `Customer`.`PostalCode` AS `PostalCode`,
   `Customer`.`Country` AS `Country`,
   `Customer`.`Phone` AS `Phone`,
   `Orders`.`OrderID` AS `OrderID`,
   `Orders`.`CustID` AS `Orders_CustID`,
   `Orders`.`ProdID` AS `ProdID`,
   `Orders`.`OrderDate` AS `OrderDate`,
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
    `Orders`.`Quantity` AS `Quantity`,
    `Product`.`ProdID` AS `Product_ProdID`,
    `Product`.`ProdName` AS `ProdName`,
    `Product`.`UnitPrice` AS `UnitPrice`,
    `Product`.`Descript` AS `Descript`,
    `Product`.`ShipWt` AS `ShipWt`
   FROM
    (`Customer` `Customer`
   INNER JOIN
    `Orders` `Orders`
      ON `Customer`.`CustID` = `Orders`.`CustID`
    )
INNER JOIN
   `Product` `Product`
    ON `Orders`.`ProdID` = `Product`.`ProdID`
WHERE
   (
    `Customer`.`Region` = 'BC'
    OR `Customer`.`Region` = 'WA'
   )
 ) END_QUERY
```

Importing data using a Windows DSN (data source name)

You need to import data from a Microsoft Excel file. To do so, you use a Windows DSN to connect to Excel and complete the import:

```
ACCESSDATA32 ODBC "Dsn" NAME "Excel Files" TO "Trans_April_15_cutoff.FIL" CHARMAX 50
MEMOMAX 100
 SOURCE( dbq=C:\Users\lachlan_murray\Documents\ACL Data\Sample Data Files\Trans_
April.xls;defaultdir=C:\Users\lachlan_murray\Documents\ACL Data\Sample Data Files;driv-
erid=1046;maxbuffersize=2048;pagetimeout=5)
 SQL_QUERY(
  SELECT
   `Trans_Apr_`.`CARDNUM` AS `CARDNUM`,
   `Trans_Apr_`.`AMOUNT` AS `AMOUNT`,
   `Trans_Apr_`.`TRANS_DATE` AS `TRANS_DATE`,
   `Trans_Apr_`.`CODES` AS `CODES`,
   `Trans_Apr_`.`CUSTNO` AS `CUSTNO`,
   `Trans_Apr_`.`DESCRIPTION` AS `DESCRIPTION`
  FROM
   `Trans_Apr$` `Trans_Apr_`
  WHERE
   (
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
    `Trans_Apr_`.`TRANS_DATE` <= {ts '2003-04-15 00:00:00'}
  )
) END_QUERY
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Data connector updates

When you upgrade Analytics, the Robots Agent, or AX Server, you should test any of your scripts that import data using one of the Analytics data connectors (ACCESSDATA command).

The possibility exists that changes made by third-party data sources or ODBC driver vendors required updates to one or more of the data connectors. Scripted data connections may need to be updated in order to continue working correctly.

- **Re-run the import** – The easiest way to update a connection is to manually perform an import using the Data Access window in the upgraded version of Analytics. Copy the ACCESSDATA command from the log and use it to update your script.

  > **Note**
  >
  > Before connecting to a data source and re-running the import, clear the connector cache to flush the existing set of table names.
  >
  > In the **Existing Connections** tab in the Data Access window, beside the connector name, select ··· **> Clear cache**.

- **Update field specifications** – You may also need to update field specifications in the script body to align with table schema changes in the data source or ODBC driver. Possible changes include field names, field data types, and field and record lengths.
- **Check the results of any filtering** – You should also check the results of any filtering that you apply as part of the data import. Confirm that the import filtering is including and excluding records correctly.

Creating ODBC connection settings and SQL import statements

ODBC connection settings, and SQL import statements, are often quite lengthy and involved, as shown in the examples.

The easiest way to create these portions of the ACCESSDATA command is to first use the Data Access window in Analytics to connect to the target data source, and import data. You can then copy the entire

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ACCESSDATA command from the log, including the connection settings and import statement, and customize the command in any way you require.

Password value suppressed

When you use the Data Access window in Analytics to run the ACCESSDATA command, and provide a password, the password value is not written to the log. Instead, the PROMPT_PASSWORD parameter is substituted.

ACCESSDATA log files

Two log files record the transactions associated with the ACCESSDATA command, and can be used for troubleshooting if a data connection fails:

- **ServerDataAccess.log** - records all activities and errors prior to importing the data

  Location: `C:\Users\<user account>\AppData\Local\ACL\ACL for Windows\Data Access\ServerDataAccess.log`

  > **Note**
  >
  > The "Server" in `ServerDataAccess.log` refers to the data access component of Analytics running locally on the computer where Analytics is installed.

- **DataAccess.log** - records information about the import operation and the Analytics project that you are importing data to

  Location: `..\<Analytics project folder>\DataAccess.log`

Comparing data hashed with ACCESSDATA to data hashed with the ACLScript HASH( ) function

Even though you cannot read the raw values of hashed data, it is still useful when combining or analyzing data.

If you want to compare values that are hashed by ACCESSDATA during import with values that are hashed using ACLScript's HASH( ) function, you must convert any numeric or datetime Analytics fields to character values and trim all leading and trailing spaces before hashing the data.

Datetime fields must use the following formats when converted to character:

- **Datetime** - "YYYY-MM-DD hh:mm:ss"
- **Date** - "YYYY-MM-DD"
- **Time** - "hh:mm:ss"

The following example uses the STRING( ) and ALLTRIM( ) functions to convert a numeric credit card number field to character data before hashing the value using ACLScript's HASH( ) function:

```
COMMENT ACL HASH function used after importing data
HASH(ALLTRIM(STRING(CC_No, 16)), "QZ3x7")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Once you hash the Analytics values, you can compare them with values hashed as part of the ACCESSDATA command import.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ACTIVATE command

Adds field definitions stored in an Analytics workspace to the existing set of field definitions in an Analytics table layout.

Syntax

```
ACTIVATE <WORKSPACE> workspace_name <OK>
```

Parameters

Name	Description
WORKSPACE *workspace_name*	The name of the workspace to activate.
OK optional	Deletes or overwrites items without asking you to confirm the action. If there is a field in the table with an identical name to one in the activated workspace, it will be overwritten without confirmation. You cannot replace any field that is referenced by a computed field.

Examples

Activating a workspace in your Analytics project

You activate the **ComplexFormulas** workspace:

```
ACTIVATE WORKSPACE ComplexFormulas OK
```

Activating a workspace saved as a file (.wsp) in the same folder as your Analytics project

You activate the **ComplexFormulas** workspace that was saved to a .wsp file:

```
ACTIVATE WORKSPACE ComplexFormulas.WSP OK
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

How it works

ACTIVATE makes workspace field definitions available to the active table. Once you activate a workspace, its fields remain available for use with the active table until you close the table.

Editing table layouts

The workspace fields are permanently added to the table layout if:

- you edit the table layout after you activate a workspace
- you make a change that causes the table layout to be saved

Once the workspace fields are saved in the table layout, you can:

1. Use the DEFINE COLUMN command to add the fields to a view.
2. Use the SAVE command to save your changes.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

AGE command

Groups records into aging periods based on values in a date or datetime field. Counts the number of records in each period, and also subtotals specified numeric fields for each period.

Syntax

AGE <ON> *date_field* <CUTOFF *cutoff_date*> <INTERVAL *days <,...n>*> <SUPPRESS> <SUBTOTAL *numeric_field <...n>*|SUBTOTAL ALL> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <TO {SCREEN|*filename*|GRAPH|PRINT}> <KEY *break_field*> <HEADER *header_text*> <FOOTER *footer_text*> <APPEND> <STATISTICS>

Parameters

Name	Description
ON *date_field*	The name of the date or datetime field or the expression to be aged. Although you can age on a datetime field, only the date portion of datetime values is considered. The time portion is ignored. You cannot age on time data alone.
CUTOFF *cutoff_date* optional	The date that values in *date_field* are compared against. You must specify *cutoff_date* as an unquoted string in YYMMDD or YYYYMMDD format, regardless of the format of the date field. For example: `CUTOFF 20141231` If you omit CUTOFF, the current system date is used as the cutoff date.
INTERVAL *days <,...n>* optional	The date intervals (that is, number of days) to use in calculating the aging periods. *days* represents the beginning of each aging period measured backward in time from *cutoff_date*: ◦ the first *days* value identifies the beginning of the first aging period ◦ a first *days* value of '0' specifies that the first aging period begins on the specified *cutoff_date* ◦ the last *days* value identifies the end of the final aging period You must specify the intervals as an unquoted string with comma separated values: `INTERVAL 0,90,180,270,365` The default aging periods are 0, 30, 60, 90, 120, and 10,000 days. An interval of 10,000 days is used to isolate records with dates that are probably invalid. If required, date intervals can be customized to mirror other internal aging reports.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
SUPPRESS optional	Suppresses dates that fall outside the aging period from the command output.
SUBTOTAL *numeric_field <...n>* | SUBTOTAL ALL optional	One or more numeric fields or expressions to subtotal for each group. Multiple fields must be separated by spaces. Specify ALL to subtotal all the numeric fields in the table.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
TO SCREEN | *filename* | GRAPH | PRINT	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"` ◦ **GRAPH** - displays the results in a graph in the Analytics display area ◦ **PRINT** - sends the results to the default printer
KEY *break_field*	The field or expression that groups subtotal calculations. A subtotal is calculated each

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
optional	time the value of *break_field* changes. *break_field* must be a character field or expression. You can specify only one field, but you can use an expression that contains more than one field.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
STATISTICS optional	**Note** Cannot be used unless SUBTOTAL is also specified. Calculates average, minimum, and maximum values for all SUBTOTAL fields.

Examples

Age invoices with subtotaled amounts

You want to age an accounts receivable table on the **Invoice_Date** field and subtotal the **Invoice_Amount** field.

Invoices are grouped into 30-day periods:

- from the cutoff date to 29 days previous
- from 30 days previous to 59 days previous
- so on

The results include the total outstanding invoice amount for each period:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
OPEN Ar
AGE ON Invoice_Date CUTOFF 20141231 INTERVAL 0,30,60,90,120,10000 SUBTOTAL Invoice_
Amount TO SCREEN
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Aging periods

The AGE command groups records into aging periods based on values in a date or datetime field. The output results contain a single record for each period, with a count of the number of records in the source table that fall into each period.

Interval measurement

Aging periods are based on date intervals (that is, number of days) measured backward in time from the current system date, or from a cutoff date you specify such as a fiscal period end date.

Future periods

You can create aging periods more recent than the cutoff date by entering negative values for date intervals. For example, the following creates aging periods running forward and backward from the cutoff date:

```
INTERVAL -60,-30,0,30,60,90
```

This approach creates a date profile of all the records in a table using different points in time.

Common use cases

Common uses of aging include evaluating sales trends, looking at transaction volumes, and grouping invoices by the number of days outstanding.

Analytics automatically creates one or two additional aging periods for any dates that fall outside the specified aging periods, assuming you are not using SUPPRESS.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

APPEND command

Combines records from two or more Analytics tables by appending them in a new Analytics table.

Syntax

APPEND *table_1*, *table_2*, *<...n>* TO *table_name* <COMMONFIELDS> <OPEN> <ASCHAR> <ALLCHAR> <SOURCETABLE>

Parameters

Name	Description
table_1, *table_2*, *<...n>*	The tables to append. The records from each table are appended in the order in which you specify the tables. The output table contains the records from *table_1*, followed by the records from *table_2*, and so on. The source tables can have different or identical record structures, and can be sorted or unsorted.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
COMMONFIELDS optional	Only those fields that are common to all tables being appended are included in the output table. If you omit COMMONFIELDS, all fields from all tables are included in the output table. Blank values appear in the output table where no fields exist in the source tables.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Tip For diagrams and screen captures illustrating the two options, see Appending tables. **Note** The APPEND command does not support appending computed fields. For more information, see "Computed fields not supported" on page 78. **What makes fields common?** For fields to be considered common they must: ◦ occur in every source table ◦ have an identical physical name ◦ belong to the same data category: • Character • Numeric • Datetime • Logical **Identical name, different data category** If two fields have an identical name but belong to different data categories, an error message appears and the APPEND command is not executed. The error message contains all data category conflicts in the set of tables specified by APPEND. The message is saved to the command log. **Note** You can avoid this situation by using either ASCHAR or ALLCHAR to harmonize data categories.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.
ASCHAR optional	Harmonizes fields with identical names but different data categories by converting non-character fields to the character data category. For example, you append two tables in which the Employee_ID field is character data in one table, and numeric data in the other. The numeric Employee_ID field is converted to character data and the two fields are appended without an error. ASCHAR is ignored if ALLCHAR is also specified.
ALLCHAR optional	Converts all non-character fields in all tables being appended to the character data category. This global conversion to character data ensures that all identically named fields are appended without error. **Note** After appending, you can change the data category of an entire appended field if appropriate for the data contained by the field.
SOURCETABLE	Include the **Source Table** field (`Source_Table`) in the output table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
optional	For each record in the output table, the **Source Table** field identifies the table from which the record originated. **Tip** Including the names of the source tables you are appending may provide useful information when you analyze data in the output table.

Examples

Append three monthly transaction tables

The example below appends three monthly transaction tables and outputs a quarterly transaction table that includes all fields from the three source tables:

```
APPEND Trans_Jan, Trans_Feb, Trans_Mar TO Trans_Q1
```

Append three employee tables and include only common fields

The example below appends three divisional employee tables and outputs a master employee table that includes only common fields from the three source tables:

```
APPEND Employees_central, Employees_east, Employees_west TO Employees_master
COMMONFIELDS
```

Append three employee tables and harmonize fields with different data categories

The examples below append three divisional employee tables in which some identically named fields use different data categories.

The first example converts non-character fields to the character data category only when required for harmonization:

```
APPEND Employees_central, Employees_east, Employees_west TO Employees_master ASCHAR
```

The second example converts all non-character fields to the character data category whether required for harmonization or not:

```
APPEND Employees_central, Employees_east, Employees_west TO Employees_master ALLCHAR
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

How it works

The APPEND command combines records from two or more tables by appending them and creating a new table. Appending means to add one group of records to the bottom of another group of records.

Source table fields with identical physical names and identical data categories are directly appended to one another.

Fields with physical names that are unique across all the source tables are added to the output table but not directly appended to any other field.

> **Tip**
>
> If you want to directly append inconsistently named fields, standardize the physical names of the fields in the table layouts before appending. (Assumes that the fields belong to the same data category, or that you use ASCHAR or ALLCHAR to harmonize the data category of the fields.)

When to use APPEND

Use APPEND when you want to combine data from multiple tables with an identical or similar structure. For example, APPEND is a good choice for combining monthly or quarterly tables into an annual table.

> **Tip**
>
> A single execution of the APPEND command can replace multiple executions of the EXTRACT command with the APPEND option.

Not a substitute for JOIN or DEFINE RELATION

APPEND is generally not a substitute for the JOIN or DEFINE RELATION commands because it does not allow you to include or exclude records based on matched or unmatched values in a common key field. With APPEND, all records from each source table are included in the output table.

Appending completely dissimilar tables

You can append completely dissimilar tables – that is, two or more tables that do not have any fields in common. While not the primary intended use of the APPEND command, there may be instances in which appending dissimilar tables serves an analytical purpose.

Appending datetime fields

For two or more datetime fields to be appended, the following conditions must be met:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- identical physical names
- identical data category (Datetime)
- identical data subtypes (date, datetime, or time)
- identical use of time zone indicator – either used, or not used, by all fields being appended

If two datetime fields have an identical name but fail to meet one of the other conditions an error message appears and the APPEND command is not executed.

The error message contains all failed conditions in the set of tables specified by APPEND. The message is saved to the command log.

> **Note**
>
> You can harmonize dissimilar datetime fields by converting them to the character data category, and then append them. This approach allows you to combine the data in a single table. However, depending on the nature of the source data, you may not be able to convert the combined data back to datetime data.

Automatic harmonization

In some situations the APPEND command automatically harmonizes fields in order to append them:

Data category of fields	Harmonization performed
Character	◦ Different field lengths are harmonized. ◦ Different character data types such as Custom, PCASCII, and EBCDIC are harmonized by converting the fields to the ASCII or UNICODE data type.
Numeric	◦ Different field lengths are harmonized. The fields are converted to the ACL data type. ◦ A different number of defined decimal places are harmonized. Decimal places are standardized on the greatest number of places, with trailing zeros added to numeric values where necessary. The fields are converted to the ACL data type. ◦ Different numeric data types such as Print, Float, EBCDIC, and Micro are harmonized by converting the fields to the ACL data type.
Datetime	◦ Different date, datetime, or time formats in the source data are harmonized by converting the fields to the Analytics default formats: • YYYYMMDD • YYYYMMDD hh:mm:ss • hh:mm:ss

When automatic harmonization is not performed

Analytics does not automatically harmonize fields in the following situations. An error message appears and the append operation is not executed.

- Two fields with an identical name belong to different data categories.
- Two datetime fields with an identical name belong to different datetime subtypes (date, datetime, or time).
- Two datetime fields with an identical name are inconsistent in their use of the time zone indicator.

> **Note**

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> User-specified harmonization of fields with identical names but different data categories is explained above. For more information, see "ASCHAR" on page 74 and "ALLCHAR" on page 74.

Computed fields not supported

The APPEND command does not support appending computed fields. When you append tables, any computed fields in the source tables are automatically excluded from the output table.

If a computed field in a source table has the same name as a physical field in another source table, an error message appears and the APPEND command is not executed.

> **Tip**
>
> You can append a computed field by first extracting it to convert the field to a physical field. (For more information, see "EXTRACT command" on page 203.) You then use the extracted table in the append operation.
>
> Another approach is to recreate the computed field in the appended output table.

Record Note fields not supported

The APPEND command does not support appending Record Note fields. When you append tables, any Record Note fields in the source tables are automatically excluded from the output table.

If a Record Note field in a source table has the same name as a physical field in another source table, an error message appears and the APPEND command is not executed.

A Record Note field is automatically generated by Analytics when you add a note to a record.

Record length

If you include all fields from all source tables when appending, the record length in the output table can be longer than the longest record in the source tables.

An error message appears if the output record length exceeds the Analytics maximum of 32 KB.

Appending and decimal places

Specific behavior governs the appending of numeric fields that include decimal places.

The Decimal setting

The APPEND command uses the number of decimal places defined in the **Dec** setting in the field definition in the table layout.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> The **Dec** setting may not be the same as the actual number of decimal places in the source data. Decimal places that exceed the **Dec** setting are undefined, and are rounded in calculations.

Inconsistent Decimal settings

If appended numeric fields have inconsistent **Dec** settings, the fields are converted to the ACL data type and automatically harmonized on the longest **Dec** setting.

Any decimal places in source data files that exceed the longest **Dec** setting are **excluded** from the output table generated by APPEND.

Consistent Decimal setting

If appended numeric fields have a consistent **Dec** setting, no data type conversion or harmonization occurs.

Any decimal places in source data files that exceed the **Dec** setting are **included** in the output table generated by APPEND.

Sorting

Any existing sort orders in the source tables are separately maintained in the respective record sets in the output table.

Even if the records in all source tables are sorted, the output table is considered unsorted because the source records are appended as groups, without consideration of any existing sort order in other source tables.

For example, if you append monthly or quarterly tables to create an annual table, any internal sorting of the monthly or quarterly data is retained. If required, you can sort the output table after performing the append operation.

How field order works

Common fields

Common fields in source tables do not have to be in the same order to be appended.

For example, these fields are correctly appended even though they are in a different order:

Table	Fields
table_1	Last_name | First_name | Middle_name
table_2	First_name | Middle_name | Last_name

The first table specified in the APPEND command dictates the order of the fields in the output table. So in the example above, the order in the output table is:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- Last_name | First_name | Middle_name

Non-common fields

Non-common fields in source tables appear in the output table in the order that they appear in the selected group of source tables.

For example, when appending these two tables:

Table	Fields
table_1	Title | Last_name | First_name | Middle_name
table_2	First_name | Middle_name | Last_name | Date_of_birth

the order in the output table is:

- Title | Last_name | First_name | Middle_name | Date_of_birth

Alternate Column Title

Alternate column titles in source tables appear in the output table. If more than one source table has an alternate column title for the same field, the title from the first selected table takes precedence.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ASSIGN command

Creates a variable and assigns a value to the variable.

Syntax

```
ASSIGN variable_name = value <IF test>
```

> **Tip**
>
> You can omit the ASSIGN keyword because Analytics automatically interprets the following syntax as an assignment operation:
>
> ```
> variable_name = value
> ```

Parameters

Name	Description
variable_name	The name of the variable to assign the value to. If the variable does not exist, it is created. If the variable already exists, it is updated with the new value. Do not use non-English characters, such as é, in the names of variables. Variable names that contain non-English characters will cause scripts to fail. **Note** Variable names are limited to 31 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
value	The value to assign to the variable. If a new variable is created, the variable type is based on the data type in *value*.
IF *test* Optional	A conditional expression that must be true to create the variable or assign the value to the variable.

Examples

Assigning a value to a variable

You assign the value of the **Amount** field in the current record to a variable named *v_current_amount*.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Because *v_current_amount* is a variable, its value does not change unless explicitly changed by another ASSIGN command:

```
ASSIGN v_current_amount = Amount
```

Conditionally assigning a value to a variable

You want to update the value of a variable called *v_quantity* to 1, but only if the value in another variable called *v_counter* is less than 10.

If *v_counter* is greater than or equal to 10, no assignment is made and the value of *v_quantity* remains unchanged.

Note that the optional ASSIGN keyword is omitted:

```
v_quantity = 1 IF v_counter < 10
```

Remarks

Duration of variables

Variables with names that are not prefaced with an underscore are retained for the duration of the current Analytics session only.

If you want a variable to be permanently saved with an Analytics project, preface the variable name with an underscore:

```
ASSIGN value = _variable_name
```

Reassigning variables used in a computed field or GROUP

If you assign a value to an existing variable in the following situations, then the new value is assigned but the previous value's length and decimal count are retained:

- variables used in computed fields
- variables reassigned inside a GROUP

The length of the new value is padded or truncated and the decimals are adjusted if required.

If you reassign a variable in any other context, then the previous value as well as its length and decimal specifications are overwritten.

Variables created by Analytics commands

When you execute certain commands, either by entering information in dialog boxes in Analytics or by running scripts, system variables are automatically created by Analytics. You can use these variables, and the

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

values they contain, when processing subsequent Analytics commands.

The value in a system variable is replaced with an updated value if you execute the same command again.

For more information, see "Variables created by Analytics commands" on page 947.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

BENFORD command

Counts the number of times each leading digit (1-9) or leading digit combination occurs in a field, and compares the actual count to the expected count. The expected count is calculated using the Benford formula.

Syntax

BENFORD <ON> *numeric_field* <LEADING *n*> <IF *test*> <BOUNDS> <TO {SCREEN|*table_name*|GRAPH|PRINT}> <LOCAL> <HEADER *header_text*> <FOOTER *footer_text*> <WHILE *test*> <FIRST *range*|NEXT *range*> <APPEND> <OPEN>

Parameters

Name	Description
ON *numeric_field*	The numeric field to analyze. **Note** Select a field that contains "naturally occurring numbers", such as transaction amounts. Benford analysis is not suitable for numeric data that is constrained in any way. For more information, see "What data can I test using Benford analysis?" on page 86
LEADING *n* optional	The number of leading digits to analyze. The value of *n* must be from 1 to 6. If LEADING is omitted, the default value of 1 is used.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
BOUNDS optional	Includes computed upper and lower bound values in the output results. If the actual count of more than one digit or digit combination in the output results exceeds either of the bounds, the data may have been manipulated and should be investigated.
TO SCREEN | *table_name* | GRAPH | PRINT optional	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Tip You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. ◦ **GRAPH** - displays the results in a graph in the Analytics display area ◦ **PRINT** - sends the results to the default printer
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.

Examples

Outputting results to graph

You run the BENFORD command against the **Amount** field and output the results to a graph:

```
BENFORD ON Amount LEADING 2 BOUNDS TO GRAPH
```

Remarks

> **Note**
> For more information about how this command works, see the Analytics Help.

What data can I test using Benford analysis?

You should only use Benford analysis for testing numeric data composed of "naturally occurring numbers", such as accounting amounts, transaction amounts, expenses, or address numbers. Benford analysis is not suitable for numeric data that is constrained in any way.

Follow these guidelines for identifying numeric data that is suitable for Benford analysis:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- **Size of the data set** – The data set must be large enough to support a valid distribution. Benford analysis may not give reliable results for fewer than 500 records.
- **Leading digit requirement** – All numbers from 1 to 9 must have the possibility of occurring as the leading digit.
- **Leading digit combination requirement** – All numbers from 0 to 9 must have the possibility of occurring as the second leading digit, and as any additional digits being analyzed.
- **Constrained data** – Numeric data that is assigned or generated according to a pre-ordained pattern is not suitable for Benford analysis. For example, do not use Benford to analyze:
  - sequential check or invoice numbers
  - social security numbers or telephone numbers that map to a specific pattern
  - any numbering scheme with a range that prevents certain numbers from appearing
- **Random numbers** – Numbers generated by a random number generator are not suitable for Benford analysis.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CALCULATE command

Calculates the value of one or more expressions.

Syntax

```
CALCULATE expression <AS result_label> <,...n>
```

Parameters

Name	Description
expression	The expression to calculate. The expression can be any of the four types: ◦ character ◦ numeric ◦ datetime ◦ logical Separate multiple expressions with a comma: `CALCULATE 4.7 * 18.5, 1 + 2, "a" + "b"`
AS *result_label* optional	The name of the result when displayed on screen and in the Analytics command log. *result_label* must be a quoted string or a valid character expression. If omitted, the expression being calculated is used as the result name.

Examples

Performing a simple calculation

You use CALCULATE to multiply 4.70 by 18.50, returning the result 86.95:

```
CALCULATE 4.70 * 18.50
```

Naming the results of a calculation

You use CALCULATE to derive the gross margin for the currently selected record using previously defined fields for sale price and unit cost:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
CALCULATE Sale_price - Unit_cost AS "Margin"
```

The result is identified on screen, and in the log, as "Margin".

Remarks

How it works

CALCULATE provides the functionality of a calculator combined with access to Analytics functions, variables, and the data in the currently selected record.

Command output

Depending on where you run CALCULATE, the results are output to different locations:

- **From the command line** – the result is displayed on screen
- **From within a script** – the result is recorded in the log

The *result_label* value is not a variable that you can use in a script. It is only used to identify the calculation on screen or in the log.

Number of decimal places in output

In a numeric calculation, the result has as many decimal places as the expression component with the greatest number of decimal places.

Returns 1:

```
CALCULATE 365/52/7
```

Returns 1.0027:

```
CALCULATE 365.0000/52/7
```

Working with table input

If the expression contains a field value, the table the field belongs to must be open. You can use the FIND, SEEK, or LOCATE commands to move to the record to be analyzed by CALCULATE.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CLASSIFY command

Groups records based on identical values in a character or numeric field. Counts the number of records in each group, and also subtotals specified numeric fields for each group.

Syntax

CLASSIFY <ON> *key_field* <SUBTOTAL *numeric_field <...n>*|SUBTOTAL ALL> <INTERVALS *number*> <SUPPRESS> <TO {SCREEN|*table_name*|GRAPH|PRINT}> <LOCAL> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <HEADER *header_text*> <FOOTER *footer_text*> <KEY *break_field*> <OPEN> <APPEND> <STATISTICS>

Parameters

Name	Description
ON *key_field*	The character or numeric field to classify. Maximum key field length is 2048 characters. If you want to classify a table using a key field longer than 2048 characters, use the SUMMARIZE command. It does not restrict key field length.
SUBTOTAL *numeric_field <...n>* | SUBTOTAL ALL optional	One or more numeric fields or expressions to subtotal for each group. Multiple fields must be separated by spaces. Specify ALL to subtotal all the numeric fields in the table.
INTERVALS *number* optional	The maximum number of groups in the output result. If the number of sets of identical values in the field being classified exceeds the specified maximum, sets are used starting from the top of the column. Sets exceeding the maximum are grouped together in a group called "OTHER". If INTERVALS is omitted, a group is created for each set of identical values in the field being classified. **Note** This parameter is not available in the Analytics user interface and can only be used as part of ACLScript syntax in a script or the command line.
SUPPRESS optional	**Note** Cannot be used unless INTERVALS is also specified. SUPPRESS is not available in the Analytics user interface and can only be used as part of ACLScript syntax in a script or the command line. Excludes sets of identical values exceeding the maximum specified by INTERVALS

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	from the command output.
TO SCREEN | *table_name* | GRAPH | PRINT	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. ◦ **GRAPH** - displays the results in a graph in the Analytics display area ◦ **PRINT** - sends the results to the default printer
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.
KEY *break_field* optional	The field or expression that groups subtotal calculations. A subtotal is calculated each time the value of *break_field* changes. *break_field* must be a character field or expression. You can specify only one field, but you can use an expression that contains more than one field.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
STATISTICS optional	**Note** Cannot be used unless SUBTOTAL is also specified. Calculates average, minimum, and maximum values for all SUBTOTAL fields.

Examples

Total transaction amount per customer

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You want to classify an accounts receivable table on the **Customer_Number** field and subtotal the **Trans_Amount** field. The output results are grouped by customer, and include the total transaction amount for each customer:

```
OPEN Ar
CLASSIFY ON Customer_Number SUBTOTAL Trans_Amount TO "Customer_total.FIL"
```

Total, average, minimum, and maximum transaction amounts per customer

As with the previous example, you classify an accounts receivable table on the **Customer_Number** field and subtotal the **Trans_Amount** field.

Now you include STATISTICS to calculate the average, minimum, and maximum transaction amounts for each customer:

```
OPEN Ar
CLASSIFY ON Customer_Number SUBTOTAL Trans_Amount TO "Customer_stats.FIL"
STATISTICS
```

Identical invoice amounts

You need to identify invoice amounts that appear more than once in the **Ap_Trans** table.

To do this, you classify the table on the **Invoice_Amount** field. The output results are grouped by invoice amount with an associated count that you can use to identify any invoice amounts that occur more than once:

```
OPEN Ap_Trans
CLASSIFY ON Invoice_Amount TO "Grouped_invoice_amounts.FIL" OPEN
SET FILTER TO COUNT > 1
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

How it works

CLASSIFY groups records that have the same value in a character or numeric field.

Output contains a single record for each group, with a count of the number of records in the source table that belong to the group.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Sorting and CLASSIFY

CLASSIFY can process either sorted or unsorted data. Output is automatically sorted in ascending order.

Names of auto-generated subtotal and statistics fields

If you use STATISTICS to perform statistical calculations on one or more SUBTOTAL fields, and output the results to an Analytics table, the fields auto-generated by the parameters have the following names:

Description of auto-generated field	Field name in output table	Alternate column title (display name) in output table
Subtotal field	*subtotaled field name in source table*	**Total** + *subtotaled alternate column title in source table*
Average field	**a_***subtotaled field name in source table*	**Average** + *subtotaled alternate column title in source table*
Minimum field	**m_***subtotaled field name in source table*	**Minimum** + *subtotaled alternate column title in source table*
Maximum field	**x_***subtotaled field name in source table*	**Maximum** + *subtotaled alternate column title in source table*

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CLOSE command

Closes an Analytics table, index file, or log file, or ends a **Script Recorder** session.

Syntax

```
CLOSE <table_name|PRIMARY|SECONDARY|INDEX|LOG|LEARN>
```

Parameters

Name	Description
table_name |PRIMARY|SECONDARY|INDEX|LOG|LEARN optional	The item to close: ◦ ***table_name*** - the name of the Analytics table to close ◦ **PRIMARY** - closes the primary Analytics table Using CLOSE without any parameters also closes the primary table. ◦ **SECONDARY** - closes the secondary Analytics table ◦ **INDEX** - closes the current index applied to the Analytics table ◦ **LOG** - returns the log file to the default command log, after the SET LOG command has been used to specify another log file ◦ **LEARN** - ends the active **Script Recorder** session and prompts you to save the script file the session was recorded to LEARN can be used in scripts, but its intended use is in the command line. The **Script Recorder** records the ACLScript syntax for commands that are executed using dialog boxes in the Analytics user interface.

Examples

Closing a table by name

You want to close a table called **Inventory**:

```
CLOSE Inventory
```

Closing a table by type

You want to close the current secondary table:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
CLOSE SECONDARY
```

Restoring the default Analytics command log

You want to restore the default command log after using a separate log file to capture the data verification phase of a script:

```
SET LOG TO "DataVerificationPhase.log"
COMMENT Execute data verification commands
CLOSE LOG
```

Remarks

When to use CLOSE

You typically do not need to close Analytics tables. The active Analytics table automatically closes when you open another table. The primary table also closes automatically before the OPEN or QUIT commands execute.

You cannot use CLOSE to close an Analytics project. Use QUIT instead.

Related fields and tables

When you close a primary or secondary table, all related field definitions are removed from memory. Any changes to the table layout are saved before the table is closed.

If you have defined table relations in an Analytics project, the CLOSE command closes both the primary and any secondary tables. It also closes the related tables.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CLUSTER command

Groups records into clusters based on similar values in one or more numeric fields. Clusters can be uni-dimensional or multidimensional.

> **Note**
>
> The CLUSTER command is not supported if you are running Analytics on a 32-bit computer. The computation required by the command is processor-intensive and better suited to 64-bit computers.

Syntax

CLUSTER ON *key_field* <...*n*> KVALUE *number_of_clusters* ITERATIONS *number_of_iterations* INITIALIZATIONS *number_of_initializations* <SEED *seed_value*> <OTHER *field* < ...*n*>|OTHER ALL> TO *table_name* <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> OPEN {*no_keyword*|NOCENTER|NOSCALE}

Parameters

Name	Description
ON *key_field* <...*n*>	One or more numeric fields to cluster. Multiple fields must be separated by spaces.
KVALUE *number_of_clusters*	The number of clusters generated in the output results.
ITERATIONS *number_of_iterations*	The maximum number of times the cluster calculation is re-performed.
INITIALIZATIONS *number_of_initializations*	The number of times to generate an initial set of random centroids.
SEED *seed_value* optional	The seed value to use to initialize the random number generator in Analytics. If you omit SEED, Analytics randomly selects the seed value.
OTHER *field* <...*n*> | OTHER ALL optional	One or more additional fields to include in the output. ◦ **OTHER *field* <...*n*>** - include the specified field or fields Fields are included in the order that you list them. ◦ **OTHER ALL** - include all fields in the table Fields are included in the order that they appear in the table layout.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note Key fields are automatically included in the output table, although the values are scaled unless you specify NOSCALE. You can use OTHER to include a second, unscaled instance of a key field or fields.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
no_keyword | NOCENTER | NOSCALE	The method for standardizing key field numeric values. ◦ ***no_keyword*** - center key field values around zero (0), and scale the values to unit variance when calculating the clusters ◦ **NOCENTER** - scale key field values to unit variance when calculating the clusters, but do not center the values around zero (0) ◦ **NOSCALE** - use the raw key field values, unscaled, when calculating the clusters

Examples

Clustering on invoice amount

In addition to stratifying an accounts receivable table on the **Invoice_Amount** field, you also decide to cluster on the same field.

- Stratifying groups the amounts into strata with predefined numeric boundaries – for example, $1000 intervals.
- Clustering discovers any organic groupings of amounts that exist in the data without requiring that you decide on numeric boundaries in advance.

```
Open Ar
CLUSTER ON Invoice_Amount KVALUE 8 ITERATIONS 30 INITIALIZATIONS 10 OTHER No Due
Date Ref Type TO "Clustered_invoices" NOSCALE
```

As a quick way of discovering how many records are contained in each output cluster, you classify the Clustered_invoices output table on the **Cluster** field.

```
OPEN Clustered_invoices
CLASSIFY ON Cluster TO SCREEN
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

COMMENT command

Adds an explanatory note to a script without affecting processing.

Syntax

Single-line comments

```
COMMENT comment_text
```

Multiline comments

```
COMMENT
 comment_text
 <...n>
<END>
```

> **Note**
>
> Do not use a caret character `^` to preface lines of comment text. The caret has a special use in the .acl project file, and comment text is not saved if you preface it with a caret.

Parameters

Name	Description
comment_text	The comment you are adding. ◦ **single-line comment** - enter the entire comment text without a line break ◦ **multiline comment** - enter as many lines of comment text as necessary starting on the line immediately following the COMMENT command Terminate a multiline comment with the END keyword on a separate line, or with a blank line.
END optional	The end of a multiline COMMENT command. If you use END, it must be entered on the line immediately following the last comment line. If you omit END, a blank line must follow the last comment line.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Single-line comments

You use single-line comments before commands to add documentation for future users who will maintain the script:

```
COMMENT Generate the standard deviation and average.
STATISTICS ON %v_amt% STD TO SCREEN NUMBER 5
COMMENT Create fields for storing standard deviation and average.
DEFINE FIELD Standard_Dev COMPUTED STDDEV1
DEFINE FIELD Average COMPUTED AVERAGE1
```

Multiline comment

You begin each script you write with a multiline comment that explains the purpose of the script:

```
COMMENT
  This analytic identifies multiple records having common
  transaction originator IDs (like vendor ID or merchant ID)
  where the transaction date values are either equal or one day apart.
  This analytic can be used for split invoices, split purchase orders,
  split requisitions, and split corporate card transactions.
END
```

Remarks

When to use COMMENT

Use COMMENT to include information about the purpose of a script, the logic used, and other information such as the required inputs for the script and the purpose of each variable you define.

The comments in a script are written to the Analytics command log each time the script runs.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

COUNT command

Counts the total number of records in the current view, or only those records that meet the specified condition.

Syntax

```
COUNT <IF test> <WHILE test> <FIRST range|NEXT range>
```

Parameters

Name	Description
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics output variables

Name	Contains
COUNT*n*	The record count calculated by the command. ◦ If the variable name is COUNT1, it is storing the record count for the most recent command executed. ◦ If the variable name is COUNT*n*, where *n* is greater than 1, the variable is storing the record count for a command executed within a GROUP command. The value of *n* is assigned based on the line number of the command in the GROUP. For example, if the command is one line below the GROUP command it is assigned the value COUNT2. If the command is four lines below the GROUP command, it is assigned the value COUNT5.

Examples

Storing COUNT1

The result of the COUNT command is stored in the COUNT1 output variable. You can retrieve and store this value in a user-defined variable.

The COUNT command overwrites the COUNT1 variable each time it is executed, so the value needs to be stored in a user-defined variable before the command is executed for the second time after the filter is applied to the table:

```
OPEN CustomerAddress
COUNT
TotalRec = COUNT1
SET FILTER TO ModifiedDate > '20100101'
COUNT
TotalFilteredRec = COUNT1
```

Remarks

When to use COUNT

Use the COUNT command to count the number of records in an Analytics table, or to count the number of records that meet a particular test condition. If no test is specified, the total number of records in the Analytics table is displayed.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

How filters affect COUNT

If a filter has been applied to a view, the command counts the number of records remaining in the view after the filtering condition has been applied.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CREATE LAYOUT command

Creates an empty Analytics table layout, which may be required in certain scripting situations.

Syntax

```
CREATE LAYOUT layout_name WIDTH characters <RECORD 0|RECORD 1>
```

Parameters

Name	Description
layout_name	The name of the layout.
WIDTH *characters*	The record length in characters.
RECORD 0 | RECORD 1 optional	◦ If you specify `RECORD 0`, or omit this parameter, the table layout is created without any records or a source data file. ◦ If you specify `RECORD 1` the table layout is created with a single empty record and a source data file named *layout_name*.fil.

Examples

Creating an empty table layout without any records

You create an empty table layout with a record length of 100 characters:

```
CREATE LAYOUT empty_table WIDTH 100
```

Creating an empty table layout with one record

You create:

- an empty table layout with one empty record
- a record length of 50 characters
- an associated Analytics data file called `empty_table.fil`

```
CREATE LAYOUT empty_table WIDTH 50 RECORD 1
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

The empty table layout is created with a single character field called **Field_1**. The field length is the same as the record length you specify with WIDTH.

> **Note**
>
> This command is not supported for use in Analytics analytics run on AX Server.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CROSSTAB command

Groups records based on identical combinations of values in two or more character or numeric fields, and displays the resulting groups in a grid of rows and columns. Counts the number of records in each group, and also subtotals specified numeric fields for each group.

Syntax

CROSSTAB <ON> *row_field* <*...n*> COLUMNS *column_field* <SUBTOTAL *numeric_field* <*...n*>|SUBTOTAL ALL> TO {SCREEN|*table_name*|*filename*|GRAPH|PRINT} <LOCAL> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <APPEND> <COUNT> <OPEN> <HEADER *header_text*> <FOOTER *footer_text*>

Parameters

Name	Description
ON *row_field* <*...n*>	The field or expression to use for rows in the resulting grid of rows and columns. You can specify one or more fields or expressions as the basis for the rows.
COLUMNS *column_field*	The field or expression to use for columns in the resulting grid of rows and columns. You can specify only one field or expression for the columns.
SUBTOTAL *numeric_field* <*...n*> | SUBTOTAL ALL optional	One or more numeric fields or expressions to subtotal for each group. Multiple fields must be separated by spaces. Specify ALL to subtotal all the numeric fields in the table.
TO SCREEN | *table_name* | *filename* | GRAPH | PRINT	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"` ◦ **GRAPH** - displays the results in a graph in the Analytics display area ◦ **PRINT** - sends the results to the default printer
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
COUNT optional	Includes record counts as columns. Counts are useful when you use SUBTOTAL. Counts are automatically included if you do not select any subtotal fields.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.

Examples

Cross-tabulating an accounts receivable table with SUBTOTAL

You want to cross-tabulate an accounts receivable table on the **Customer Number** and **Transaction Type** fields. You also want to subtotal the **Transaction Amount** field.

The output is grouped by customer, and within each customer by transaction type. It includes the total transaction amount for each customer for each transaction type:

```
OPEN Ar
CROSSTAB ON Customer_Number COLUMNS Trans_Type SUBTOTAL Trans_Amount COUNT
TO SCREEN
```

Cross-tabulating an accounts receivable table to find duplicate transactions

You need to find evidence of duplicate transactions in an accounts receivable table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

To do this, you cross-tabulate an accounts receivable table on the **Transaction Amount** and **Transaction Type** fields. The output groups and counts identical transaction amounts for each transaction type:

```
OPEN Ar
CROSSTAB ON Trans_Amount COLUMNS Trans_Type TO SCREEN
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

How it works

CROSSTAB groups records that have the same combination of values in two or more character or numeric fields.

The output contains a grid of rows and columns similar to a pivot table. It includes a single row-column intersection for each group, with a count of the number of records in the source table that belong to the group.

Sorting and CROSSTAB

CROSSTAB can process either sorted or unsorted data. Both the *row_field* and the *column_field* in the output are automatically sorted in ascending order.

If you specify more than one *row_field*, the fields use a nested sort, starting with the first *row_field* you specify.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CVSEVALUATE command

For classical variables sampling, provides four different methods for projecting the results of sample analysis to the entire population.

Syntax

CVSEVALUATE BOOKED *book_value_field* AUDITED *audit_value_field* ETYPE {MPU|DIFFERENCE|RATIO SEPARATE|RATIO COMBINED|ALL} STRATA *boundary_value* <,...*n*> POPULATION *stratum_count*,*stratum_book_value* <,...*n*> CONFIDENCE *confidence_level* CUTOFF *value*,*certainty_stratum_count*,*certainty_stratum_book_value* ERRORLIMIT *number* PLIMIT {BOTH|UPPER|LOWER} <BCUTOFF *value*,*certainty_stratum_count*,*certainty_stratum_book_value*> <TO {SCREEN|*filename*}>

Parameters

> **Note**
>
> If you are using the output results of the CVSPREPARE and CVSSAMPLE commands as input for the CVSEVALUATE command, a number of the parameter values are already specified and stored in variables. For more information, see "CVSPREPARE command" on page 115 and "CVSSAMPLE command" on page 120.
>
> Do not include thousands separators, or percentage signs, when you specify values.

Name	Description
BOOKED *book_value_field*	The numeric book value field to use in the evaluation.
AUDITED *audit_value_field*	The numeric audit value field to use in the evaluation.
ETYPE MPU | DIFFERENCE | RATIO SEPARATE | RATIO COMBINED | ALL	The estimation type to use: • MPU (Mean-per-unit) • Difference • Ratio Separate • Ratio Combined • All For more information, see "Which estimation type should I use?" on page 113
STRATA *boundary_value* <,...*n*>	The upper boundary values to use for stratifying the *book_value_field*.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
POPULATION *stratum_count*, *stratum_value* <,...*n*>	The number of records and the total value for each stratum in the *book_value_field*.
CONFIDENCE *confidence_level*	The confidence level used during the preparation stage of the classical variables sample.
CUTOFF *value*, *certainty_stratum_count*, *certainty_stratum_book_value*	◦ ***value*** - the top certainty stratum cutoff value used during the preparation and sampling stage of the classical variables sample ◦ ***certainty_stratum_count*** - the number of records in the top certainty stratum ◦ ***certainty_stratum_book_value*** - the total book value of the records in the top certainty stratum
ERRORLIMIT *number*	The minimum number of errors you expect in the sample. **Note** If the actual number of errors you found when you analyzed the sample is less than the ERRORLIMIT *number*, the only evaluation method available is mean-per-unit.
PLIMIT BOTH | UPPER | LOWER	The type of precision limit to use: ◦ Both ◦ Upper ◦ Lower For more information, see "CVSPREPARE command" on page 115.
BCUTOFF *value*, *certainty_stratum_count*, *certainty_stratum_book_value* optional	◦ ***value*** - the bottom certainty stratum cutoff value used during the preparation and sampling stage of the classical variables sample ◦ ***certainty_stratum_count*** - the number of records in the bottom certainty stratum ◦ ***certainty_stratum_book_value*** - the total book value of the records in the bottom certainty stratum
TO SCREEN | *filename*	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Project errors found in the sampled data to the entire population

You have completed your testing of the sampled data and recorded the misstatements you found. You can now project the errors you found to the entire population.

The example below uses the Difference estimation type to project the results of sample analysis to the entire population:

```
CVSEVALUATE BOOKED invoice_amount AUDITED AUDIT_VALUE ETYPE DIFFERENCE
STRATA 4376.88,9248.74,16904.52,23864.32 POPULATION
1279,3382131.93,898,5693215.11,763,9987014.57,627,12657163.59,479,13346354.63
CONFIDENCE 95.00 CUTOFF 35000.00,36,1334318.88 ERRORLIMIT 6 PLIMIT BOTH TO
SCREEN
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Which estimation type should I use?

The estimation type that you should use depends on the nature of the data: the sample book values, the sample audit values, and the relation between them.

Guidelines

The guidelines below help you select an estimation type.

> **Tip**
>
> If you want to compare the results produced by different estimation types, you can specify ETYPE ALL to include all estimation types in the evaluation output.

Estimation type	Presence of mis-statements	Size of mis-statements	Sign of book values	Comparison of strata ratios
Mean-per-unit	**No misstatements, or very few mis-statements** The only valid estim-ation type if there are no misstatements, or very few mis-statements, in the	n/a	n/a	n/a

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Estimation type	Presence of misstatements	Size of misstatements	Sign of book values	Comparison of strata ratios
	audited sample population.			
Difference	**Misstatements required** Requires a number of misstatements in the audited sample population. For example, 5% or more of the samples contain misstatements.	**Misstatements are non-proportional** More suitable when misstatements are non-proportional: the size of a misstatement is not related to the size of the associated book value. In other words, small and large book values can have either small or large misstatements.	n/a	n/a
Ratio Separate		**Misstatements are proportional** More suitable when misstatements are proportional: the size of a misstatement is related to the size of the associated book value. In other words, small book values have small misstatements, and large book values have large misstatements.	**Book values have the same sign** All sample book values must have the same sign: either all positive, or all negative.	**Ratios vary** More suitable when the ratio of average sample audit value to average sample book value varies widely between strata.
Ratio Combined				**Ratios are consistent** More suitable when the ratio of average sample audit value to average sample book value is relatively consistent between strata.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CVSPREPARE command

Stratifies a population, and calculates a statistically valid sample size for each stratum, for classical variables sampling.

Syntax

CVSPREPARE ON *book_value_field* NUMSTRATA *number* MINIMUM *minimum_strata_sample_size* PRECISION *value* CONFIDENCE *confidence_level* <CUTOFF *value*> <BCUTOFF *value*> NCELLS *number* PLIMIT {BOTH|UPPER|LOWER} ERRORLIMIT *number* <IF *test*> <MINSAMPSIZE *minimum_sample_size*> TO {SCREEN|*filename*}

Parameters

> **Note**
>
> Do not include thousands separators, or percentage signs, when you specify values.

Name	Description
ON *book_value_field*	The numeric book value field to use as the basis for preparing the classical variables sample.
NUMSTRATA *number*	The number of strata to use for numerically stratifying the *book_value_field*. The minimum number of strata is 1, and the maximum is 256. If you specify NUMSTRATA 1, and do not specify a CUTOFF, the population is unstratified prior to drawing a sample. > **Note** > The number of strata cannot exceed 50% of the number of cells specified for NCELLS.
MINIMUM *minimum_strata_sample_size*	The minimum number of records to sample from each stratum. Leave the default of zero (0) if you do not have a specific reason for specifying a minimum number.
PRECISION *value*	The monetary amount that is the difference between the tolerable misstatement and the expected misstatement in the account. ◦ **Tolerable misstatement** - the maximum total amount of misstatement that can occur in the sample field without being considered a material misstatement ◦ **Expected misstatement** - the total amount of misstatement that you expect the sample field to contain The precision establishes the range of acceptability for an account to be considered fairly stated.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Reducing the precision decreases the range of acceptability (the margin of error) which requires an increased sample size.
CONFIDENCE *confidence_level*	The desired confidence level that the resulting sample is representative of the entire population. For example, specifying 95 means that you want to be confident that 95% of the time the sample will in fact be representative. Confidence is the complement of "sampling risk". A 95% confidence level is the same as a 5% sampling risk. ◦ If PLIMIT is BOTH, the minimum confidence level is 10%, and the maximum is 99.5%. ◦ If PLIMIT is UPPER or LOWER, the minimum confidence level is 55%, and the maximum is 99.5%.
CUTOFF *value* optional	A top certainty stratum cutoff value. Amounts in the *book_value_field* greater than or equal to the cutoff value are automatically selected and included in the sample. If you omit CUTOFF, a default cutoff value equal to the maximum amount in the *book_value_field* is used, and no records are included in the top certainty stratum.
BCUTOFF *value* optional	A bottom certainty stratum cutoff value. Amounts in the *book_value_field* less than or equal to the cutoff value are automatically selected and included in the sample. If you omit BCUTOFF, a default cutoff value equal to the minimum amount in the *book_value_field* is used, and no records are included in the bottom certainty stratum.
NCELLS *number*	The number of cells to use for pre-stratifying the *book_value_field*. Cells are narrower numeric divisions than strata. Pre-stratification is part of an internal process that optimizes the position of strata boundaries. Cells are not retained in the final stratified output. The minimum number of cells is 2, and the maximum is 999. **Note** The number of cells must be at least twice (2 x) the number of strata specified for NUMSTRATA.
PLIMIT BOTH | UPPER | LOWER	The type of precision limit to use. ◦ BOTH - specify this option if: • the account as a whole could be either overstated or understated • you are interested in estimating whether misstatement in either direction exceeds the specified PRECISION ◦ UPPER - specify this option if: • the account as a whole is likely to be understated • you are only interested in estimating whether the total amount of understatement exceeds the specified PRECISION ◦ LOWER - specify this option if: • the account as a whole is likely to be overstated • you are only interested in estimating whether the total amount of overstatement exceeds the specified PRECISION

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Caution Specify BOTH if you are not sure which option to specify.
ERRORLIMIT *number*	The minimum number of errors you expect in the sample. **Note** If the actual number of errors you find when you analyze the sample is less than the ERRORLIMIT *number*, the only evaluation method available is mean-per-unit.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Caution** If you specify a conditional expression, an identical conditional expression must be used during both the calculation of the sample size, and the drawing of the sample. If you use a condition at one stage and not the other, or if the two conditions are not identical, the sampling results will probably not be statistically valid.
MINSAMPSIZE *minimum_sample_size* optional	The minimum number of records to sample from the entire population. Leave the default of zero (0) if you do not have a specific reason for specifying a minimum number.
TO SCREEN | *filename*	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: TO "Output.TXT" By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • TO "C:\Output.TXT" • TO "Results\Output.TXT"

Analytics output variables

Name	Contains
CONFIDENCE	The confidence level specified by the user.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Contains
ERRLIMIT	The minimum number of errors specified by the user.
NSTRATA	The number of strata specified by the user.
PLIMIT	The type of precision limit specified by the user.
S_IF	A conditional expression specified by the user
S_TOP	The top certainty stratum cutoff value specified by the user, or if none was specified, the upper boundary of the top stratum calculated by the command.
SAMPLEFIELD	The book value field specified by the user.
SBOTTOM	The bottom certainty stratum cutoff value specified by the user, or if none was specified, the lower boundary of the bottom stratum calculated by the command.
SBOUNDARY	All strata upper boundaries calculated by the command. Does not include top or bottom certainty strata.
SPOPULATION	The count of the number of records in each stratum, and the total monetary value for each stratum. Does not include top or bottom certainty strata.
SSAMPLE	The sample size for each stratum calculated by the command. Does not include top or bottom certainty strata.

Examples

Prepare a classical variables sample

You have decided to use classical variables sampling to estimate the total amount of monetary misstatement in an account containing invoices.

Before drawing the sample, you must first stratify the population, and calculate a statistically valid sample size for each stratum.

You want to be confident that 95% of the time the sample drawn by Analytics will be representative of the population as a whole.

Using your specified confidence level, the example below stratifies a table based on the **invoice_amount** field, and calculates the sample size for each stratum and the top certainty stratum:

```
CVSPREPARE ON invoice_amount NUMSTRATA 5 MINIMUM 0 PRECISION 928003.97
CONFIDENCE 95.00 CUTOFF 35000 NCELLS 50 PLIMIT BOTH ERRORLIMIT 6 MINSAMPSIZE
0 TO SCREEN
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Numeric length limitation

Several internal calculations occur during the preparation stage of classical variables sampling. These calculations support numbers with a maximum length of 17 digits. If the result of any calculation exceeds 17 digits, the result is not included in the output, and you cannot continue with the sampling process.

Note that source data numbers of less than 17 digits can produce internal calculation results that exceed 17 digits.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CVSSAMPLE command

Draws a sample of records using the classical variables sampling method.

Syntax

CVSSAMPLE ON *book_value_field* NUMSTRATA *number* <SEED *seed_value*> CUTOFF *value* <BCUTOFF *value*> STRATA *boundary_value* <,...*n*> SAMPLESIZE *number* <,...*n*> POPULATION *stratum_count*,*stratum_value* <,...*n*> <IF *test*> TO *table_name*

Parameters

> **Note**
>
> If you are using the output results of the CVSPREPARE command as input for the CVSSAMPLE command, a number of the parameter values are already specified and stored in variables. For more information, see "CVSPREPARE command" on page 115.
>
> Do not include thousands separators, or percentage signs, when you specify values.

Name	Description
ON *book_value_field*	The numeric book value field to use as the basis for the sample.
NUMSTRATA *number*	The number of strata to use for stratifying the *book_value_field*.
SEED *seed_value* optional	The seed value to use to initialize the random number generator in Analytics. If you omit SEED, Analytics randomly selects the seed value.
CUTOFF *value*	A top certainty stratum cutoff value. Amounts in the *book_value_field* greater than or equal to the cutoff *value* are automatically selected and included in the sample.
BCUTOFF *value* optional	A bottom certainty stratum cutoff value. Amounts in the *book_value_field* less than or equal to the cutoff *value* are automatically selected and included in the sample.
STRATA *boundary_value* <,...*n*>	The upper boundary values to use for stratifying the *book_value_field*.
SAMPLESIZE *number* <,...*n*>	The number of records to sample from each stratum.
POPULATION *stratum_*	The number of records in each stratum, and the total value for each stratum.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
count, *stratum_value* <,...*n*>	
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Caution** If you specify a conditional expression, an identical conditional expression must be used during both the calculation of the sample size, and the drawing of the sample. If you use a condition at one stage and not the other, or if the two conditions are not identical, the sampling results will probably not be statistically valid.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.

Analytics output variables

Name	Contains
S_TOPEV	The top certainty stratum cutoff value specified by the user, or if none was specified, the upper boundary of the top stratum previously calculated by the CVSPREPARE command. Also stores the count of the number of records in the top certainty stratum, and their total monetary value.
SBOTTOMEV	The bottom certainty stratum cutoff value specified by the user, or if none was specified, the lower boundary of the bottom stratum previously calculated by the CVSPREPARE command. Also stores the count of the number of records in the bottom certainty stratum, and their total monetary value.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Contains
SBOUNDARYEV	All strata upper boundaries prefilled by the command, or specified by the user. Does not include top or bottom certainty strata.
SPOPULATION	The count of the number of records in each stratum, and the total monetary value for each stratum. Does not include top or bottom certainty strata.

Examples

Draw a classical variables sample

You are going to use classical variables sampling to estimate the total amount of monetary misstatement in an account containing invoices.

After stratifying the population, and calculating a statistically valid sample size for each stratum, you are ready to draw the sample.

The example below draws a stratified sample of records based on the **invoice_amount** field, and outputs the sampled records to the `Invoices_sample` table:

```
CVSSAMPLE ON invoice_amount NUMSTRATA 5 SEED 12345 CUTOFF 35000.00 STRATA
4376.88,9248.74,16904.52,23864.32,35000.00 SAMPLESIZE 37,36,49,36,39 POPULATION
1279,3382131.93,898,5693215.11,763,9987014.57,627,12657163.59,479,13346354.63 TO
"Invoices_sample"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

System-generated fields

Analytics automatically generates four fields and adds them to the sample output table. For each record included in the sample, the fields contain the following descriptive information:

- **STRATUM** - the number of the stratum to which the record is allocated
- **ORIGIN_RECORD_NUMBER** - the original record number in the source data table
- **SELECTION_ORDER** - on a per-stratum basis, the order in which the record was randomly selected
- **SAMPLE_RECORD_NUMBER** - the record number in the sample output table

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DEFINE COLUMN command

Creates and adds one or more columns to an existing view.

Syntax

DEFINE COLUMN *view_name field_name* <AS *display_name*> <POSITION *n*> <WIDTH *characters*> <PIC *format*> <SORT|SORT D> <KEY> <PAGE> <NODUPS> <NOZEROS> <LINE *n*>

Parameters

Name	Description
view_name	The view to add the column to.
field_name	The field to create the column for. To use a field from a related table, specify the field name as *table_name.field_name*.
AS *display_name* optional	The display name (alternate column title) for the field in the view. If you want the display name to be the same as the field name do not use AS. Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title.
POSITION *n* optional	The position of the column in the view numerically from left to right: ◦ if omitted, the column is placed as the rightmost column at the time that the column is added ◦ if a position number is missing, column positions are adjusted so that the columns are positioned sequentially ◦ if a position number is already in use, the new column is placed to the left of the column already using the position number
WIDTH *characters* optional	The display width of the field in characters. The specified value controls the display width of the field in Analytics views and reports. The display width never alters data, however it can hide data if it is shorter than the field length. If you omit WIDTH, the display width is set to the character width specified for the field in the table layout.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note The characters specified by WIDTH are fixed-width characters. Every character is allotted the same amount of space, regardless of the width of the actual character. By default, views in Analytics use a proportional width font that does not correspond with fixed-width character spacing. If you want a one-to-one correspondence between the WIDTH value and the characters in the view, you can change the **Proportional Font** setting in the **Options** dialog box to a fixed-width font such as Courier New.
PIC *format* optional	**Note** Applies to numeric or datetime fields only. ◦ **numeric fields** – the display format of numeric values in Analytics views and reports ◦ **datetime fields** – the physical format of datetime values in the source data (order of date and time characters, separators, and so on) **Note** For datetime fields, *format* must exactly match the physical format in the source data. For example, if the source data is 12/31/2014, you must enter the format as "MM/DD/YYYY". *format* must be enclosed in quotation marks.
SORT | SORT D optional	Sorts the column: ◦ **ascending order** – SORT ◦ **descending order** – SORT D
KEY optional	The column is designated as a break field in reports. Reports are subtotaled and subdivided when the value of the column changes. The following restrictions apply to break fields: ◦ must be a character field or expression ◦ if a break field is set in the view, it must be the leftmost column ◦ the last column in the view cannot be a break field ◦ if you have more than one break field, all of the columns to the left of any additional break field must also be break fields
PAGE optional	Inserts a page break each time the value in the break field changes.
NODUPS optional	Substitutes blank values for repeated values in the field. For example, if the customer name is listed for each invoice record, the report is easier to read if it shows only the first instance of each customer name.
NOZEROS optional	Substitutes blank values for zero values in the field. For example, if a report includes a large number of zero values in a field, the report is easier to read if it only displays non-zero values.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
LINE *n* optional	The number of lines in the column. If no value is specified, the column defaults to a single line. The value of *n* must be between 2 and 60.

Examples

Defining a view with six columns

With the **AR** table open, you define a view called **AR_Report**, and define six columns. The columns are displayed in the listed order:

```
OPEN Ar
DEFINE VIEW AR_Report OK
DEFINE COLUMN AR_Report No AS "Customer Number" WIDTH 7 KEY
DEFINE COLUMN AR_Report Date AS "Invoice Date" WIDTH 10
DEFINE COLUMN AR_Report Due AS "Due Date" WIDTH 10
DEFINE COLUMN AR_Report Reference AS "Reference Number" WIDTH 6
DEFINE COLUMN AR_Report Type AS "Transaction Type" WIDTH 5
DEFINE COLUMN AR_Report Amount AS "Transaction Amount" WIDTH 12 PIC "-9999999999.99"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DEFINE FIELD command

Defines a physical data field in an Analytics table layout.

Syntax

DEFINE FIELD *field_name data_type start_position length* <*decimals*|*date_format*> <NDATETIME> <PIC *format*> <AS *display_name*> <WIDTH *characters*> <SUPPRESS> <*field_note*>

Parameters

Name	Description
field_name	The name of the field. **Note** Field names are limited to 256 upper and lowercase alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. Analytics has a number of reserved keywords that cannot be used as field names. For a complete list, see "Reserved keywords" on page 951.
data_type	The data type to use when interpreting the data. For a list of supported data types, see "Supported data types" on page 131. For example, invoice numbers may be stored as numeric values in the data source. To treat these values as strings rather than numbers, you can define the field as character data instead.
start_position	The starting byte position of the field in the Analytics data file. **Note** non-Unicode Analytics: 1 byte = 1 character Unicode Analytics, extended ASCII (ANSI) data: 1 byte = 1 character Unicode Analytics, Unicode data: 2 bytes = 1 character For Unicode data, typically you should specify an odd-numbered starting byte position. Specifying an even-numbered starting position can cause characters to display incorrectly.
length	The length of the field in bytes.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>

Note

<table>
<tr><td>non-Unicode Analytics</td><td>1 byte = 1 character</td></tr>
<tr><td>Unicode Analytics, extended ASCII (ANSI) data</td><td>1 byte = 1 character</td></tr>
<tr><td>Unicode Analytics, Unicode data</td><td>2 bytes = 1 character</td></tr>
</table>

For Unicode data, specify an even number of bytes only. Specifying an odd number of bytes can cause characters to display incorrectly.

</td></tr>
<tr><td>decimals
optional</td><td>The number of decimals for numeric fields.</td></tr>
<tr><td>date_format
optional</td><td>The date format in the source date fields.
For datetime or time fields, use PIC format instead. You can also use PIC format for date fields.
If the source data includes separators such as slashes, you must include the separators in date_format. For example, if the source data is 12/31/2014, you must enter the format as MM/DD/YYYY. Do not enclose date_format in quotation marks.</td></tr>
<tr><td>NDATETIME
optional</td><td>Date, datetime, or time values stored in a numeric field are treated as datetime data.
NDATETIME requires that you also specify the source datetime format using PIC format.</td></tr>
<tr><td>PIC format
optional</td><td>

Note

Applies to numeric or datetime fields only.

- **numeric fields** - the display format of numeric values in Analytics views and reports
- **datetime fields** - the physical format of datetime values in the source data (order of date and time characters, separators, and so on)

Note

For datetime fields, format must exactly match the physical format in the source data. For example, if the source data is 12/31/2014, you must enter the format as "MM/DD/YYYY".

format must be enclosed in quotation marks.

</td></tr>
<tr><td>AS display_name
optional</td><td>The display name (alternate column title) for the field in the view. If you want the display name to be the same as the field name do not use AS.
Specify display_name as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title.</td></tr>
<tr><td>WIDTH characters
optional</td><td>The display width of the field in characters.
The specified value controls the display width of the field in Analytics views and reports. The display width never alters data, however it can hide data if it is shorter than the field length.
The display width cannot be less than the length of field_name, or display_name.</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	If you omit WIDTH, the display width is set to the field length in characters. **Note** The characters specified by WIDTH are fixed-width characters. Every character is allotted the same amount of space, regardless of the width of the actual character. By default, views in Analytics use a proportional width font that does not correspond with fixed-width character spacing. If you want a one-to-one correspondence between the WIDTH value and the characters in the view, you can change the **Proportional Font** setting in the **Options** dialog box to a fixed-width font such as Courier New.
SUPPRESS optional	Only applies to numeric fields. Suppresses automatic totaling of a numeric field in Analytics reports. Totaling of some numeric fields is not appropriate. For example, a unit cost field, or a discount rate field.
field_note optional	Field note text that is added to the field definition in the table layout. *field_note* must be last, after all other required and optional parameters. The text cannot be multiline. Quotation marks are not required.

Examples

Defining a character field

Defines a character field called **ProdDesc**. The column title in the view is **Product Description**.

non-Unicode Analytics

- Starts at: byte 12 (character position 12)
- Length: 24 bytes (24 characters)

```
DEFINE FIELD ProdDesc ASCII 12 24 AS "Product Description"
```

Unicode Analytics, extended ASCII (ANSI) data

- Starts at: byte 12
- Length: 24 bytes (24 characters)

```
DEFINE FIELD ProdDesc ASCII 12 24 AS "Product Description"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Unicode Analytics, Unicode data

- Starts at: byte 13
- Length: 48 bytes (24 characters)

```
DEFINE FIELD ProdDesc UNICODE 13 48 AS "Product Description"
```

Defining a numeric field

Defines a numeric field called **QtyOH**. In the view, the column uses the specified display format, and the title is **Quantity On Hand**.

- Starts at: byte 61
- Length: 10 bytes
- Decimal places: none

```
DEFINE FIELD QtyOH NUMERIC 61 10 0 PIC "(9,999,999)" AS "Quantity On Hand"
```

Defining a datetime field from character data

From source character data, the first two examples below define a datetime field called **Transaction_date**. In the source data, the date format is DD/MM/YYYY. No column title is specified, so the column title defaults to using the field name.

- Starts at: byte 20
- Length: 10 bytes

Here, the date format is specified using *date_format*:

```
DEFINE FIELD Transaction_date DATETIME 20 10 DD/MM/YYYY
```

Here, the date format is specified using PIC *format*:

```
DEFINE FIELD Transaction_date DATETIME 20 10 PIC "DD/MM/YYYY"
```

When defining datetime fields that include time data, you must use PIC *format*,

The example below defines a datetime field called **email_timestamp**. In the source data, the datetime format is YYYY/MM/DD hh:mm:ss-hh:mm.

- Starts at: byte 1
- Length: 25 bytes

```
DEFINE FIELD email_timestamp DATETIME 1 25 PIC "YYYY/MM/DD hh:mm:ss-hh:mm"
```

Defining a datetime field from numeric data

From source numeric data, defines a datetime field called **Receipt_timestamp** that has the specified

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

datetime format in the source data.

- Starts at: byte 15
- Length: 15 bytes

```
DEFINE FIELD Receipt_timestamp DATETIME 15 15 PIC "YYYYMMDD.hhmmss"
```

Defining a "numeric" datetime field

From source numeric data, defines a numeric field called **Receipt_timestamp** that has the specified datetime format in the source data.

The NDATETIME parameter allows datetime values stored in the numeric field to be treated as datetime data by Analytics.

- Starts at: byte 15
- Length: 15 bytes
- Decimal places: 6

```
DEFINE FIELD Receipt_timestamp PRINT 15 15 6 NDATETIME PIC "YYYYMMDD.hhmmss"
```

Defining a physical data field that reads mainframe Packed data

You can use the NDATETIME option to create a physical data field that reads date values from a Packed numeric field.

Analytics cannot recognize a date in a number that is compressed into fewer bytes than one per digit and that displays no date format. Consequently, you must unpack the number with NDATETIME to obtain the full number of digits, then specify the date format with PIC.

To accurately indicate which numbers represent the day, the month, and the year, you specify the same date format as the one in the Packed record layout:

```
DEFINE FIELD date_field_name NUMERIC 1 8 0 NDATETIME PIC "YYYYMMDD"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Overwriting fields in a script

You can overwrite a field in a table layout by defining a field that uses the same name as the existing field. If SET SAFETY is ON, a confirmation dialog box appears before overwriting the existing field.

To avoid interrupting a script, you can SET SAFETY to OFF. The existing field is overwritten without additional confirmation.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Supported data types

Data category	Data type
Character	ASCII
	CUSTOM
	EBCDIC
	NOTE
	PCASCII
	UNICODE
Numeric	ACCPAC
	ACL
	BASIC
	BINARY
	FLOAT
	HALFBYTE
	IBMFLOAT
	MICRO
	NUMERIC
	PACKED
	PRINT
	UNISYS
	UNSIGNED
	VAXFLOAT
	ZONED
Datetime	DATETIME
Logical	LOGICAL

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DEFINE FIELD . . . COMPUTED command

Defines a computed field in an Analytics table layout.

Syntax

To define a computed field:

```
DEFINE FIELD field_name COMPUTED expression
```

To define a computed field with optional parameters:

```
DEFINE FIELD field_name COMPUTED
<IF test> <STATIC> <PIC format> <AS display_name> <WIDTH characters> <SUPPRESS>
<field_note>
expression
```

To define a conditional computed field:

```
DEFINE FIELD field_name COMPUTED
*** BLANK_LINE ***
value IF condition
<value IF condition>
<...n>
default_value
```

To define a conditional computed field with optional parameters:

```
DEFINE FIELD field_name COMPUTED
<IF test> <STATIC> <PIC format> <AS display_name> <WIDTH characters> <SUPPRESS>
<field_note>
value IF condition
<value IF condition>
<...n>
default_value
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> Multiline syntax must be structured exactly as shown in the generic syntax above and the examples below.

Parameters

Name	Description
field_name	The name of the computed field. **Note** Field names are limited to 256 upper and lowercase alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. Analytics has a number of reserved keywords that cannot be used as field names. For a complete list, see "Reserved keywords" on page 951.
expression	A valid Analytics expression that defines the value of the computed field.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
STATIC optional	The field displays the same value on every line of the table until a new value is encountered. For example, if there is a Last Name field in the source data where: ◦ the first record displays the value "Smith" ◦ the next five records display blank lines ◦ the seventh record displays the value "Wong" In this case, "Smith" displays on six consecutive lines, then "Wong" displays on the seventh line.
PIC *format* optional	**Note** Applies to numeric fields only. The display format of numeric values in Analytics views and reports. *format* must be enclosed in quotation marks.
AS *display_name* optional	The display name (alternate column title) for the field in the view. If you want the display name to be the same as the field name do not use AS. Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title.
WIDTH *characters*	The display width of the field in characters.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
optional	The specified value controls the display width of the field in Analytics views and reports. The display width never alters data, however it can hide data if it is shorter than the field length. The display width cannot be less than the length of *field_name*, or *display_name*. If you omit WIDTH, the display width is set to the field length in characters. **Note** The characters specified by WIDTH are fixed-width characters. Every character is allotted the same amount of space, regardless of the width of the actual character. By default, views in Analytics use a proportional width font that does not correspond with fixed-width character spacing. If you want a one-to-one correspondence between the WIDTH value and the characters in the view, you can change the **Proportional Font** setting in the **Options** dialog box to a fixed-width font such as Courier New.
SUPPRESS optional	Applies to numeric fields only. Suppresses automatic totaling of numeric computed fields in Analytics reports. Totaling of some numeric fields is not appropriate. For example, a unit cost field, or a discount rate field.
field_note optional	Field note text that is added to the field definition in the table layout. *field_note* must be last, after all other required and optional parameters. The text cannot be multiline. Quotation marks are not required.
value IF *condition*	Conditional computed field only. ◦ ***value*** - the computed field value or expression to use if the *condition* evaluates to true ◦ ***condition*** - the logical test that is evaluated
default_value	Conditional computed field only. The value or expression to use in the computed field if none of the conditions evaluate to true. **Note** The decimal precision of all numeric computed values is controlled by the precision of *default_value*. For example, if you specify a default value of 0.00, all computed values are calculated to two decimal places, and rounded if necessary. For greater precision, increase the number of decimal places in *default_value*.

Examples

Defining a computed field

You define a computed field named **Value** that is the product of the **Cost** and **Quantity** fields:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DEFINE FIELD Value COMPUTED Cost * Quantity
```

Defining a computed field with options

You define a computed field named **Value_03**, with several options defined. You include an IF condition that limits which records are processed by the computed field:

```
DEFINE FIELD Value_03 COMPUTED
IF Product_Class = "03" PIC "($9,999,999.99)" AS "Value Prod Class 3" Value is cost times quantity
Cost * Quantity
```

Defining a conditional computed field

You define a conditional computed field named **Sales_tax** that calculates a different sales tax depending on the state in which the transaction occurred. Transactions that occurred outside the three states have a default sales tax of $0.00.

> **Note**
>
> The second line must be left blank because there are no optional parameters.

```
DEFINE FIELD Sales_tax COMPUTED

.0750 * Sale_amount IF State = "CA"
.0400 * Sale_amount IF State = "NY"
.0625 * Sale_amount IF State = "TX"
0.00
```

Defining a conditional computed field with options

You define a conditional computed field named **Sales_tax_100** that calculates a different sales tax depending on the state in which the transaction occurred. The field only calculates tax on amounts of $100 or greater.

Transactions that occurred outside the three states have a default sales tax of $0.00.

> **Note**
>
> When you specify optional parameters, do not leave any lines blank.

```
DEFINE FIELD Sales_tax_100 COMPUTED
IF Sale_amount >= 100
.0750 * Sale_amount IF State = "CA"
.0400 * Sale_amount IF State = "NY"
.0625 * Sale_amount IF State = "TX"
0.00
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Two types of computed fields

There are two types of computed fields:

- **standard computed field**

  A standard computed field performs the same calculation on every record in a table.

  For example, in an Inventory table you could create a computed field that multiplies the value in the Cost field by the value in the Quantity field to calculate the Inventory Value at Cost for each record.

- **conditional computed field**

  A conditional computed field is capable of performing different calculations on the records in a table, based on a set of conditions that you specify. The calculation performed on a record depends on which condition the record meets.

  For example, in a Transactions table, you could create a conditional computed field that calculates sales tax using a rate that is adjusted based on the state in which the transaction occurred. Conditions such as `IF State = "CA"` and `IF State = "NY"` would test each record to identify which rate to use.

Guidelines for creating a conditional computed field

> **Note**
>
> When defining a conditional computed field, if you do not specify any of the optional parameters on the second line, you must leave the second line **blank**.

In addition to a default value, conditional computed fields require at least one conditional value. You must use the following multiline syntax to define a conditional computed field:

- optional parameters appear on the second line
- if there are no optional parameters, the second line must be left blank
- the first condition statement appears on the third line
- each additional condition statement requires a separate line
- the default value appears on the last line

Overwriting field definitions

You can overwrite a field definition in a table layout by defining a field that uses the same name as the existing field.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

If SET SAFETY is ON, Analytics displays a confirmation dialog box before overwriting the existing field. To avoid interrupting a script, you can SET SAFETY to OFF, and Analytics overwrites the existing field without asking for confirmation.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DEFINE RELATION command

Defines a relation between two Analytics tables.

> **Note**
>
> You can relate up to 18 Analytics tables and access and analyze data from any combination of fields in the related tables as if they existed in a single table. You must specify a separate DEFINE RELATION command for each pair of related tables.

Syntax

```
DEFINE RELATION key_field WITH related_table_name INDEX index_name <AS relation_name>
```

Parameters

Name	Description
key_field	The key field in the parent table. You can select only one key field for each relation. **Note** When creating relations between parent tables and grandchild tables, you must specify a fully qualified key field name in the format *table_name.field_name*. In "Relate three tables" on the facing page, see: Vouchers.created_by
WITH *related_table_name*	The name of the related table.
INDEX *index_name*	The name of the index for the key field in the related table. You must index the related table on the key field before you can relate the table.
AS *relation_name* optional	A unique name for the relation. By default, the name of the child table is used as the relation name. If you are defining additional relations to the same child table, you must specify a unique name.

Examples

Relate two tables

The example below relates the open table to the **Customer** table by using the customer number field

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

(**CustNum**) as the key field:

```
DEFINE RELATION CustNum WITH Customer INDEX Customer_on_CustNum
```

Customer_on_CustNum is the name of the child table index on the key field. A child table index is required when you relate tables.

If the child table index does not already exist when you run the DEFINE RELATION command, an error message appears and the relation is not performed.

> **Tip**
>
> If you define a relation in the Analytics user interface, the child table index is automatically created for you.

Create a child table index before relating two tables

If required, you can create a child table index immediately before relating two tables. The example below shows creating an index for the **Customer** child table before relating the **Ar** table to the **Customer** table.

```
OPEN Customer
INDEX ON CustNum TO Customer_on_CustNum
Open Ar
DEFINE RELATION CustNum WITH Customer INDEX Customer_on_CustNum
```

Relate three tables

The example below relates three tables in the `ACL_Rockwood.ACL` sample project:

- **Vouchers_items** - the parent table
- **Vouchers** - the child table
- **Employees** - the grandchild table

By using the **Vouchers** table as an intermediary table in the relation, you can relate each voucher item with the employee who processed the item.

```
OPEN Vouchers
INDEX ON voucher_number TO "Vouchers_on_voucher_number"
OPEN Vouchers_items
DEFINE RELATION voucher_number WITH Vouchers INDEX Vouchers_on_voucher_number
OPEN Employees
INDEX ON employee_number TO "Employees_on_employee_number"
OPEN Vouchers_items
DEFINE RELATION Vouchers.created_by WITH Employees INDEX Employees_on_employee_number
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Explanation of the syntax logic

1. Open the **Vouchers** table and index it on the **voucher_number** field.
2. Open the **Vouchers_items** table and relate it to the **Vouchers** table using **voucher_number** as the key field.
3. Open the **Employees** table and index it on the **employee_number** field.
4. Open the **Vouchers_items** table and relate it to the **Employees** table using **Vouchers.created_by** as the key field.

> **Note**
>
> **Vouchers.created_by** is available as a key field in the second relation because you already related **Vouchers_items** and **Vouchers** in the first relation.

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DEFINE REPORT command

Creates a new view or opens an existing view.

Syntax

```
DEFINE REPORT view_name
```

Parameters

Name	Description
view_name	The name of a new view or an existing view. ◦ **new view** - creates a blank view with the specified name in the open table Any spaces in the view name are replaced with underscore characters. ◦ **existing view** - opens the specified view in the open table

Examples

Creating a new view

You create a new view called **Q4_AR_review**:

```
DEFINE REPORT Q4_AR_review
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DEFINE TABLE DB command

Defines an Analytics server table by connecting to a database table using AX Connector. You can connect to a Microsoft SQL Server, Oracle, or DB2 database.

Syntax

DEFINE TABLE DB {SOURCE *database_profile* <PASSWORD *num*> <PASSWORD *num*> | SERVER *server_profile* <PASSWORD *num*>} <FORMAT *format_name*> SCHEMA *schema* <TITLED *acl_table_name*> <PRIMARY|SECONDARY> DBTABLE *db_tablename* FIELDS {*field_names*|ALL} <...*n*> <WHERE *condition*> <ORDER *field_names*>

Parameters

SOURCE *database_profile*	The Analytics database profile to use to access the database engine. Database profiles include information required to connect to the database engine, including: ◦ a reference to the associated server profile ◦ the database type ◦ the database name ◦ user account information **Note** DEFINE TABLE DB supports connecting to only the following databases: Microsoft SQL Server, Oracle, or DB2.
PASSWORD *num* optional	The password definition to use. You do not use PASSWORD *num* to prompt for, or specify, an actual password. The password definition refers to a password previously supplied or set using the PASSWORD command, the SET PASSWORD command, or the PASSWORD analytic tag. *num* is the number of the password definition. For example, if two passwords have been previously supplied or set in a script, or when scheduling an analytic, PASSWORD 2 specifies that password #2 is used. For more information about supplying or setting passwords, see: • "PASSWORD command" on page 360 • "SET command" on page 418 • PASSWORD analytic tag The password is only required if the database profile does not contain saved passwords. Use PASSWORD twice after the SOURCE keyword. The first password logs you on to the server, and the second one logs you on to the database.
SERVER *server_profile*	No longer used.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

	Prior to version 10.0 of Analytics, used when connecting to ACL Server Edition for z/OS. From version 10.0 of Analytics, ACL Server Edition for z/OS is no longer included.
FORMAT *format_name* optional	The name of an Analytics table, or table layout file (.layout), with a table layout that you want to use.
SCHEMA *schema*	The schema to connect to. You must enclose the schema name in quotation marks.
TITLED *acl_table_name* optional	The name of the Analytics table to create. *acl_table_name* must be a quoted string. If you omit TITLED, Analytics uses the database table name. When you access more than one table at a time, Analytics uses the name of the first one.
PRIMARY | SECONDARY optional	Use the table as either a primary or secondary table in multi-file commands. If neither option is specified, the default value of PRIMARY is used.
DBTABLE *database_table*	The database table that you want to access. *database_table* must be a quoted string.

FIELDS *field_names* | ALL	The fields to include in the output:
	◦ **FIELDS *field_names*** - use the specified fields
	field_names must be a quoted string.
	◦ **ALL** - use all fields in the table
	To use fields from more than one table:
	a. Enter the first table name followed by the fields from that table.
	b. Enter the next table name followed by the fields from that table.
	c. For each additional table, repeat step b.
	`DBTABLE "DSN1310" FIELDS "Field_A Field_B Field_C"`
	`DBTABLE "DSN2516" FIELDS "Field_L Field_M Field_N"`
	Note
	Using AX Connector, you can access an unlimited number of related tables, but no more than five is recommended. Processing time increases when you access multiple tables.

WHERE *condition* optional	An SQL WHERE clause that limits the data to those records that meet the specified condition. You must use valid SQL syntax entered as a quoted string. When you join tables, Analytics displays the condition of the join in the WHERE clause: `"Table_1.First_name = Table_2.First_name"`
ORDER *field_names* optional	The fields the database engine uses to sort records. *field_names* must be a quoted string. The command takes longer to run when sorting records. Only use ORDER when sorting is important.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Example

You want to access data from a Microsoft SQL Server database via AX Connector. To do this, you use the DEFINE TABLE DB command. You include the SOURCE parameter to connect to AX Connector through a database profile:

```
DEFINE TABLE DB SOURCE "SQLServer_Audit" SCHEMA "HR" TITLED "Payroll" DBTABLE "HR.Employee" FIELDS "EmployeeID" DBTABLE "HR.EmployeePayHistory" FIELDS "Rate PayFrequency" WHERE "HR.Employee.EmployeeID=HR.EmployeePayHistory.EmployeeID"
```

Remarks

How it works

The Analytics server table is defined as a query that uses a database profile to connect to a database table.

Suppressing the time portion of datetime values

Preface the DEFINE TABLE DB command with the SET SUPPRESSTIME command to suppress the time portion of datetime values.

Using SET SUPPRESSTIME ON is for pre-version-10.0 Analytics scripts that assume the time portion of datetime values will be truncated. If SET SUPPRESSTIME ON is not added to these scripts, they cannot run in the datetime-enabled version of Analytics.

For more information, see the "SET SUPPRESSTIME" section in "SET command" on page 418.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DEFINE VIEW command

Defines a new view or overwrites an existing view.

Syntax

DEFINE VIEW *view_name* <RLINES *n*> <ALL> <SUPPRESS> <SUMMARIZED> <IF *test*> <WHILE *test*> <HEADER *header_text*> <FOOTER *footer_text*> <TO *report_file_name* <HTML>> <OK>

Parameters

Name	Description
view_name	The name of the view to create or overwrite. **Note:** View names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
RLINES *n* optional	The line spacing for detail records in views and reports. By default, detail lines are single spaced.
ALL optional	All fields in the active Analytics table layout are added to the view.
SUPPRESS optional	Suppresses blank detail lines in reports generated from the view. When the report is generated the blank detail lines will be omitted from the output. This option applies to reports based on multiline views.
SUMMARIZED optional	Specifies that reports generated from the view should include subtotals and totals, but not include the detail lines. The subtotals are generated based on the break fields defined in the view. If this option is not selected, the report includes detail lines, as well as subtotals for each of the specified break fields.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.
TO *report_filename* HTML optional	The filename and type for reports created from this view. Use the HTML keyword to save reports generated from this view as HTML files (.htm). By default, generated reports are output as ASCII text files.
OK optional	Deletes or overwrites items without asking you to confirm the action.

Examples

Creating a view

You open the **Ar** table and create a view called **AR_Report**, which includes all of the fields in the table lay-out:

```
OPEN Ar
DEFINE VIEW AR_Report HEADER "AR Report" ALL OK
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DELETE command

Deletes an Analytics project item, a field from a table layout, a variable, one or more table history entries, a relation between tables, or a file in a Windows folder. Also removes a column from a view.

Syntax

Purpose	Syntax
To delete an Analytics project item	DELETE *item_type item_name* <OK>
To delete a field from a table layout	DELETE *field_name* <OK>
To remove a column from a view	DELETE COLUMN *view_name field_name* <ALL> <OK>
To delete a variable or all variables	DELETE {*variable_name*|ALL} <OK>
To delete the history for the current Analytics table	DELETE HISTORY *<retain_history_entries>* <OK>
To delete a relation between two tables	DELETE RELATION <*child_table_name*|*relation_name*> <OK>
To delete a file	DELETE *file_name* <OK>
To delete all record notes, and the auto-generated **RecordNote** field, from the open table	DELETE NOTES <OK>

Parameters

Name	Description
item_type item_name	The type and name of the item to delete. Specify one of the following item types: ◦ **FOLDER** – the specified project folder and all its contents

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ **FORMAT** - the specified table layout, all its views, and its associated indexes and relations Any other table layouts for the associated table are retained. The data file (.fil) associated with the table layout is not deleted unless the **Delete Data File with Table** option is selected in the **Table** tab in the **Options** dialog box (**Tools > Options**). You can also use the SET DELETE_FILE {ON|OFF} command in a script or on the command line to turn this option on or off. For more information, see "SET command" on page 418. **Caution** Use caution when turning on the **Delete Data File with Table** option. It may be an original data file that is deleted along with the table layout. Data files are deleted outright. They are not sent to the Windows Recycle Bin. ◦ **REPORT** - the specified view You cannot delete a view if it is currently active. ◦ **COLUMN** - the specified column ◦ **SCRIPT (or BATCH)** - the specified script ◦ **WORKSPACE** - the specified workspace ◦ **INDEX** - the specified index ◦ **NOTES** - all record notes from the open table, and the **RecordNote** field from the table layout
field_name ALL	**Delete a field** The name of the field to delete from the current Analytics table layout. You can delete a field from a table layout even if the field is included in the current view. **Note** You cannot delete a field referenced by a computed field unless you first delete the computed field. **Remove a column** The name of the column to remove from the specified view. **Note** Use the physical field name, not the column display name. ◦ **ALL included** - removes all occurrences of the column in the view ◦ **ALL omitted** - removes the first (leftmost) occurrence of the column in the view
view_name	The name of the view to remove a column from.
variable_name | ALL	The name of the variable to delete. Use ALL to delete all variables. If you specify ALL, all occurrences of the following types of the variables are deleted from the project: ◦ system variables

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ temporary user-defined variables ◦ permanent user-defined variables **Note** You cannot delete a variable referenced by a computed field unless you first delete the computed field.
HISTORY *retain_history_entries*	Deletes all table history entries except for the number of most recent entries specified by *retain_history_entries*. Omit *retain_history_entries* to delete all entries.
RELATION *child_table_name* | *relation_name*	Deletes any relation that has no dependent relations and no related fields referenced in either the active view or in an active computed field. Use the options to specify which relation to delete: ◦ ***child_table_name*** - use when the relation was not specifically named (default name when a relation is created) ◦ ***relation_name*** - use when the relation was specifically named when it was created. Otherwise, use *child_table_name* If you do not use either option, the last relation that was defined gets deleted.
file_name	The name of a physical file to delete. You can specify an absolute or relative path to a file you want to delete. If the path has spaces, enclose it in double quotation marks.
OK optional	Deletes items without presenting a confirmation dialog box.

Examples

Deleting a date field

You delete the **Date** field from the table layout associated with the **Ar** table:

```
OPEN Ar
DELETE Date
```

Deleting multiple columns from a view

You delete two columns from the **AR_Report** view associated with the **Ar** table. You specify OK for both DELETE commands so that no confirmation prompt is displayed when the script runs:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
OPEN Ar
DELETE COLUMN AR_Report Date OK
DELETE COLUMN AR_Report Invoice_Date OK
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DIALOG command

Creates a custom dialog box that interactively prompts users for one or more script input values. Each input value is stored in a named variable.

> **Note**
>
> Using the DIALOG command to enter passwords is not secure. You should use the "PASSWORD command" on page 360 instead.
>
> The DIALOG command is not supported in AX Server analytics.
>
> You can create a basic interactive dialog box with the "ACCEPT command" on page 54.

> **Tip**
>
> The easiest way to create custom dialog boxes is with the **Dialog Builder**. For more information, see Creating custom dialog boxes.

Syntax

DIALOG (DIALOG TITLE *title_text* WIDTH *pixels* HEIGHT *pixels*) (BUTTONSET TITLE "&OK;&Cancel" AT *x_pos y_pos* <WIDTH *pixels*> <HEIGHT *pixels*> DEFAULT *item_num* <HORZ>) < [label_syntax]|[text_box_syntax]|[check_box_syntax]|[radio_button_syntax]|[drop_down_list_syntax]| [project_item_list_syntax]> <...*n*>

label_syntax ::=
(TEXT TITLE *title_text* AT *x_pos y_pos* <WIDTH *pixels*> <HEIGHT *pixels*> <CENTER|RIGHT>)

text_box_syntax ::=
(EDIT TO *var_name* AT *x_pos y_pos* <WIDTH *pixels*> <HEIGHT *pixels*> <DEFAULT *string*>)

check_box_syntax ::=
(CHECKBOX TITLE *title_text* TO *var_name* AT *x_pos y_pos* <WIDTH *pixels*> <HEIGHT *pixels*> <CHECKED>)

radio_button_syntax ::=
(RADIOBUTTON TITLE *value_list* TO *var_name* AT *x_pos y_pos* <WIDTH *pixels*> <HEIGHT *pixels*> <DEFAULT *item_num*> <HORZ>)

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

drop_down_list_syntax ::=
(DROPDOWN TITLE *value_list* TO *var_name* AT *x_pos y_pos* <WIDTH *pixels*> <HEIGHT *pixels*> <DEFAULT *item_num*>)

project_item_list_syntax ::=
(ITEM TITLE *project_item_category* TO *var_name* AT *x_pos y_pos* <WIDTH *pixels*> <HEIGHT *pixels*> <DEFAULT *string*>)

Parameters

General parameters

Name	Description
DIALOG TITLE *title_text*	Creates the main dialog box and the dialog box title. *title_text* must be specified as a quoted string.
BUTTONSET TITLE "&OK;&Cancel"	The labels for the **OK** and **Cancel** buttons in the dialog box. The values should normally not be edited, but if you do edit the values make sure that the positive value comes before the negative value. For example: "&Yes;&No"
WIDTH *pixels*	The width of the individual control, or the width of the dialog box if specified for the DIALOG control. The value is specified in pixels. If no value is specified for a control the width is calculated based on the longest value contained by the control.
HEIGHT *pixels*	The height of the individual control, or the height of the dialog box if specified for the DIALOG control. The value is specified in pixels.
AT *x_posy_pos*	The location of the top left corner of the control in the custom dialog box: ◦ *x_pos* is the horizontal distance in pixels from the left-hand side of the dialog box ◦ *y_pos* is the vertical distance in pixels from the top of the dialog box
DEFAULT *item_num*	The numeric value that corresponds to the BUTTONSET value that you want to select as the default. For example, if the BUTTONSET values are "&OK;&Cancel", specify DEFAULT 1 to select OK by default.
HORZ optional	Displays the values for the BUTTONSET control horizontally. Values are displayed vertically by default.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> For most of the control types, the DIALOG command creates a variable to store user input. You cannot use non-English characters, such as é, in the names of variables that will be used in variable substitution. Variable names that contain non-English characters will cause the script to fail.
>
> By default, some of the DIALOG variables are created as character variables. If you use a character variable to store numeric or datetime values, you must convert the variable to the required data type in subsequent processing in a script. For more information, see "Input data type" on page 157.

Label parameters

Name	Description
TEXT	Creates a text label to identify, notify, or instruct.
TITLE *title_text*	The control label. *title_text* must be specified as a quoted string.
CENTER | RIGHT optional	The alignment of the text in the control. If you omit CENTER or RIGHT, left alignment is used by default.

Text box parameters

Name	Description
EDIT	Creates a text box for user input.
TO *var_name*	The name of the character variable that stores the input specified by the user. If the variable already exists, the specified value is assigned. If the variable does not exist, it is created, and the specified value is assigned.
DEFAULT *string* optional	The default text string to display in the control. *string* must be specified as a quoted string.

Check box parameters

Name	Description
CHECKBOX	Creates a check box to present an option to the user.
TITLE *title_text*	The control label. *title_text* must be specified as a quoted string.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
TO *var_name*	The name of the logical variable that stores the True or False value specified by the user. If the variable already exists, the specified value is assigned. If the variable does not exist, it is created, and the specified value is assigned.
CHECKED optional	Sets the control to checked by default.

Radio button parameters

Name	Description
RADIOBUTTON	Creates radio buttons to present mutually exclusive options to the user.
TITLE *value_list*	The list of values displayed for the control. The values must be specified as a quoted string. Separate each value with a semi-colon (;).
TO *var_name*	The name of the numeric variable that stores the numeric position of the radio button value selected by the user. If the variable already exists, the specified value is assigned. If the variable does not exist, it is created, and the specified value is assigned.
DEFAULT *item_num* optional	The numeric value that corresponds to the list item that you want to select as the default. For example, if *value_list* is "Red;Green;Blue", specify `DEFAULT 2` to select Green by default.
HORZ optional	Displays the values for the control horizontally. Values are displayed vertically by default.

Drop-down list parameters

Name	Description
DROPDOWN	Creates a drop-down list to present a list of options to the user.
TITLE *value_list*	The list of values displayed for the control. The values must be specified as a quoted string. Separate each value with a semi-colon (;).
TO *var_name*	The name of the character variable that stores the drop-down list value selected by the user. If the variable already exists, the specified value is assigned. If the variable does not exist, it is created, and the specified value is assigned.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
DEFAULT *item_num* optional	The numeric value that corresponds to the list item that you want to select as the default. For example, if *value_list* is "Red;Green;Blue", specify `DEFAULT 2` to select Green by default when the drop-down list is displayed.

Project item list parameters

Name	Description
ITEM	Creates a project item list to present a list of Analytics project items, such as fields, to the user.
TITLE *project_item_category*	The category of project item to include in the control. You can specify one or more categories. The user can select a single value from the project item list. Enclose *project_item_category* in quotation marks, with no space or punctuation between categories. For the codes used to specify categories, see "Codes for project item categories" on the next page. **Note** Do not mix dissimilar categories in the same ITEM control, unless you have a reason for doing so. For example, do not mix tables and fields. The resulting project item list is potentially confusing for the user.
TO *var_name*	The name of the character variable that stores the name of the project item selected by the user. If the variable already exists, the specified value is assigned. If the variable does not exist, it is created, and the specified value is assigned.
DEFAULT *string* optional	The exact name of the project item that you want to select as the default. *string* must be specified as a quoted string.

Examples

Prompting the user for a table and script

In your script, you need to prompt the user to select the Analytics table and script to use to run an analysis .

You specify that the **Metaphor_Inventory_2012** table from the `ACL_Demo.acl` project is selected by default as the Analytics table, but the user can select any table in the project.

The script to run must also be selected from the list of scripts in the Analytics project:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DIALOG (DIALOG TITLE "Inventory analysis" WIDTH 500 HEIGHT 200 ) (BUTTONSET TITLE "&OK;&Cancel" AT 370 12 DEFAULT 1 ) (TEXT TITLE "Choose the Analytics project items to analyze." AT 50 16 ) (TEXT TITLE "Table:" AT 50 50 ) (ITEM TITLE "f" TO "v_table" AT 50 70 DEFAULT "Metaphor_Inventory_2012" ) (TEXT TITLE "Script:" AT 230 50 ) (ITEM TITLE "b" TO "v_script" AT 230 70 )
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Interactivity

Use DIALOG to create an interactive script. When the DIALOG command is processed, the script pauses and a dialog box is displayed that prompts the user for input that Analytics uses in subsequent processing.

You can create separate dialog boxes that prompt for one item at a time, or you can create one dialog box that prompts for multiple items.

ACCEPT versus DIALOG

The ACCEPT command allows you to create a basic interactive dialog box that can have one or more of the following types of controls:

- text box
- project item list

For basic interactivity, ACCEPT may be all you need. For more information, see "ACCEPT command" on page 54.

Codes for project item categories

Use the following codes to specify the category of project item to display in a project item list.

Project categories

Code	Category
f	Tables
b	Scripts
i	Indexes

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Code	Category
r	Views and reports
w	Workspaces

Field categories

Code	Category
C	Character fields
N	Numeric fields
D	Datetime fields
L	Logical fields

Variable categories

Code	Category
c	Character variables
n	Numeric variables
d	Datetime variables
l	Logical variables

Input data type

Some of the controls in the DIALOG command store user input in character variables. If you need numeric or datetime input, you can use the VALUE( ) or CTOD( ) functions to convert the contents of a character variable to a numeric or datetime value:

```
SET FILTER TO BETWEEN(%v_date_field%, CTOD(%v_start_date%), CTOD(%v_end_date%))
```

In the example, the start and end dates for this filter are stored as character values. They must be converted to date values in order to be used with a date field that uses a Datetime data type.

Enclosing the variable name in percent signs (%) substitutes the character value contained by the variable for the variable name. The CTOD( ) function then converts the character value to a date value.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Position of the DIALOG command

It is good practice to place all DIALOG commands at the beginning of a script, if possible. If you ask for all input at the beginning, the script can then run unimpeded once the user enters the necessary information.

> **Note**
>
> You cannot use the DIALOG command inside the GROUP command.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DIRECTORY command

Generates a list of files and folders in the specified directory.

Syntax

DIRECTORY <*file_spec*> <SUPPRESS> <SUBDIRECTORY> <APPEND> <TO *table_name*|*file-name*>

Parameters

Name	Description
file_spec optional	The Windows folder or files to list and display information for. You can use the asterisk wildcard (*) to list all files with a particular extension, all files that start with a particular string, or all files in a folder. For example: ◦ `*.fil` - lists all files with the .fil extension (Analytics data files) ◦ `Inv*.*` - lists all files that begin with "Inv" regardless of what file extension they have ◦ `Results\*` or `Results\*.*` - lists all files in the Results folder To limit the files listed to a particular folder, you can specify a path relative to the Analytics project folder, or specify a full path. For example: ◦ `Results\*.*` - displays the contents of the Results subfolder in the Analytics project folder ◦ `C:\ACL Data\Results\*.*` - displays the contents of the specified folder **Note** The wildcard character cannot be used in intermediary levels of a specified file path. It can only be used in the final level of the path, as shown above. Paths or file names that contain spaces must be enclosed in double quotation marks. If you use *file_spec*, it must be placed before any of the other parameters. If *file_spec* appears in any other position, the DIRECTORY command is not processed and an error is generated. If you omit *file_spec*, all files in the folder containing the Analytics project are listed. You cannot use any of the other parameters if you omit *file_spec*.
SUPPRESS optional	Suppresses path information in the output, leaving only the file names and properties.
SUBDIRECTORY optional	Includes the contents of subfolders in the directory listing. For example, if *file_spec* specifies `Results\*.fil`, the Results folder, and all subfolders

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	contained in the Results folder, are searched for `.fil` files. Depending on the number of subfolders and files that need to be listed, using SUBDIRECTORY may result in a delay while the subfolders are searched. Analytics displays a dialog box showing progress of the command.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
TO *table_name* | *filename* optional	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"` If you omit TO, the directory listing appears in the Analytics display area.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Different options for listing files

The ability to list files is useful for ad hoc investigation, and for incorporating in scripting.

A number of different options for listing files with the DIRECTORY command appear below.

List all files

Lists all the files in the folder containing the Analytics project:

```
DIRECTORY
```

List all the files of a specific type

Lists all the .fil files (Analytics data files) in the folder containing the Analytics project:

```
DIRECTORY *.fil
```

Use wildcards to list files

Lists all the file names beginning with "Inv" in the folder containing the Analytics project:

```
DIRECTORY Inv*.*
```

List all the files in a subfolder relative to the Analytics project folder

Lists all the files in the `Results` subfolder in the folder containing the Analytics project:

```
DIRECTORY "Results\*"
```

List all the files in a specified folder

Lists all the files in the `Results` subfolder:

```
DIRECTORY "C:\ACL Data\Results\*"
```

List all the files of a specific type in a specified location

Lists all the .fil files (Analytics data files) in the specified folder and any subfolders:

```
DIRECTORY "C:\ACL Data\Results\*.fil" SUBDIRECTORY
```

List all the files in a specified folder and output the list to an Analytics table

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Lists all the files in the `Results` folder and outputs the list to an Analytics table in the folder containing the Analytics project:

```
DIRECTORY "C:\ACL Data\Results\*" TO Results_Folder_Contents.fil
```

The new table `Results_Folder_Contents` is added to the open project.

List all the files in one folder and output the list to an Analytics table in another folder

Lists all the files in the `ACL Data\Results` folder and outputs the list to an Analytics table in the `GL Audit 2014\Results` folder:

```
DIRECTORY "C:\ACL Data\Results\*" TO "C:\ACL Projects\GL Audit 2014\Results\Results_Folder_Contents.fil"
```

The new table `Results_Folder_Contents` is added to the open project. The associated data file (`Results_Folder_Contents.fil`) is created in the specified output folder, which may or may not be the folder containing the Analytics project.

Remarks

Properties displayed by DIRECTORY

The DIRECTORY command is similar to the DIR command in Windows. In addition to listing files and sub-folders in a folder, the DIRECTORY command also displays the following file and folder properties:

◦ File Size ◦ Attributes	◦ Create Date ◦ Create Time	◦ Access Date ◦ Access Time	◦ Modified Date ◦ Modified Time ◦ the total number of files and folders that match the specified criteria

Uses for DIRECTORY in a script

When used in a script, the DIRECTORY command provides the ability to examine the file system. For example, you could use DIRECTORY in conjunction with other commands to detect the presence or absence of files, check a file's size, or make decisions based on other file properties.

Outputting the results of DIRECTORY

You can run the command from the command line to display a directory listing on screen, or save the listing to an Analytics table or .txt file.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

How to open table-based results of DIRECTORY

The DIRECTORY command does not include the OPEN parameter. If you are using the command in a script and outputting the results to an Analytics table, and you want to open the resulting table, follow the DIRECTORY command with the OPEN command. For example:

```
DIRECTORY "C:\ACL Data\Results\*" TO Results_Folder_Contents.fil
OPEN Results_Folder_Contents
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DISPLAY command

Displays information about the specified Analytics item type. Can also display the result of an expression, or the output of a function.

Syntax and parameters

Syntax	Purpose
DISPLAY	Displays the field definitions, and any related child tables, for the currently active Analytics table.
DISPLAY OPEN	Displays a list of open Analytics tables and project files. ◦ **Analytics tables** - displays the name of the source data file, not the table layout name. ◦ **multiple-table mode** - the source data file identified as PRIMARY is associated with the currently active table. ◦ **related tables** - if the parent table is open, displays the source data file for both the parent and the child tables, even if the child table is not open in the View tab.
DISPLAY {<PRIMARY>|SECONDARY}	Displays the name and table layout information for the primary or secondary table. ◦ **PRIMARY (or no keyword specified)** - display information for the currently active table. ◦ **SECONDARY** - display information for the secondary table. In multiple-table mode, SECONDARY refers to the secondary table associated with the currently active table. The information displayed includes: ◦ the table layout name ◦ the source data file name ◦ any relations between the table and other tables ◦ the field definition information from the table layout
DISPLAY HISTORY	Displays the table history for the currently active Analytics table. **Note** A table may or may not have associated table history.
DISPLAY RELATION	Displays relation information for the currently active Analytics table: ◦ the names of any child tables ◦ key field names ◦ index names

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Syntax	Purpose
DISPLAY {*variable_name*|VARIABLES}	Displays the value of a single variable or all variables. ◦ ***variable_name*** - the name of a single variable to display the value of. ◦ **VARIABLES** - display the values of all system and user-defined variables, and the remaining memory available to store variables.
DISPLAY VERSION	Displays information in the following format about the installed version of Analytics: ◦ **Version** - *major version number.minor version number* ◦ **Patch** - *patch number* ◦ **Type** - 000 (non-Unicode), or 001 (Unicode) edition of Analytics ◦ **Build** - *software build number*
DISPLAY {DATE|TIME}	Displays the current operating system date and time. **DATE | TIME** - specify either keyword. The two keywords do the same thing.
DISPLAY {FREE|SPACE}	Displays the amount of physical memory (RAM) available for use by Analytics. The amount displayed does not include memory reserved for variables. By default, Analytics reserves 60 KB of physical memory to store variables, but the amount is automatically increased as necessary. **FREE | SPACE** - specify either keyword. The two keywords do the same thing.
DISPLAY *expression*	Displays the result of an expression. ***expression*** - the expression to display the result of.
DISPLAY *function*	Displays the output of a function. ***function*** - the function to display the output of.

Examples

Display the layout of an Analytics table

Displaying the layout of a table can be useful in a number of circumstances. For example, you may want to combine two or more tables, and you need to examine field lengths and data types.

The example below displays the layout of the Ap_Trans table:

```
OPEN Ap_Trans
DISPLAY
```

The DISPLAY command produces the output to screen shown below.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> If you enter DISPLAY directly in the Analytics command line, the output appears immediately.
>
> If you run DISPLAY in a script, double-click the corresponding **DISPLAY** entry in the command log to display the output.

Output to screen

Relationship

'Vendor' related by 'Vendor_No' using index 'Vendor_on_Vendor_No'

File

'Ap_Trans.fil' (format 'Ap_Trans') is your PRIMARY file.

The record length is 59

Fields

Name	Type	Start	Length	Decimals	Field explanation
Vendor_No	ASCII	1	5		AS "Vendor;Number" WIDTH 7
Invoice_No	ASCII	6	15		AS "Invoice;Number"
Invoice_Date	DATETIME	21	8		PICTURE "MM/DD/YY" AS "Invoice;Date" WIDTH 8
Invoice_ Amount	NUMERIC	29	12	2	PICTURE "(9,999,999.99)" AS "Invoice;Amount" WIDTH 12
Prodno	ASCII	41	9		AS "Product;Number"
Quantity	MICRO	50	4	0	PICTURE "(9,999,999)"
Unit_Cost	NUMERIC	54	6	2	PICTURE "(9,999,999)" AS "Unit;Cost" SUPPRESS

Display the values of all variables in an Analytics project

DISPLAY VARIABLES generates the same information that appears in the **Variables** tab in the **Navigator**. One benefit of using DISPLAY VARIABLES is that you can copy-and-paste the displayed information.

The example below creates two user-defined variables and two system variables and then displays the values of the variables:

```
ASSIGN v_table_name = "Ap_Trans"
ASSIGN v_field_name = "Invoice_Amount"
OPEN %v_table_name%
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
TOTAL FIELDS %v_field_name%
DISPLAY VARIABLES
```

The DISPLAY command produces the output to screen shown below.

> **Note**
>
> If you enter DISPLAY VARIABLES directly in the Analytics command line, the output appears immediately.
>
> If you run DISPLAY VARIABLES in a script, double-click the corresponding **DISPLAY VARIABLES** entry in the command log to display the output.

Output to screen

Name	Type	Value
TOTAL1	N	278,641.33
OUTPUTFOLDER	C	"/Tables/Accounts_Payable"
v_field_name	C	"Invoice_Amount"
v_table_name	C	"Ap_Trans"

Display the result of an expression

For the selected record, the example below displays the result of multiplying the value in the Sale_Price field by the value in the Quantity_on_Hand field:

```
DISPLAY Sale_Price * Quantity_on_Hand
```

Display the output of a function

For the selected record, the example below displays the number of days that have elapsed since the date in the Invoice_Date field:

```
DISPLAY AGE(Invoice_Date)
```

Remarks

Location of command results

DISPLAY run from the Analytics command line - the results are displayed on screen.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DISPLAY executed in a script – the results are written to the Analytics command log. You can double-click the command log entry to display the results on screen.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DO REPORT command

Generates the specified Analytics report.

Syntax

```
DO REPORT report_name
```

Parameters

Name	Description
report_name	The name of the view to generate and print as a report.

Example

Printing the default view

You open the **AP_Trans** table and print the default view:

```
OPEN AP_Trans
DO REPORT Default_View
```

Remarks

Running DO REPORT on the command line vs in a script

The settings used to print the report depend on where you run the command:

- **from the command line** - the **Print** dialog box opens for you to select the pages to print and configure other options for the report
- **in a script** - the report is printed immediately using the default settings for the report

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DO SCRIPT command

Executes a secondary script, or an external script, from within an Analytics script.

Syntax

DO <SCRIPT> *script_name* {<IF *test*>|<WHILE *test*>}

Parameters

Name	Description
SCRIPT *script_name*	The name of the script to run. You can run secondary scripts in the Analytics project, or external scripts stored in text files with extensions such as .aclscript, .txt. or .bat. You can specify a file path to an external script. You must enclose the path in quotation marks if it contains any spaces. **Note** You cannot call a script that is already running. For example, if ScriptA calls ScriptB, ScriptB cannot call ScriptA. ScriptA is still running while it waits for ScriptB to complete.
IF *test* optional	A conditional expression that is evaluated one time to determine if the script should be executed. If the condition evaluates to true the script runs, otherwise it does not. Cannot be used with WHILE in the same command. If both are used, WHILE is ignored when the script is processed. A comment is entered in the log, but the script does not stop executing.
WHILE *test* optional	A conditional expression that is evaluated after the script runs to determine if the script should be executed again. If the test evaluates to true the script runs again, otherwise it does not. **Note** If you use WHILE, ensure that your test eventually evaluates to false. If you do not, the script enters an infinite loop. In case of an infinite loop, press the **Esc** key to cancel script processing. Cannot be used with IF in the same command. If both are used, WHILE is ignored when the script is processed. A comment is entered in the log, but the script does not stop executing.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Executing a subscript repeatedly until the input is validated

You have a subscript that gathers user input using a dialog box. It does the following:

1. Prompts the user for the required values.
2. Checks the user input.
3. Sets the *v_validated* variable to true when the input values are validated.

To ensure that the user enters valid input, you use DO SCRIPT and include a WHILE condition so that the script repeats this command until input is validated. Once the value of the variable changes, the main script moves on to the next command:

```
DO SCRIPT GetUserInput WHILE v_validated = F
```

Running a subscript from a shared location

You maintain utility subscripts in a shared location. When you require one of the subscripts during an analysis, you reference it using the full path to the shared location:

```
DO SCRIPT "C:\My utility scripts\GetUserInput.aclscript"  WHILE v_validated = F
```

Remarks

Related commands

DO SCRIPT is the equivalent of the DO BATCH command found in scripts created with earlier releases of Analytics.

You cannot include the DO SCRIPT command inside a GROUP command.

Usefulness of an external script

Storing a script externally and calling it from within an Analytics script is useful if you want to reuse the same subscript in different Analytics scripts and projects.

You can store a single copy of the script in one location, and make updates to it in one place, rather than maintaining it in multiple locations.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DUMP command

Displays the contents of a file, or the current record, in hexadecimal, ASCII, and EBCDIC character encodings.

> **Note**
>
> This command can only be entered in the command line. It cannot be used in scripts.

Syntax

```
DUMP {RECORD|file_name} <SKIP bytes> <COLUMN bytes> <HORIZONTAL>
```

Parameters

Name	Description
RECORD	Displays the contents of the selected record. Required if you do not specify a *file_name*.
file_name	The name of the file you want to display. Required if you do not specify RECORD. **Note** To display the character encodings for an Analytics table you must specify the name of the source data file and the file extension. For example: `Ap_Trans.fil`
SKIP *bytes* optional	The number of bytes to bypass before the dump begins. The default is 0.
COLUMN *bytes* optional	In the output, the width of the columns in bytes. **Note** The number specified by *bytes* refers to the bytes contained by the Analytics record or table. The encoded characters in the output may not have a one-to-one relation with the characters in the view. For example, the hexadecimal encoding for the number 1 is 31. The default is 16 bytes for each column in a vertical display, and 64 bytes for the single column in a horizontal display. The maximum number of bytes you can specify is 255.
HORIZONTAL	Displays the character encodings in horizontal rows rather than in side-by-side vertical blocks (the default).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
optional	

Examples

Display the character encoding of the Inventory table

The example below displays the hexadecimal, ASCII, and EBCDIC character encoding of the data in the Inventory table. The output is arranged in horizontal rows (hexadecimal encoding uses a double row). Each row represents 97 bytes of data in the Analytics table:

```
DUMP Inventory.fil COLUMN 97 HORIZONTAL
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DUPLICATES command

Detects whether duplicate values or entire duplicate records exist in an Analytics table.

Syntax

DUPLICATES <ON> {*key_field* <D> <...*n*>|ALL} <OTHER *field* <...*n*>|OTHER ALL> <UNFORMATTED> <ADDGROUP> <PRESORT> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <APPEND> <OPEN> <TO {SCREEN|*table_name*|*filename*|PRINT}> <LOCAL> <HEADER *header_text*> <FOOTER *footer_text*> <ISOLOCALE *locale_code*>

Parameters

Name	Description
ON *key_field* D <...*n*> | ALL	The key field or fields, or the expression, to test for duplicates. ◦ ***key_field*** - use the specified field or fields If you test by more than one field, records identified as duplicates require identical values in every specified field. Include D to sort the key field in descending order. The default sort order is ascending. ◦ **ALL** - use all fields in the table If you test by all the fields in a table, records identified as duplicates must be entirely identical. An ascending sort order is the only option for ALL. **Note** Undefined portions of records are not tested.
OTHER *field <...n>* | OTHER ALL optional	One or more additional fields to include in the output. ◦ ***field <...n>*** - include the specified field or fields ◦ **ALL** - include all fields in the table that are not specified as key fields
UNFORMATTED optional	Suppresses page headings and page breaks when the results are output to a file.
ADDGROUP optional	Include the **Group Number** field (GROUP_NUM) in the output table. The **Group Number** field assigns a sequentially incremented number to each unique group of duplicates.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Tip The ability to reference groups of duplicates by number can be useful when you analyze data in the output table.
PRESORT optional	Sorts the table on the key field before executing the command. **Note** You cannot use PRESORT inside the GROUP command.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
TO SCREEN | *table_name* | *filename* | PRINT optional	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"` ◦ **PRINT** - sends the results to the default printer
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
ISOLOCALE *locale_code* optional	**Note** Applicable in the Unicode edition of Analytics only. The system locale in the format *language_country*. For example, to use Canadian French, enter fr_ca. Use the following codes: ◦ **language** - ISO 639 standard language code ◦ **country** - ISO 3166 standard country code If you do not specify a country code, the default country for the language is used. If you do not use ISOLOCALE, the default system locale is used.

Analytics output variables

Name	Contains
GAPDUP*n*	The total number of gaps, duplicates, or fuzzy duplicate groups identified by the command.

Examples

Test for duplicate values in one field

The following example:

- tests for duplicate values in the **Invoice_Number** field
- outputs any records that contain duplicate invoice numbers to a new Analytics table

```
DUPLICATES ON Invoice_Number OTHER Vendor_Number Invoice_Date Invoice_Amount
PRESORT TO "Duplicate_Invoices.FIL"
```

Test for duplicate values in two or more fields in combination

The following example:

- tests for duplicate combinations of values in the **Invoice_Number** and **Vendor_Number** fields
- outputs any records that contain the same invoice number and the same vendor number to a new Analytics table

The difference between this test and the previous test is that a identical invoice number from two different vendors is not reported as a false positive.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DUPLICATES ON Invoice_Number Vendor_Number OTHER Invoice_Date Invoice_Amount
PRESORT TO "Duplicate_Invoices.FIL"
```

Test for duplicate records

The following examples:

- test for duplicate values in every field in an Inventory table
- output any entirely identical records to a new Analytics table

```
DUPLICATES ON ProdNum ProdClass Location ProdDesc ProdStatus UnitCost CostDate
SalePrice PriceDate PRESORT TO "Duplicate_Inventory_Items.FIL"
```

You can simplify the syntax by using `ALL` :

```
DUPLICATES ON ALL PRESORT TO "Duplicate_Inventory_Items.FIL"
```

Filter duplicates output table by group number

You use several key fields in combination to test an accounts payable table for duplicate records:

- vendor number
- invoice number
- invoice date
- invoice amount

You want to filter the resulting duplicates output table so that only some of the groups of duplicates are subject to additional processing.

To create a filter using the combination of key fields would be laborious. For example:

```
SET FILTER TO ((Vendor_No = "11475") AND (Invoice_No = "8752512") AND (Invoice_Date =
`20191021`) AND (Invoice_Amount = 7125.80)) OR ((Vendor_No = "12130") AND (Invoice_No =
"589134") AND (Invoice_Date = `20191117`) AND (Invoice_Amount = 10531.71)) OR ((Vendor_No
= "13440") AND (Invoice_No = "5518912") AND (Invoice_Date = `20191015`) AND (Invoice_Amount
= 11068.20))
```

Instead, you achieve the same result by creating a filter based on group number:

```
SET FILTER TO MATCH(GROUP_NUM, 3 , 8, 11)
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Sorting and duplicates

Generally, you should run the duplicates command only on a sorted key field or fields. Duplicate values in a key field are only found if they are immediately adjacent.

If you run the duplicates command on an unsorted key field, non-adjacent duplicate values are not reported as duplicates. If two or more clusters of the same duplicate value exist, they are reported as duplicates, but in separate groups.

Depending on your analysis goal, it may make sense to run the duplicates command on an unsorted key field. For example, you may want to find only those duplicate values that are immediately adjacent in the source table, and ignore duplicate values that are non-adjacent.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ESCAPE command

Terminates the script being processed, or all scripts, without exiting Analytics.

Syntax

```
ESCAPE <ALL> <IF test>
```

Parameters

Name	Description
ALL optional	Terminates all active scripts. If omitted, the current script is terminated.
IF *test* optional	A test that must evaluate to true before the command is executed. If the test evaluates to false the command does not run.

Examples

Terminating a script conditionally

You count the number of records in a table, and use the ESCAPE command to terminate the script if the number of records is less than 100:

```
COUNT
ESCAPE IF COUNT1 < 100
```

Remarks

When to use ESCAPE

Use ESCAPE to halt the execution of a script or subscript based on a condition, or to stop the execution of all running scripts.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Using ESCAPE in subscripts

If you execute ESCAPE inside a subscript, then the subscript stops executing and the main script resumes processing from the DO SCRIPT command that invoked the subscript.

If you include the ALL option in the ESCAPE command in the subscript, then both the subscript and the main script stop processing:

```
ESCAPE ALL
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EVALUATE command

For record sampling or monetary unit sampling, projects errors found in sampled data to the entire population, and calculates upper limits on deviation rate, or misstatement amount.

Record samplingMonetary unit sampling

Syntax

EVALUATE RECORD CONFIDENCE *confidence_level* SIZE *sample_size* ERRORLIMIT *number_of_errors* <TO {SCREEN|*filename*}>

Parameters

> **Note**
>
> Do not include thousands separators, or percentage signs, when you specify values.

Name	Description
RECORD	Evaluate errors found in a record sample.
CONFIDENCE *confidence_level*	The same confidence level that you specified when you calculated the sample size.
SIZE *sample_size*	The number of records in the sample. **Note** Specify the actual sample size as drawn, which might differ from the sample size initially calculated by Analytics.
ERRORLIMIT *number_of_errors*	The total number of errors, or deviations, that you found in the sample.
TO SCREEN | *filename*	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Use either an absolute or relative file path to save the file to a different, existing folder: • TO "C:\Output.TXT" • TO "Results\Output.TXT"

Analytics output variables

Name	Contains
MLE*n*	The Upper error limit frequency rate (computed upper deviation rate) calculated by the command.

Examples

Project errors found in the sampled data to the entire population

You have completed your testing of the sampled data and recorded the control deviations you found. You can now project the errors you found to the entire population.

The example below projects two errors found in the sampled data to the entire population, and calculates **an upper error limit frequency rate** (computed upper deviation rate) of 6.63%.

```
EVALUATE RECORD CONFIDENCE 95 SIZE 95 ERRORLIMIT 2 TO SCREEN
```

For a detailed explanation of how Analytics calculates values when evaluating errors, see Evaluating errors in a record sample.

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Syntax

```
EVALUATE MONETARY CONFIDENCE confidence_level <ERRORLIMIT book_value, mis-
statement_amount <,...n>> INTERVAL interval_value <TO {SCREEN|filename}>
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Parameters

> **Note**
>
> Do not include thousands separators, or percentage signs, when you specify values.

Name	Description
MONETARY	Evaluate errors found in a monetary unit sample.
CONFIDENCE *confidence_level*	The same confidence level that you specified when you calculated the sample size.
ERRORLIMIT *book_value,misstatement_amount*	All misstatement errors that you found in the sample. Specify the book value of the amount and the misstatement amount, separated by a comma. For example, if an amount has a book value of $1,000 and an audit value of $930, specify `1000,70`. Specify overstatements as positive amounts, and understatements as negative amounts. For example, if an amount has a book value of $1,250 and an audit value of $1,450, specify `1250,-200`. Separate multiple *book_value, misstatement_amount* pairs with a comma: `1000,70,1250,-200`
INTERVAL *interval_value*	The interval value that you used when you drew the sample. **Note** The interval value that you used might differ from the interval value initially calculated by Analytics.
TO SCREEN | *filename*	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics output variables

Name	Contains
MLE*n*	The Most Likely Error amount (projected misstatement) calculated by the command.
UEL*n*	The Upper Error Limit amount (upper misstatement limit) calculated by the command.

Examples

Project errors found in the sampled data to the entire population

You have completed your testing of the sampled data and recorded the misstatements you found. You can now project the errors you found to the entire population.

The example below projects three errors found in the sampled data to the entire population, and calculates several values, including:

- **Basic Precision** – the basic allowance for sampling risk (18,850.00)
- **Most Likely Error** – the projected misstatement amount for the population (1,201.69)
- **Upper Error Limit** – the upper misstatement limit for the population (22,624.32)

```
EVALUATE MONETARY CONFIDENCE 95 ERRORLIMIT 1000,70,1250,-200,3200,900 INTERVAL
6283.33 TO SCREEN
```

For a detailed explanation of how Analytics calculates values when evaluating errors, see Evaluating errors in a monetary unit sample.

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EXECUTE command

Executes an application or process external to Analytics. Emulates the Windows Run command. Can be used to interact with the Windows command prompt.

> **Note**
>
> Because the EXECUTE command gives you the ability to interact with the operating system and applications external to Analytics, technical issues may arise that are beyond the scope of Analytics's native functionality.
>
> Support can assist with operation of the EXECUTE command inside Analytics, but issues that arise with processes and applications external to Analytics are not covered under Support.

Syntax

```
EXECUTE Windows_Run_command_syntax <ASYNC>
```

Parameters

Name	Description
Windows_Run_command_syntax	The name of the application to execute, the folder or file to open, or the command to run, followed by any required arguments or command switches. Requires valid Windows Run command syntax enclosed by quotation marks.
ASYNC optional	Runs the command in asynchronous mode. In asynchronous mode, the Analytics script continues running without waiting for the process started by the EXECUTE command to complete. If you omit ASYNC, then the process started by the EXECUTE command must complete before the Analytics script continues. Analytics is inaccessible while external processes are running. **Note** When running EXECUTE from the Analytics command line, you must specify ASYNC.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics output variables

Name	Contains
RETURN_CODE	The code returned by an external application or process run using the EXECUTE command. **What are return codes?** Return codes are numbers generated by the external application or process and sent back to Analytics to indicate the outcome of the external process. Analytics does not generate the return code, it only receives it. **Typical return codes** Typical return codes are integer values that map to specific notifications or error messages. For example, the return code "0" could mean "The operation completed successfully". The return code "2" could mean "The system cannot find the file specified". **The meaning of specific return codes** Specific return codes and their meanings vary depending on the external application or process. Lists of return codes, also called 'error codes' or 'exit codes', and their meanings, can often be found in the documentation for the associated external application. Lists of return codes can also be found on the Internet. **Variable created in default mode only** The RETURN_CODE variable is created when the EXECUTE command is run in default mode. The variable is not created when the command is run in asynchronous mode.

Examples

Open an application

Opens Microsoft Excel:

```
EXECUTE "Excel"
```

Opens Adobe Acrobat Reader:

```
EXECUTE "AcroRd32.exe"
```

Close an application

Closes Microsoft Excel:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
EXECUTE "TASKKILL /f /im Excel.exe"
```

> **Note**
>
> Use the /f switch with caution. It forces an application to close without presenting any dialog boxes, such as those for saving changes.

Open a file

Opens the Excel workbook `AP_Trans.xlsx`:

```
EXECUTE '"C:\ACL Projects\Source Data\AP_Trans.xlsx"'
```

Create a new folder

Creates a new folder named `Source Data`:

```
EXECUTE 'cmd /c MD "C:\ACL Projects\Source Data"'
```

Run external scripts or non-Analytics batch files (.bat)

Runs the script `My_Batch.bat`:

```
EXECUTE '"C:\ACL Projects\Batch Files\My_Batch.bat"'
```

Pass parameters to a non-Analytics batch file

Passes two parameters to `My_Batch.bat`. Parameters can be literals or Analytics variables:

```
EXECUTE '"C:\ACL Projects\Batch Files\My_Batch.bat" param1%v_param2%'
```

Run Analytics scripts in other Analytics projects

Runs "AP_Trans_script" in `AP Trans Tests.acl`"

```
EXECUTE 'aclwin.exe "C:\ACL Projects\AP Trans Tests.acl" /b AP_Trans_script'
```

> **Note**
>
> Running an Analytics script in another project launches a second instance of Analytics. The script in the second project should end with the QUIT command so that the second instance of Analytics closes and control is returned to the initial instance of Analytics.

Incorporate a waiting period in an Analytics script

Both examples create a waiting period of 30 seconds:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
EXECUTE "TIMEOUT /t 30"
```

```
EXECUTE "cmd /c PING -n 31 127.0.0.1 > nul"
```

Remarks

Use EXECUTE to perform useful tasks

The EXECUTE command allows you to run Windows and DOS commands from the Analytics command line or from an Analytics script.

You can use this ability to increase the automation of Analytics scripts by performing a variety of useful tasks that are not possible using ACLScript syntax alone.

Examples of tasks that can be started using EXECUTE

Open other programs and applications and perform tasks required by the Analytics script	Pass parameters to a batch file	Access data from network locations	Incorporate Active Directory account lists
Open any file in its default application	Run Analytics scripts in other Analytics projects	Use FTP to access data from remote locations	Integrate with VBScript
Perform file and folder administrative tasks such as copying, moving, creating, deleting, or comparing files or folders that exist outside of Analytics	Incorporate waiting periods in Analytics scripts	Zip or unzip data	Integrate with SQL databases
Run external scripts or non-Analytics batch files (.bat)	Incorporate Windows task scheduling in Analytics scripts	Encrypt or decrypt data	Open web pages

> **Note**
>
> Specific details of how to perform any of these tasks are beyond the scope of Galvanize Help documentation. For assistance, consult appropriate Windows operating system documentation, or other third-party documentation.

Default mode and asynchronous mode

You can run the EXECUTE command in either default mode or asynchronous mode:

- **Default mode** - the process started by EXECUTE must complete before the Analytics script can continue.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics is inaccessible while external processes are running.

- **Asynchronous mode** – the Analytics script continues to run without waiting for the process started by EXECUTE to complete.

  Analytics continues to be accessible while external processes are running.

  If you specify ASYNC, the EXECUTE command runs in asynchronous mode.

Which mode should I use?

When you create Analytics scripts that use the EXECUTE command you need to consider which mode of operation is appropriate.

Use default mode	Use asynchronous mode / ASYNC
◦ file and folder administrative tasks ◦ specifying waiting periods ◦ any task that subsequent tasks depend on ◦ subsequent script execution depends on the result in the RETURN_CODE variable	◦ external tasks cause an application interface or pop-up dialog box to open

Analytics scripts that run unattended

If you want an Analytics script that contains the EXECUTE command to run unattended, use one of the following methods:

- use asynchronous mode for any tasks that cause an application interface or pop-up dialog box to open
- avoid opening interface elements in unattended scripts

> **Note**
>
> Until interface elements are closed, they represent processes that are still running. If these interface elements are opened with EXECUTE in default mode, they prevent subsequent lines in an Analytics script from executing, and cause the script to hang.

EXECUTE command in analytic scripts

If you want to use the EXECUTE command in analytic scripts in Robots or on AX Server, you must specifically configure the command to run. For more information, see:

- **Robots** – Configuring a Robots Agent
- **AX Server** – AX Server settings

Quotation marks

The Windows Run command syntax that you use with the EXECUTE command must be enclosed by either single or double quotation marks.

The following example uses the Windows `MD` command to create a new folder:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
EXECUTE 'cmd /c md C:\New_Data_Folder'
```

Nested quotation marks

If any paths within the Run command syntax contain spaces, the paths must also be enclosed within quotation marks.

You have two options when enclosing paths within quotation marks:

- **Double quotation marks inside single quotation marks** – Use single quotation marks to enclose the entire Run command string, and use double quotation marks internally to enclose paths:

  ```
  EXECUTE 'cmd /c md "C:\New Data Folder"'
  ```

  You may find this method easier to read than the second method.

  > **Note**
  >
  > Reversing the order of the nesting – using double quotation marks to enclose the entire string, and single quotation marks to enclose paths – does not work.

- **Two double quotation marks** – Use double quotation marks to enclose the entire Run command string, and use two double quotation marks internally to enclose paths:

  ```
  EXECUTE "cmd /c md ""C:\New Data Folder"""
  ```

  If you use this second method, the two double quotation marks used internally must be immediately adjacent and cannot have a space between them.

Internal and external commands

Windows commands can be either **internal** or **external**.

- **Internal commands** – can be run from the command prompt only, which means that you have to open the command shell using `cmd /c` or `cmd /k` before specifying the command.
- **External commands** – can be run from either the command prompt or directly using the EXECUTE command, which means opening the command shell is an option, but not required.

The example below uses the internal Windows `DIR` command (displays the contents of a directory), and the external Windows `COMP` command (compares two files), to illustrate the difference:

```
EXECUTE 'cmd /k dir "C:\ACL DATA\Sample Data Files"'
EXECUTE 'comp C:\File_1.txt C:\File_2.txt'
```

You can avoid this complication by creating external scripts or batch files to contain Windows commands, and by using the EXECUTE command only to start the batch file. For example:

```
EXECUTE 'C:\My_Batch.bat'
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Multi-line Run command syntax

The EXECUTE command does not support multi-line Run command syntax. To incorporate multi-line Run commands in an Analytics script, use one of the following methods:

Method	Example
Repeat the EXECUTE command for each Run command.	EXECUTE 'cmd /c md "C:\New Data Folder"' EXECUTE 'cmd /c copy C:\File_1.txt "C:\New Data Folder"'
Combine Run commands using '&'.	EXECUTE 'cmd /c md "C:\New Data Folder" & copy C:\File_1.txt "C:\New Data Folder"'
Create an external script or batch file to contain multi-line Run commands, and use the EXECUTE command only to start the batch file.	EXECUTE 'C:\My_Batch.bat'

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EXPORT command

Exports data from Analytics to the specified file format, or to HighBond Results.

Syntax

EXPORT {<FIELDS> *field_name* <AS *export_name*> <...*n*>|<FIELDS> ALL} <UNICODE> *export_type* <SCHEMA> PASSWORD *num* TO {*filename*|*aclgrc_id*} <OVERWRITE> <IF *test*> <WHILE *test*> <{FIRST *range*|NEXT *range*}> <APPEND> <KEEPTITLE> <SEPARATOR *character*> <QUALIFIER *character*> <WORKSHEET *worksheet_name*> <DISPLAYNAME>

Parameters

Name	Description
FIELDS *field_name* AS *export_name* <...*n*> | FIELDS ALL	The fields to export. ◦ ***field_name*** - export the specified field or fields Separate field names with spaces. You can optionally include a different name for the field in the export file using AS *export_name*. Enclose *export_name* in quotation marks. If you are exporting to HighBond Results (ACLGRC), it is possible to combine AS with the DISPLAYNAME parameter. For more information, see "How DISPLAYNAME interacts with AS when exporting to HighBond Results" on page 202. ◦ **ALL** - export all fields in the table
UNICODE optional	Available in the Unicode edition of Analytics only. Applies to text (ASCII), delimited text (DELIMITED), and XML files only, and to Windows Clipboard (CLIPBOARD) output. Exports Analytics data with Unicode UTF-16 LE character encoding applied. ◦ **Specify UNICODE** - if the data you are exporting contains characters that are not supported by extended ASCII (ANSI) ◦ **Do not specify UNICODE** - if all the characters in the data you are exporting are supported by extended ASCII (ANSI) The exported data is encoded as extended ASCII (ANSI). **Note** Any unsupported characters are omitted from the exported file. For more information, see Galvanize Unicode products.
export_type	The output file format or destination using one of the following options: ◦ **ACCESS** - Microsoft Access database file (.mdb) By default, the data is exported as Unicode.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ **ACLGRC** - HighBond Results ◦ **ASCII** - ASCII plain text (.txt) ◦ **CLIPBOARD** - Windows Clipboard ◦ **DBASE** - dBASE compatible file (.dbf) ◦ **DELIMITED** - delimited text file (.del), or comma-separated values file (.csv) ◦ **EXCEL** - Microsoft Excel file (.xls) compatible with Excel 1997 to 2003 ◦ **JSON** - JSON file (.json) ◦ **LOTUS** - Lotus 123 file ◦ **WDPF6** - Wordperfect 6 file ◦ **WORD** - MS Word file (.doc) ◦ **WP** - Wordperfect file ◦ **XLS21** - Microsoft Excel version 2.1 file ◦ **XLSX** - Microsoft Excel .xlsx file By default, the data is exported as Unicode. ◦ **XML** - XML file (.xml)
SCHEMA optional	Applies to XML file output only. Include the XML schema in the exported XML file. The XML schema contains metadata that describes the structure of the XML file, including the data type of the fields. You can validate the file against the schema once the file has been exported.
PASSWORD *num*	Applies to HighBond Results (ACLGRC) only. The password definition to use. You do not use PASSWORD *num* to prompt for, or specify, an actual password. The password definition refers to a password previously supplied or set using the PASSWORD command, the SET PASSWORD command, or the PASSWORD analytic tag. *num* is the number of the password definition. For example, if two passwords have been previously supplied or set in a script, or when scheduling an analytic, PASSWORD 2 specifies that password #2 is used. For more information about supplying or setting passwords, see: • "PASSWORD command" on page 360 • "SET command" on page 418 • PASSWORD analytic tag PASSWORD *num* must be placed immediately before TO, or at the end of the string of command syntax. The required password value is a HighBond access token. For more information, see "Exporting to HighBond Results" on page 200.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>
Note
PASSWORD may or may not be required, depending on the environment in which the script runs:
<table>
<tr><td>Analytics
(online activation)</td><td>PASSWORD is not required.
The current user's HighBond access token is automatically used.</td></tr>
<tr><td>Analytics
(offline activation)</td><td rowspan="4">PASSWORD is required.</td></tr>
<tr><td>Robots</td></tr>
<tr><td>Analytics Exchange</td></tr>
<tr><td>Analysis App window</td></tr>
</table>
</td></tr>
<tr><td>TO filename | aclgrc_id</td><td>
The destination for the export:

TO filename - export data to a file
If required, you can include either an absolute or relative file path, but the Windows folder must already exist. You must specify the filename value as a quoted string.
Note
To export to a comma-separated values file (<code>*.csv</code>), you must specify the .csv file extension as part of filename. For example: <code>vendors.csv</code>
TO aclgrc_id - export data to HighBond Results
The aclgrc_id value must include the control test ID number, and if you are exporting to a data center other than North America, the data center code. The aclgrc_id value must be enclosed in quotation marks.
The control test ID number and the data center code must be separated by the at sign (@). For example, <code>TO "99@eu"</code>.
If you do not know the control test ID number, use the Analytics user interface to begin an export to Results. Cancel the export once you have identified the control test ID number. For more information, see Exporting exceptions to HighBond Results.
The data center code specifies which regional HighBond server you are exporting the data to:

<code>ap</code> - Asia Pacific
<code>au</code> - Australia
<code>ca</code> - Canada
<code>eu</code> - Europe
<code>us</code> - North America

You can use only the data center code or codes authorized for your organization's instance of HighBond. The North America data center is the default, so specifying <code>"@us"</code> is optional.

</td></tr>
<tr><td>OVERWRITE</td><td>Applies to HighBond Results (<code>ACLGRC</code>) only.</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
optional	◦ **OVERWRITE specified** - Exported data overwrites any existing data in the target control test (table). You must have a Professional Manager role in the target Collection to overwrite data. ◦ **OVERWRITE omitted** - Exported data is appended to any existing data in the target control test (table). For more information, see "Exporting to HighBond Results" on page 200. Any interpretations related to the target control test (table) dynamically update to reflect the imported data, whether you overwrite or append.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
APPEND optional	Applies to text (ASCII) and delimited text (DELIMITED) files only. Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
KEEPTITLE optional	Applies only to text files (ASCII), and delimited text and comma-separated values files (DELIMITED). Include the Analytics field names with the exported data. If omitted, no field names

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	appear in the output file.
SEPARATOR *character* optional	Applies only to delimited text and comma-separated values files (DELIMITED). The character to use as the separator between fields. You must specify the character as a quoted string. By default, Analytics uses a comma. Do not specify any character other than a comma if you are exporting to a comma-separated values file.
QUALIFIER *character* optional	Applies only to delimited text and comma-separated values files (DELIMITED). The character to use as the text qualifier to wrap and identify field values. You must specify the character as a quoted string. By default, Analytics uses double quotation marks.
WORKSHEET *worksheet_name* optional	Applies to Microsoft Excel (.xlsx) files only. The name of the Excel worksheet created in a new or existing Excel file. By default, Analytics uses the name of the Analytics table you are exporting as the worksheet name. The *worksheet_name* can contain only alphanumeric characters or the underscore character (_). The name cannot contain special characters, spaces, or start with a number. Enclosing the value in quotation marks is optional. For details about overwriting Excel workbooks and worksheets when exporting, see "The WORKSHEET parameter and overwriting" on page 199.
DISPLAYNAME optional	Applies to HighBond Results (`ACLGRC`) only. Exports field names as field names and display names as display names so the display names appear in column headings in Results without affecting the actual field name. It is possible to combine DISPLAYNAME with AS. For more information see "How DISPLAYNAME interacts with AS when exporting to HighBond Results" on page 202.

Examples

Export data to an Excel .xlsx file

You export specific fields from the **Vendor** table to an Excel .xlsx file:

```
OPEN Vendor
EXPORT FIELDS Vendor_No Vendor_Name Vendor_City XLSX TO "VendorExport"
```

Export data to an Excel .xlsx file and specify a worksheet name

You export specific fields from the **Vendor** table to a worksheet called **Vendors_US** in an Excel .xlsx file:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
OPEN Vendor
EXPORT FIELDS Vendor_No Vendor_Name Vendor_City XLSX TO "VendorExport"
WORKSHEET Vendors_US
```

Export all fields to a delimited file

You export all fields from the **Vendor** table to a delimited file:

```
OPEN Vendor
EXPORT FIELDS ALL DELIMITED TO "VendorExport"
```

Export all fields to a comma-separated values file

You export all fields from the **Vendor** table to a comma-separated values file:

```
OPEN Vendor
EXPORT FIELDS ALL DELIMITED TO "VendorExport.csv"
```

Export data to multiple delimited files using GROUP

You export specific fields from the **Vendor** table to two delimited files:

- one file for vendor names from "A" to "M"
- one file for vendor names from "N" to "Z"

Using the GROUP command, you test the vendor name of each record with an IF condition:

```
GROUP
 EXPORT FIELDS Vendor_No Vendor_Name DELIMITED TO "AtoM" IF BETWEEN(UPPER
(VENDOR_NAME), "A", "M")
 EXPORT FIELDS Vendor_No Vendor_Name DELIMITED TO "NtoZ" IF BETWEEN(UPPER
(VENDOR_NAME), "N", "Z")
END
```

Export data to HighBond Results

You export specific fields from the **AR_Exceptions** table to HighBond Results. You overwrite existing data in the target control test (table):

```
OPEN AR_Exceptions
EXPORT FIELDS No Due Date Ref Amount Type ACLGRC PASSWORD 1 TO "10926@us"
OVERWRITE
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Using EXPORT with the GROUP command

For most export formats, you can export data into multiple files simultaneously using the GROUP command.

Only one file can be created at a time when you are exporting data to Microsoft Excel and Microsoft Access.

Exporting to Excel

The following limits apply when exporting data to an Excel file:

Number of records	◦ **Excel 2007 and later (*.xlsx)** - a maximum of 1,048,576 records ◦ **Excel 97 and 2003** - a maximum of 65,536 records Analytics tables that exceed these maximums export successfully, but the excess records are ignored and not exported.
Length of fields	◦ no specific field length limit ◦ combined field lengths cannot exceed the overall record length limit of 32 KB (32,765 characters in non-Unicode Analytics, 16,382 characters in Unicode Analytics) ◦ for Excel 2.1, a maximum of 247 characters
Length of field names	◦ a maximum of 64 characters ◦ for Excel 2.1, a maximum of 248 characters

The WORKSHEET parameter and overwriting

The result of using or not using the WORKSHEET parameter when exporting from an Analytics table to an Excel file is explained below:

Matching	Description	WORKSHEET parameter used	WORKSHEET parameter not used
No matching Excel file name	◦ TO *filename* value does not match any existing Excel file name	A new Excel file is created, with a worksheet with the specified name	A new Excel file is created, with a worksheet that uses the name of the exported Analytics table
Matching Excel file name No matching worksheet name	◦ TO *filename* value, and an existing Excel file name, are identical ◦ WORKSHEET *worksheet_name* does not match a worksheet	A worksheet with the specified name is added to the existing Excel file	The existing Excel file is overwritten by a new Excel file, with a worksheet that uses the name of the exported Analytics table

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Matching	Description	WORKSHEET parameter used	WORKSHEET parameter not used
	name in the Excel file		
Matching Excel file name and worksheet name	◦ TO *filename* value, and an existing Excel file name, are identical ◦ WORKSHEET *worksheet_name* matches a worksheet name in the Excel file	A worksheet with the specified name overwrites the existing worksheet if it was originally created from Analytics. An error message appears and the export operation is canceled if the existing worksheet was originally created directly in Excel.	The existing Excel file is overwritten by a new Excel file, with a worksheet that uses the name of the exported Analytics table

Exporting to HighBond Results

The table below contains additional information about exporting to a control test in Results.

Item	Details
Required permissions	The ability to export results to a control test in Results requires a specific HighBond role assignment, or administrative privileges: ◦ Users with a Professional User or Professional Manager role for a Results collection can export results to any control test in the collection. > **Note** > Only users with the Professional Manager role can export and overwrite existing data in a control test. ◦ HighBond System Admins and Results admins automatically get a Professional Manager role in all collections in the HighBond organization or organizations they administer.
Export limits	The following limits apply when exporting to a control test: ◦ A maximum of 100,000 records per export ◦ A maximum of 100,000 records per control test ◦ A maximum of 500 fields per record ◦ A maximum of 256 characters per field You can export multiple times to the same control test, but you cannot exceed the overall limits.
Appending fields (OVERWRITE not specified)	Regardless of their order in an Analytics table, exported fields are appended to existing fields in a control test if they have matching physical field names. In Analytics, the physical field name is the name in the table layout. Exported fields that do not match the name of any existing field are added as additional columns to the table in Results. Display names of fields in Analytics, and in Results, are not considered. However, if you use the optional AS *export_name* parameter, the *export_name* value is used as the physical field name if you do not use DISPLAYNAME .

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Item	Details
	When appending data to questionnaire fields, the display name of the column in Results remains the name that is specified in the questionnaire configuration. Appending works differently if the target control test has a primary key field specified. For more information, see Exporting exceptions to HighBond Results. **Note** If you are round-tripping data between Results and Analytics, and data ends up misaligned in Results, you probably have mismatched field names. For more information, see "Field name considerations when importing and exporting Results data" on page 268.
Creating a password definition and specifying a password value	PASSWORD command If you use the PASSWORD command to create the numbered password definition for connecting to HighBond, no password value is specified, so a password prompt is displayed when the script attempts to connect. For more information, see "PASSWORD command" on page 360. SET PASSWORD command If you use the SET PASSWORD command to create the numbered password definition for connecting to HighBond, a password value is specified, so no password prompt is displayed, which is appropriate for scripts designed to run unattended. For more information, see SET PASSWORD command. Acquire a HighBond access token Regardless of which method you use to create the password definition, the required password value is a HighBond access token, which users can generate in Launchpad. **Caution** The generated access token matches the account used to sign in to Launchpad. As a scriptwriter, specifying your own access token in a script may not be appropriate if the script will be used by other people. a. Do one of the following: • From the Analytics main menu, select **Tools > HighBond Access Token**. • In the **Script Editor**, right-click and select **Insert > HighBond Token**. The **Manage API tokens** page opens in your browser. You may be required to first sign in to Launchpad. b. Do one of the following: • **Use an existing token** - In the **Token** column, click the partially masked token that you want to use and enter your HighBond account password. The unmasked token is displayed. **Tip** Use an existing token unless you have a reason for creating a new one. If the existing token does not work, create a new one. Using an existing token cuts down on the number of tokens you need to manage. • **Create a new token** - Click **Create token > Analytics** and enter your HighBond account password.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Item	Details
	A new Analytics token is created. **Note** If you are a Launchpad System Admin, you also have the option of creating an API token. You should reserve API tokens for their intended purpose, which is programmatic access to the HighBond platform. c. Click **Copy** to copy the token. **Tip** Do not close the dialog box containing the token until you have successfully pasted the token. d. In Analytics, do one of the following: • paste the token into the password prompt • paste the token at the appropriate point in the SET PASSWORD command syntax in a script e. In Launchpad, close the dialog box containing the token. If you created a new token, a partially masked version of the token is added to the top of your list of tokens. For more information, see Creating and managing access tokens.

How DISPLAYNAME interacts with AS when exporting to HighBond Results

The matrix below shows how the DISPLAYNAME parameter interacts with AS when exporting field names from Analytics to Results.

	Without AS	With AS
Without DISPLAYNAME	Field name and display name in Results are the field name from Analytics.	Field name and display name in Results are the display name in the AS parameter.
With DISPLAYNAME	Field name in Results is the field name from Analytics. Display name in Results is the display name from Analytics.	Field name in Results is the field name from Analytics. Display name in Results is the display name in the AS parameter.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EXTRACT command

Extracts data from an Analytics table and outputs it to a new Analytics table, or appends it to an existing Analytics table. You can extract entire records or selected fields.

Syntax

EXTRACT {RECORD|FIELDS *field_name* <AS *display_name*> <...*n*>|FIELDS ALL} TO *table_name* <LOCAL> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <EOF> <APPEND> <OPEN>

Parameters

Name	Description
RECORD | FIELDS *field_name* | FIELDS ALL	The fields to include in the output: ◦ **RECORD** - use the entire record in the source data file: all fields in the table, and any undefined portions of the record Fields are used in the order that they appear in the table layout. Preserves computed fields. ◦ **FIELDS *field_name*** - use the specified fields Fields are used in the order that you list them. Converts computed fields to physical fields of the appropriate data type in the destination table - ASCII or Unicode (depending on the edition of Analytics), ACL (the native numeric data type), Datetime, or Logical. Populates the physical fields with the actual computed values. ◦ **FIELDS ALL** - use all fields in the table Fields are used in the order that they appear in the table layout. Converts computed fields to physical fields of the appropriate data type in the destination table - ASCII or Unicode (depending on the edition of Analytics), ACL (the native numeric data type), Datetime, or Logical. Populates the physical fields with the actual computed values.
AS *display_name* optional	Only used when extracting using FIELDS *field_name*. The display name (alternate column title) for the field in the view in the new Analytics table. If you want the display name to be the same as the field name, or an existing display name in the source table, do not use AS. Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	> Note AS works only when extracting to a new table. If you are appending to an existing table, the alternate column titles in the existing table take precedence.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` > **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
LOCAL optional	Saves the output file in the same location as the Analytics project. > **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. > **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. > **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
EOF optional	Execute the command one more time after the end of the file has been reached. This ensures that the final record in the table is processed when inside a GROUP command. Only use EOF if all fields are computed fields referring to earlier records.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.

Examples

Extracting all records in a table to a new table

You create an exact duplicate of the **AR_Customer** table by extracting all the records to a new Analytics table. Any computed fields are preserved as computed fields:

```
OPEN AR_Customer
EXTRACT RECORD TO "AR_Customer_2"
```

Extracting all fields in a table to a new table

You extract all defined fields in the **AR_Customer** table to a new Analytics table. Any computed fields are converted to physical fields and populated with the actual computed values:

```
OPEN AR_Customer
EXTRACT FIELDS ALL TO "AR_Customer_2"
```

Extracting all records in a table and appending them to an existing table

You extract all the records in the **AR_Customer** table and append them as a group to the end of the **AR_**

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Customer_Master table:

```
OPEN AR_Customer
EXTRACT RECORD TO "AR_Customer_Master" APPEND
```

Extracting all records in a table and appending them to an existing table in a different folder

You extract all the records in the **AR_Customer** table and append them as a group to the end of the **AR_Customer_Master** table, which is in a folder other than the Analytics project folder:

```
OPEN AR_Customer
EXTRACT RECORD TO "C:\Users\Customer Data\AR_Customer_Master" APPEND
```

Extracting a subset of fields from a table to a new table

You extract three fields from the **AR_Customer** table to a new Analytics table:

```
OPEN AR_Customer
EXTRACT FIELDS Name Due Date TO "AR_Customer_Dates.fil"
```

Creating display names for extracted fields

You extract three fields from the **AR_Customer** table and create display names for the fields in the new Analytics table:

```
OPEN AR_Customer
EXTRACT FIELDS Name AS "Customer;Name" Due AS "Due;Date" Date AS "Invoice;Date" TO
"AR_Customer_Dates.fil"
```

Extracting fields based on a condition

You extract three fields from the **AR_Customer** table to a new Analytics table if the date in the **Due** field is before July 1, 2014:

```
OPEN AR_Customer
EXTRACT FIELDS Name Due Date IF Due < `20140701` TO "Overdue.fil"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EXTRACT vs Copying a table

EXTRACT creates a new source data file (.fil) as well as a new table layout.

Copying a table using the **Navigator** (**Edit > Copy**) creates a new table layout that remains associated with the original source data file. It does not create a new data file.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FIELDSHIFT command

Shifts the start position of a field definition in a table layout.

Syntax

FIELDSHIFT START *starting_position* COLUMNS *bytes_to_shift* <FILTER *data_filter_name*> <OK>

Parameters

Name	Description
START *starting_position*	The starting byte position of the first field definition you want to shift. All field definitions to the right of the specified field definition are also shifted. If you specify a non-starting byte position, the next starting byte position is used. **Note** non-Unicode Analytics: 1 byte = 1 character Unicode Analytics, extended ASCII (ANSI) data: 1 byte = 1 character Unicode Analytics, Unicode data: 2 bytes = 1 character For Unicode data, typically you should specify an odd-numbered starting byte position. Specifying an even-numbered starting position can cause characters to display incorrectly.
COLUMNS *bytes_to_shift*	The number of bytes to shift the field definition. Enter a positive number to shift a field definition to the right. Enter a negative number to shift a field definition to the left. **Note** non-Unicode Analytics: 1 byte = 1 character Unicode Analytics, extended ASCII (ANSI) data: 1 byte = 1 character Unicode Analytics, Unicode data: 2 bytes = 1 character For Unicode data, specify an even number of bytes only. Specifying an odd number of bytes can cause characters to display incorrectly.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
FILTER *data_filter_name* optional	The name of the filter that identifies field definitions associated with a particular record definition.
OK optional	Deletes or overwrites items without asking you to confirm the action.

Examples

Shifting field definitions

You shift the field definition starting at byte 11, and any subsequent field definitions, 4 bytes to the right:

```
FIELDSHIFT START 11 COLUMNS 4
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Shifted field definitions must remain within the record length

When you shift one or more field definitions right or left, the fields cannot exceed the record length in either direction.

Keep in mind that FIELDSHIFT moves both the specified field definition, and any field definitions to the right of the specified definition. If the shifted block of definitions would exceed the record length in either direction, an error message appears and the command is not executed.

> **Tip**
>
> If the error message is appearing because you are exceeding the end of the record, try removing the final field definition to make room for the field definitions being shifted.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FIND command

Searches an indexed character field for the first value that matches the specified character string.

> **Note**
>
> The FIND command and the FIND( ) function are two separate Analytics features with significant differences. For information about the function, see "FIND( ) function" on page 574.

Syntax

```
FIND search_value
```

Parameters

Name	Description
search_value	The character string to search for. *search_value* is case-sensitive and cannot include leading spaces. Do not enclose the value in quotation marks unless quotation marks are part of the data being searched.

Examples

Searching for a specific value

You want to locate the first value in the **Card_Number** character field that exactly matches or starts with "8590124".

First you index the **Card_Number** field in ascending order. Then you run FIND:

```
INDEX ON Card_Number TO "CardNum" OPEN
SET INDEX TO "CardNum"
FIND 8590124
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

When to use FIND

Use the FIND command to move directly to the first record in a table containing the specified *search_value* in the indexed character field.

INDEX requirement

To use the command, the table you are searching must be indexed on a character field in ascending order.

If multiple character fields are indexed in ascending order, only the first field specified in the index is searched. The command cannot be used to search non-character index fields, or character fields indexed in descending order.

Partial matching

Partial matching is supported. The search value can be contained by a longer value in the indexed field. However, the search value must appear at the start of the field to constitute a match.

FIND output depending on match

The FIND command produces one of the following results, depending on whether the search value is found:

- **search value is found** – the first matching record in the table is selected
- **search value is not found** – the table is positioned at the first record with a greater value than the search value

  If there are no values in the indexed field greater than the search value, the table is positioned at the first record. In both cases, the message "No index matched key" is displayed.

The FIND command is not affected by the **Exact Character Comparisons** option (SET EXACT ON/OFF).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FUZZYDUP command

Detects nearly identical values (fuzzy duplicates) in a character field.

> **Note**
>
> To use fuzzy matching to combine fields from two Analytics tables into a new, single Analytics table, see "FUZZYJOIN command" on page 217.

Syntax

FUZZYDUP ON *key_field* <OTHER *fields*> LEVDISTANCE *value* <DIFFPCT *percentage*> <RESULTSIZE *percentage*> <EXACT> <IF *test*> TO *table_name* <LOCAL> <OPEN>

Parameters

Name	Description
ON *key_field*	The character field or expression to test for fuzzy duplicates.
OTHER *fields* optional	A list of fields or expressions to include in the output.
LEVDISTANCE *value*	The maximum allowable Levenshtein distance between two strings for them to be identified as fuzzy duplicates and included in the results. The LEVDISTANCE value cannot be less than 1 or greater than 10. Increasing the LEVDISTANCE value increases the number of results by including values with a greater degree of fuzziness - that is, values that are more different from one another. For more information, see "FUZZYDUP behavior" on page 215.
DIFFPCT *percentage* optional	A threshold that limits the 'difference percentage' or the proportion of a string that can be different. The percentage that results from an internal Analytics calculation performed on potential fuzzy duplicate pairs must be less than or equal to the DIFFPCT value for the pair to be included in the results. The DIFFPCT value cannot be less than 1 or greater than 99. If DIFFPCT is omitted the threshold is turned off and difference percentage is not considered during processing of the FUZZYDUP command. For more information, see "FUZZYDUP behavior" on page 215.
RESULTSIZE *percentage* optional	The maximum size of the set of output results as a percentage of the number of records in the key field. For example, for a key field with 50,000 records, a RESULTSIZE of 3 would terminate processing if the results exceeded 1500 fuzzy duplicates (50,000 x 0.03). No output

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	table is produced if processing is terminated. The RESULTSIZE value cannot be less than 1 or greater than 1000 (one thousand) percent. The limit of 1000% is to accommodate the nature of many-to-many matching, which can produce results that are more numerous than the original test data set. If RESULTSIZE is omitted the threshold is turned off and result size is not considered during processing of the FUZZYDUP command. **Caution** Omitting RESULTSIZE can produce an unduly large set of results that takes a very long time to process, or can cause available memory to be exceeded, which terminates processing. Omit RESULTSIZE only if you are confident that the results will be of a manageable size.
EXACT optional	Includes exact duplicates as well as fuzzy duplicates in the results.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.

Analytics output variables

Name	Contains
GAPDUP*n*	The total number of gaps, duplicates, or fuzzy duplicate groups identified by the command.

Examples

Test a surname field for fuzzy duplicates

You test a surname field for fuzzy duplicates (the **Last_Name** field in the **Employee_List** table in `ACL DATA\Sample Data Files\Metaphor_Employee_Data.ACL`). The results are output to a new Analytics table.

- In addition to the test field, other fields are included in the results.
- The maximum allowable Levenshtein distance is 1.
- The proportion of a string that can be different is limited to 50%.
- The size of the results is limited to 20% of the test field size.
- In addition to fuzzy duplicates, exact duplicates are included.

```
FUZZYDUP ON Last_Name OTHER First_Name EmpNo LEVDISTANCE 1 DIFFPCT 50
RESULTSIZE 20 EXACT TO "Fuzzy_Last_Name" OPEN
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

How it works

The FUZZYDUP command finds nearly identical values (fuzzy duplicates), or locates inconsistent spelling in manually entered data.

Unlike the ISFUZZYDUP( ) function, which identifies an exhaustive list of fuzzy duplicates for a single character value, the FUZZYDUP command identifies all fuzzy duplicates in a field, organizes them in groups, and outputs non-exhaustive results.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

What non-exhaustive means

Non-exhaustive means that individual fuzzy duplicate groups in the results may not contain all the fuzzy duplicates in a test field that are within the specified degree of difference of the group owner. However, if a group owner is a fuzzy duplicate of another value in the test field, the two values will appear together in a group somewhere in the results.

If producing an exhaustive list of fuzzy duplicates for a specific value in the test field is important to your analysis, you can use the ISFUZZYDUP( ) function for this purpose.

FUZZYDUP behavior

The FUZZYDUP command has two parameters that allow you to control the degree of difference between fuzzy duplicates, and the size of the results:

- LEVDISTANCE
- DIFFPCT

You may need to try different combinations of settings for these two parameters to find out what works best for a particular data set.

LEVDISTANCE (Levenshtein distance)

When processing data, the FUZZYDUP command calculates the Levenshtein distance between each evaluated pair of strings in the test field, and calculates the difference percentage. The Levenshtein distance is a value representing the minimum number of single character edits required to make one string identical to the other string. For more information, see "LEVDIST( ) function" on page 633.

DIFFPCT (Difference percentage)

The difference percentage is the percentage of the shorter of the two evaluated strings that is different, and is the result of the following internal Analytics calculation, which uses the Levenshtein distance between the two strings:

Levenshtein distance / number of characters in the shorter string × 100 = difference percentage

More information

For detailed information about the fuzzy duplicate difference settings, controlling result size, and fuzzy duplicate groups, see Fuzzy duplicates overview.

Case-sensitivity

The FUZZYDUP command is not case-sensitive, so "SMITH" is equivalent to "smith."

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Trailing blanks automatically trimmed

The FUZZYDUP command automatically trims trailing blanks in *key_field*, so there is no need to use the TRIM( ) or ALLTRIM( ) function when specifying a single field for *key_field*.

If you concatenate fields for *key_field*, you should use ALLTRIM( ), as shown below.

Improving the effectiveness of FUZZYDUP

Concatenating fields

Concatenating two or more test fields can improve the effectiveness of the FUZZYDUP command by increase the degree of uniqueness of the test values.

For example:

```
FUZZYDUP ON ALLTRIM(First_Name)+ALLTRIM(Last_Name) OTHER First_Name Last_Name
EmpNo LEVDISTANCE 4 DIFFPCT 50 RESULTSIZE 20 EXACT TO "Fuzzy_First_Name_Last_
Name" OPEN
```

The OMIT( ) function

The OMIT( ) function can also improve the effectiveness of the command by removing generic elements such as "Corporation" or "Inc." from field values.

Removal of generic elements focuses the FUZZYDUP string comparison on just the portion of the strings where a meaningful difference may occur.

For more information, see "OMIT( ) function" on page 683.

Other string comparison methods

- **DICECOEFFICIENT( ) function** – provides a method for comparing strings that de-emphasizes or completely ignores the relative position of characters or character blocks.
- **SOUNDSLIKE( ) and SOUNDEX( ) functions** – provide a method for comparing strings based on a phonetic comparison (sound) rather than on an orthographic comparison (spelling).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FUZZYJOIN command

Uses fuzzy matching to combine fields from two Analytics tables into a new, single Analytics table.

> **Note**
>
> To detect nearly identical values (fuzzy duplicates) in a single character field, see "FUZZYDUP command" on page 212.
>
> For various options when joining tables using exactly matching key field values, see "JOIN command" on page 326.

Syntax

FUZZYJOIN {DICE PERCENT *percentage* NGRAM *n-gram_length*|LEVDISTANCE DISTANCE *value*} PKEY *primary_key_field* SKEY *secondary_key_field* {FIELDS *primary_fields*|FIELDS ALL} <WITH *secondary_fields*|WITH ALL> <IF *test*> <OPEN> <FIRSTMATCH> TO *table_name* <WHILE *test*> <FIRST *range*|NEXT *range*> <APPEND>

> **Note**
>
> You cannot run the FUZZYJOIN command locally against a server table.
>
> You must specify the FUZZYJOIN command name in full. You cannot abbreviate it.

Parameters

Name	Description
DICE PERCENT *percentage* NGRAM *n-gram_length* | LEVDISTANCE DISTANCE *value*	The fuzzy matching algorithm to use. **DICE** - use the Dice coefficient algorithm ◦ **PERCENT *percentage*** - the minimum allowable Dice's coefficient of two strings for them to qualify as a fuzzy match Specify a decimal fraction, from 0.0000 to 1.0000 (for example, 0.7500). Use a maximum of four decimal places. Decreasing the value increases the number of matches by including matches with a greater degree of fuzziness - that is, strings that are more different from each another. ◦ **NGRAM *n-gram_length*** - the *n*-gram length to use Specify a whole number, 1 or greater. Increasing the *n*-gram length makes the criterion for similarity between two strings stricter. *N*-grams are overlapping substrings (character blocks) into which the comparison strings are divided as part of the Dice's coefficient calculation.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note When you specify DICE, the FUZZYJOIN command uses the DICECOEFFICIENT( ) function in an IF statement to conditionally join key field values. For detailed information about the function, see "DICECOEFFICIENT( ) function" on page 547. **LEVDISTANCE** - use the Levenshtein distance algorithm ◦ **DISTANCE *value*** - the maximum allowable Levenshtein distance between two strings for them to qualify as a fuzzy match Specify a whole number, 1 or greater. Increasing the value increases the number of matches by including matches with a greater degree of fuzziness - that is, strings that are more different from each another. **Note** When you specify LEVDISTANCE, the FUZZYJOIN command uses the LEVDIST( ) function in an IF statement to conditionally join key field values. For detailed information about the function, see "LEVDIST( ) function" on page 633. Unlike the function, the Levenshtein distance algorithm in the FUZZYJOIN command automatically trims leading and trailing blanks, and is not case-sensitive.
PKEY *primary_key_field*	The character key field, or expression, in the primary table. You can specify only one primary key field.
SKEY *secondary_key_field*	The character key field, or expression, in the secondary table. You can specify only one secondary key field.
FIELDS *primary_fields* | FIELDS ALL	The fields or expressions from the primary table to include in the joined output table. ◦ ***primary_fields*** - include the specified field or fields ◦ **ALL** - include all fields from the table **Note** You must explicitly specify the primary key field if you want to include it in the joined table. Specifying ALL also includes it.
WITH *secondary_fields* | WITH ALL optional	The fields or expressions from the secondary table to include in the joined output table. ◦ ***secondary_fields*** - include the specified field or fields ◦ **ALL** - include all fields from the table **Note** You must explicitly specify the secondary key field if you want to include it in the joined table. Specifying ALL also includes it.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT). **Note** The IF condition can reference the primary table, the secondary table, or both.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.
FIRSTMATCH optional	Specifies that each primary key value is joined to only the first occurrence of any secondary key matches. If the first occurrence happens to be an exact match, any subsequent fuzzy matches for the primary key value are not included in the joined output table. If you omit FIRSTMATCH, the default behavior of FUZZYJOIN is to join each primary key value to all occurrences of any secondary key matches. FIRSTMATCH is useful if you only want to know if any matches, exact or fuzzy, exist between two tables, and you want to reduce the processing time required to identify all matches. You can also use FIRSTMATCH if you are certain that at most only one match exists in the secondary table for each primary key value. **Note** FIRSTMATCH is available only as an ACLScript parameter. The option is not available in the Analytics user interface.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
ISOLOCALE *locale_code* optional	**Note** Applicable in the Unicode edition of Analytics only. The system locale in the format *language_country*. For example, to use Canadian French, enter fr_ca. Use the following codes: ◦ **language** - ISO 639 standard language code ◦ **country** - ISO 3166 standard country code If you do not specify a country code, the default country for the language is used. If you do not use ISOLOCALE, the default system locale is used.

Examples

Use fuzzy matching to join two tables as a way of discovering employees who may also be vendors

The example below joins the Empmast and Vendor tables using address as the common key field (the Address and Vendor_Street fields).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

The FUZZYJOIN command creates a new table with either exactly matched or fuzzy matched primary and secondary records. The result is a list of any employees and vendors with either an identical address, or a similar address.

FUZZYJOIN with the Dice coefficient algorithm

```
OPEN Empmast PRIMARY
OPEN Vendor SECONDARY
FUZZYJOIN DICE PERCENT 0.8000 NGRAM 2 PKEY Address SKEY Vendor_Street FIELDS
Employee_Number First_Name Last_Name Address WITH Vendor_Number Vendor_Name Vendor_
Street OPEN TO "Employee_Vendor_Match"
```

FUZZYJOIN with the Levenshtein distance algorithm

```
OPEN Empmast PRIMARY
OPEN Vendor SECONDARY
FUZZYJOIN LEVDISTANCE DISTANCE 5 PKEY Address SKEY Vendor_Street FIELDS Employee_
Number First_Name Last_Name Address WITH Vendor_Number Vendor_Name Vendor_Street
OPEN TO "Employee_Vendor_Match"
```

Include all fields

This version of the FUZZYJOIN command includes all fields from the primary and secondary tables in the joined output table.

```
OPEN Empmast PRIMARY
OPEN Vendor SECONDARY
FUZZYJOIN LEVDISTANCE DISTANCE 5 PKEY Address SKEY Vendor_Street FIELDS ALL WITH
ALL OPEN TO "Employee_Vendor_Match"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Case sensitivity

The FUZZYJOIN command is not case-sensitive, regardless of which fuzzy matching algorithm you use. So "SMITH" is equivalent to "smith."

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Leading and trailing blanks

The FUZZYJOIN command automatically trims leading and trailing blanks in fields, regardless of which fuzzy matching algorithm you use. There is no need to use the TRIM( ) or ALLTRIM( ) functions when specifying the primary and secondary key fields.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

GAPS command

Detects whether a numeric or datetime field in an Analytics table contains one or more gaps in sequential data.

Syntax

GAPS <ON> *key_field* <D> <UNFORMATTED> <PRESORT> <MISSING *limit*> <HEADER *header_text*> <FOOTER *footer_text*> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <TO {SCREEN|*table_name*|*filename*|PRINT}> <LOCAL> <APPEND> <OPEN>

Parameters

Name	Description
ON *key_field* D	The fields or expressions to check for gaps. Include D to sort the key field in descending order. The default sort order is ascending.
UNFORMATTED optional	Suppresses page headings and page breaks when the results are output to a file.
PRESORT optional	Sorts the table on the key field before executing the command. **Note** You cannot use PRESORT inside the GROUP command.
MISSING *limit* optional	The output results contain individual missing items rather than gap ranges. The *limit* value specifies the maximum number of missing items to report for each identified gap. The default value is 5. If the limit is exceeded for a particular gap, the missing items are reported as a range for that particular gap. The *limit* value does not restrict the total number of missing items reported, it only restricts the number of missing items reported within a specific gap.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.
IF *test*	A conditional expression that must be true in order to process each record. The command

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
optional	is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
TO SCREEN | *table_name* | *filename* | PRINT optional	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Use either an absolute or relative file path to save the file to a different, existing folder: • TO "C:\Output.TXT" • TO "Results\Output.TXT" ◦ **PRINT** - sends the results to the default printer
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.

Analytics output variables

Name	Contains
GAPDUP*n*	The total number of gaps, duplicates, or fuzzy duplicate groups identified by the command.

Examples

Testing for missing invoice numbers

You use GAPS to ensure that there are no invoice numbers missing from an **Invoices** table:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
OPEN Invoices
GAPS ON Inv_Num PRESORT TO "Invoices_Gaps.fil"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Using GAPS on character fields

In addition to testing numeric or datetime fields, you can also test for gaps in numeric data that appears in a character field. For example, you can test check numbers, which are typically formatted as character data.

If letters and numbers appear together in a character field, only the numbers are tested and the letters are ignored.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

GROUP command

Executes one or more ACLScript commands on a record before moving to the next record in the table, with only one pass through the table. Command execution can be controlled by conditions.

Syntax

```
GROUP <IF test> <WHILE test> <FIRST range|NEXT range>
 command
 <...n>
<ELSE IF test>
 command
 <...n>
<ELSE>
 command
 <...n>
END
```

> **Note**
>
> Some Analytics commands cannot be used with the GROUP command. For more information, see "Commands that can be used inside the GROUP command" on page 231.

Parameters

Name	Description
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
command <...n>	One or more ACLScript commands to execute inside the GROUP. For a complete list of commands supported inside GROUP, see "Commands that can be used inside the GROUP command" on page 231. If there is a preceding IF or ELSE IF, the test must evaluate to true. If the command is listed under ELSE, the command is executed if there are records that have not been processed by any of the preceding commands. You can include multiple commands, with each command starting on a separate line.
ELSE IF *test* optional	Opens an ELSE IF block for the GROUP command. The condition tests records that did not match the GROUP command test, or any previous ELSE IF tests. You can include multiple ELSE IF tests and they are evaluated from top to bottom, until the record evaluates to true and the commands that follow that ELSE IF statement are executed.
ELSE optional	Opens an ELSE block for the GROUP command. The commands that follow are executed for records that evaluated to false for all of the previous tests.
END	The end of the GROUP command.

Examples

Simple GROUP

Simple groups start with a GROUP command, are followed by a series of commands, and terminate with an END command:

```
GROUP
 COUNT
 HISTOGRAM ON Quantity MINIMUM 0 MAXIMUM 100 INTERVALS 10
 CLASSIFY ON Location SUBTOTAL Quantity
END
```

GROUP IF

Conditional groups execute commands based on whether a condition is true or false. The following GROUP command is executed only on records with a **Product_class** value less than 5:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
GROUP IF Product_class < "05"
 COUNT
 HISTOGRAM ON Quantity MINIMUM 0 MAXIMUM 100 INTERVALS 10
 CLASSIFY ON Location SUBTOTAL Quantity
END
```

GROUP IF ...ELSE

Records that do not meet the test condition are ignored unless you include an ELSE block.

Any number of commands can follow an ELSE statement. In the following example, all records that do not meet the condition are processed by having their **Quantity** field totaled:

```
GROUP IF Product_class < "05"
 COUNT
 HISTOGRAM ON Quantity MINIMUM 0 MAXIMUM 100 INTERVALS 10
 CLASSIFY ON Location SUBTOTAL Quantity
ELSE
 TOTAL Quantity
END
```

GROUP IF...ELSE IF...ELSE

You can include multiple ELSE IF blocks within a group, as long as each ELSE IF block contains a different test. In the following example, the ELSE IF blocks, and the ELSE block, produce four totals:

```
GROUP IF Product_class < "05"
 COUNT
 HISTOGRAM ON Quantity MINIMUM 0 MAXIMUM 100 INTERVALS 10
 CLASSIFY ON Location SUBTOTAL Quantity
ELSE IF Product_class = "05"
 TOTAL Quantity
ELSE IF Product_class = "06"
 TOTAL Quantity
ELSE IF Product_class = "07"
 TOTAL Quantity
ELSE
 TOTAL Quantity
END
```

Nested GROUP commands

Nested groups refer to groups contained within other groups. Nested groups provide a powerful way for you to control which commands are executed for which records. Most applications do not require such an advanced level of functionality, but it is available, if necessary.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

As with other groups, use the END command to terminate a nested group. Analytics starts processing the data only after all group commands have been terminated:

```
GROUP IF Product_class < "05"
 COUNT
 STRATIFY ON Quantity SUBTOTAL Quantity MIN 0 MAX 100 INT 10
 GROUP IF Quantity > 0
  STATISTICS ON Quantity
  HISTOGRAM ON Quantity
 END
ELSE
 TOTAL Quantity
END
```

In this example, all of the commands from COUNT up to and including the next GROUP are executed only if **Product_class** is less than 05.

The STATISTICS and HISTOGRAM commands are executed if **Quantity** is greater than zero. However, because the second GROUP command is nested, the STATISTICS and HISTOGRAM commands are executed only for records that meet the conditions **Product_class** < "05" and **Quantity** > 0.

Generating system variables inside a GROUP

You can use GROUP to create multiple system variables for a single command.

Normally, when you run a command such as TOTAL, COUNT, or STATISTICS, only one system variable is generated. Each time you run the command, you overwrite the value from the last execution of the command. Commands that run inside a GROUP create a specific variable for each instance of the command inside the GROUP.

In this example, the TOTAL command calculates the sum of the **Amount** field for each product class in the **Metaphor_Trans_2002** table. When the code runs, the following variables are generated and can be used in subsequent commands after the GROUP:

- **TOTAL2** - the sum of the **Amount** field for product class 03
- **TOTAL3** - the sum of the **Amount** field for product class 05
- **TOTAL4** - the sum of the **Amount** field for product class 08
- **TOTAL5** - the sum of the **Amount** field for product class 09

```
OPEN Metaphor_Trans_2002
GROUP
 TOTAL AMOUNT IF PRODCLS = "03"
 TOTAL AMOUNT IF PRODCLS = "05"
 TOTAL AMOUNT IF PRODCLS = "08"
 TOTAL AMOUNT IF PRODCLS = "09"
END
CLOSE Metaphor_Trans_2002
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

> **Tip**
>
> For a detailed tutorial covering the GROUP and LOOP commands, see "Grouping and looping" on page 32.

Commands that can be used inside the GROUP command

The table below lists the Analytics commands that can be used inside the GROUP command.

If a command is not listed below, it cannot be used inside GROUP.

AGE	ASSIGN	BENFORD
CLASSIFY	COMMENT	COUNT
CROSSTAB	DUPLICATES	EXPORT
EXTRACT	GAPS	GROUP
HISTOGRAM	JOIN	LIST
LOOP	MERGE	PROFILE
REPORT	SEQUENCE	STATISTICS
STRATIFY	SUMMARIZE	TOTAL
VERIFY		

Grouping and looping

The GROUP command allows you to execute several commands on a record before moving to the next record in the table, which can significantly reduce processing time.

You can use the LOOP command inside the GROUP command if you need to execute a series of commands more than once against a record.

Using variables with GROUP

User-defined variables

To use a variable inside the GROUP command, define the variable before you enter the GROUP block.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> While you can initialize and define a variable inside a GROUP block, it is not recommended. Variables initialized inside a GROUP may cause unexpected results when used.

Inside a GROUP, you can evaluate variables using variable substitution. The value of the variable remains the same as when the GROUP is entered.

You cannot define a variable inside a GROUP and then reference it using variable substitution:

```
ASSIGN v_test = "hello"
GROUP
 ASSIGN v_test2 = "%v_test% world"
 COMMENT this would be invalid: v_test3 = "%v_test2% again"
END
```

System-defined variables

Certain commands such as TOTAL and STATISTICS generate system variables based on calculations that the commands perform. If you use a GROUP to execute these commands, any system variables that result are numbered consecutively, starting at the line number of the command inside the GROUP (excluding empty lines) and running to *n*. The value of *n* increases by 1 for each line number in the GROUP.

> **Note**
>
> You must wait until the GROUP completes before using any system generated variables created inside the GROUP. The command must run against each record in the table before the variable is available. Use these variables after the closing END keyword of the GROUP.

In the following example, the first TOTAL command generates the variable TOTAL2 and the second generates TOTAL4. Both of these variables are available to use in subsequent commands once the GROUP completes:

```
GROUP
 TOTAL Discount IF Order_Priority = "Low"
 ASSIGN v_var = "test"
 TOTAL Discount IF Order_Priority = "High"
END
```

Syntax notes

- The multiline syntax listed for the GROUP command is required, and therefore the GROUP command cannot be entered in the command line.
- Each GROUP command must be terminated with an END command.
- When you use the GROUP command in your scripts, you can improve the readability of the command block by indenting the commands listed inside the group.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

HELP command

Launches the Analytics Help Docs in a browser.

Syntax

```
HELP
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

HISTOGRAM command

Groups records based on values in a character or numeric field, counts the number of records in each group, and displays the groups and counts in a bar chart.

Syntax

HISTOGRAM {<ON> *character_field*|<ON> *numeric_field* MINIMUM *value* MAXIMUM *value* {<INTERVALS *number*>|FREE *interval_value* <...n> *last_interval*}} <TO {SCREEN|*filename*|GRAPH|PRINT}> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <HEADER *header_text*> <FOOTER *footer_text*> <KEY *break_field*> <SUPPRESS> <COLUMNS *number*> <APPEND> <OPEN>

Parameters

Name	Description
ON *character_field*	The character field or expression to use for the histogram.
ON *numeric_field*	The numeric field or expression to use for the histogram.
MINIMUM *value*	Applies to numeric fields only. The minimum value of the first numeric interval. MINIMUM is optional if you are using FREE, otherwise it is required.
MAXIMUM *value*	Applies to numeric fields only. The maximum value of the last numeric interval. MAXIMUM is optional if you are using FREE, otherwise it is required.
INTERVALS *number* optional	Applies to numeric fields only. The number of equal-sized intervals Analytics produces over the range specified by the MINIMUM and MAXIMUM values. If you do not specify a number of intervals, the default number is used. The default is specified by the **Intervals** number on the **Command** tab in the **Options** dialog box.
FREE *interval_value* <...n> *last_interval* optional	Applies to numeric fields only. Creates custom-sized intervals by specifying the start point of each interval and the end point of the last interval. If you specify MINIMUM and MAXIMUM values, those values are the start point of the first interval and the end point of the last interval, and each *interval_value* creates an additional interval within the range. The interval values you specify must be greater than the MINIMUM value, and equal to or less than the MAXIMUM value.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Interval values must be in numeric sequence and cannot contain duplicate values: `FREE -1000, 0, 1000, 2000, 3000` If you specify both FREE and INTERVALS, then INTERVALS is ignored.
TO SCREEN | *filename* | GRAPH | PRINT	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"` ◦ **GRAPH** - displays the results in a graph in the Analytics display area ◦ **PRINT** - sends the results to the default printer **Note** Histogram results output to a file appear as a textual representation of a bar chart.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.
KEY *break_field* optional	The field or expression that groups subtotal calculations. A subtotal is calculated each time the value of *break_field* changes. *break_field* must be a character field or expression. You can specify only one field, but you can use an expression that contains more than one field.
SUPPRESS optional	Values above the MAXIMUM value and below the MINIMUM value are excluded from the command output.
COLUMNS *number* optional	The length of the x-axis in the textual representation of the bar chart if you output histogram results to a text file. The number value is the number of character spaces (text columns) to use for the x-axis (and the y-axis labels). If you omit COLUMNS, the default of 78 character spaces is used.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.

Examples

Basic histogram for hourly salary

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You use HISTOGRAM to create a graph showing the distribution of wages between 0 and 100 dollars per hour:

```
HISTOGRAM ON Rate MINIMUM 0 MAXIMUM 100 TO GRAPH
```

Histogram with defined intervals for hourly salary

Continuing from the previous example, you use HISTOGRAM to specify the ranges in the graph in a more meaningful way.

Most of the wages fall between 20 and 50 dollars per hour, so the graph includes the following number of intervals:

- three in the 20 to 50 range
- one for 0-20
- one for 50-100
- one for > 100

```
HISTOGRAM ON Rate MINIMUM 0 MAXIMUM 100 FREE 20,30,40,50,100 TO GRAPH
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Populating low and high values

You can run the STATISTICS or PROFILE commands on a numeric field before running the HISTOGRAM command to automatically populate the MINIMUM and MAXIMUM parameter values with the lowest and highest values in the field.

Related commands

Creating a histogram using a character field is similar to classifying. Creating a histogram using a numeric field is similar to stratifying.

Unlike the other grouping operations in Analytics, histograms do not support subtotaling numeric fields.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IF command

Specifies a condition that must evaluate to true in order to execute a command.

Syntax

```
IF test command
```

Parameters

Name	Description
test	The condition that must be met for *command* to be run.
command	Any valid ACLScript command to run if *test* evaluates to true.

Examples

Running a command conditionally

You want to use CLASSIFY on a table, but only if the *v_counter* variable is greater than ten:

```
IF v_counter > 10 CLASSIFY ON Location TO "Count_by_Location.fil" OPEN
```

Running a command based on a user decision

You want to allow the script user to decide whether to classify a table.

In your script, you include a dialog box with a check box that if selected allows the CLASSIFY command to run. The check box stores a True or False input value in the logical variable *v_classify_checkbox*.

You use an IF test to determine the value of *v_classify_checkbox*, and if the value is True, CLASSIFY executes:

```
IF v_classify_checkbox=T CLASSIFY ON Location TO "Count_by_Location.fil" OPEN
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

IF command versus IF parameter

The logic of the IF command differs from the IF parameter that is supported by most commands:

- **IF command** – determines whether the associated command runs or not, based on the value of the test expression
- **IF parameter** – determines whether the command runs against each record in an Analytics table based on the value of the test expression

Decision making in scripts

In a script, you can enter a series of IF command tests and run different commands based on the results. The IF command can be also be used to test the value of a variable to determine if further processing should occur.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT ACCESS command

Creates an Analytics table by defining and importing a Microsoft Access database file.

Syntax

IMPORT ACCESS TO *table* <PASSWORD *num*> *import_filename* FROM *source_filename* TABLE *input_tablename* CHARMAX *max_field_length* MEMOMAX *max_field_length*

Parameters

Name	Description
TO *table*	The name of the Analytics table to import the data into. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
PASSWORD *num* optional	Used only with password-protected Access files. The password definition to use. You do not use PASSWORD *num* to prompt for, or specify, an actual password. The password definition refers to a password previously supplied or set using the PASSWORD command, the SET PASSWORD command, or the PASSWORD analytic tag. *num* is the number of the password definition. For example, if two passwords have been previously supplied or set in a script, or when scheduling an analytic, PASSWORD 2 specifies that password #2 is used. For more information about supplying or setting passwords, see: • "PASSWORD command" on page 360 • "SET command" on page 418 • PASSWORD analytic tag
import_filename	The name of the Analytics data file to create. Specify *import_filename* as a quoted string with a .FIL file extension. For example, "Invoices.FIL". By default, the data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ "C:\data\Invoices.FIL" ◦ "data\Invoices.FIL"

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
FROM *source_filename*	The name of the source data file. *source_filename* must be a quoted string. If the source data file is not located in the same directory as the Analytics project, you must use an absolute path or a relative path to specify the file location: ◦ "C:\data*source_filename*" ◦ "data*source_filename*"
TABLE *input_tablename*	The name of the table in the Microsoft Access database file to import.
CHARMAX *max_field_length*	The maximum length in characters for any field in the Analytics table that originates as character data in the source from which you are importing. You can specify from 1 to 255 characters.
MEMOMAX *max_field_length*	The maximum length in characters for text, note, or memo fields you are importing. You can specify from 1 to 32767 characters (non-Unicode Analytics), or 16383 characters (Unicode Analytics).

Examples

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Importing into a table

You have a Microsoft Access file called `Acceptable_Codes.mdb`. You need to import the **[Acceptable_Codes]** table from the file into Analytics. To do this, you use the following command and create a table called **acc_codes** in Analytics.

The length of imported character or memo fields is set to the length of the longest value in the field, or to the specified maximum number of characters, whichever is shorter:

```
SET ECHO NONE
SET PASSWORD 1 TO "qr347wx"
SET ECHO ON
IMPORT ACCESS TO acc_codes PASSWORD 1 "C:\ACL DATA\Sample Data Files\acc_codes.fil"
FROM "Acceptable_Codes.mdb" TABLE "[Acceptable_Codes]" CHARMAX 60 MEMOMAX 70
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT DELIMITED command

Creates an Analytics table by defining and importing a delimited text file.

Syntax

IMPORT DELIMITED TO *table import_filename* FROM *source_filename* <SERVER *profile_name*> *source_char_encoding* SEPARATOR {*char*|TAB|SPACE} QUALIFIER {*char*|NONE} <CONSECUTIVE> STARTLINE *line_number* <KEEPTITLE> <CRCLEAR> <LFCLEAR> <REPLACENULL> <ALLCHAR> {ALLFIELDS|[field_syntax] <...*n*> <IGNORE *field_num*> <...*n*>}

field_syntax ::=
FIELD *name type* AT *start_position* DEC *value* WID *bytes* PIC *format* AS *display_name*

Parameters

Name	Description
TO *table*	The name of the Analytics table to import the data into. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
import_filename	The name of the Analytics data file to create. Specify *import_filename* as a quoted string with a .FIL file extension. For example, "Invoices.FIL". By default, the data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ "C:\data\Invoices.FIL" ◦ "data\Invoices.FIL"
FROM *source_filename*	The name of the source data file. *source_filename* must be a quoted string. If the source data file is not located in the same directory as the Analytics project, you must use an absolute path or a relative path to specify the file location: ◦ "C:\data*source_filename*" ◦ "data*source_filename*"
SERVER *profile_name*	The server profile name for the AX Server where the data you want to import is located.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>optional</td><td></td></tr>
<tr><td>source_char_encoding</td><td>The character set and encoding of the source data.

Depending on which edition of Analytics you are using, and the encoding of the source data, specify the appropriate code:

<table>
<tr><th>Code</th><th>Analytics edition</th><th>Source data encoding</th></tr>
<tr><td>0</td><td>Non-Unicode edition</td><td>all data</td></tr>
<tr><td>0</td><td>Unicode edition</td><td>ASCII data</td></tr>
<tr><td>2</td><td>Unicode edition</td><td>Unicode data, UTF-16 LE encoding</td></tr>
<tr><td>3 numeric_code</td><td>Unicode edition</td><td>Unicode data that does not use UTF-16 LE encoding

To determine the numeric code that matches the source data encoding, perform an import using the Data Definition Wizard, select the Encoded Text option, and find the matching encoding in the accompanying drop-down list.

To specify the code, specify 3, followed by a space, and then the numeric code.</td></tr>
</table>
</td></tr>
<tr><td>SEPARATOR char | TAB | SPACE</td><td>The separator character (delimiter) used between fields in the source data. You must specify the character as a quoted string.

You can specify a tab or a space separator by typing the character between double quotation marks, or by using a keyword:

◦ SEPARATOR "	" or SEPARATOR TAB
◦ SEPARATOR " " or SEPARATOR SPACE</td></tr>
<tr><td>QUALIFIER char | NONE</td><td>The text qualifier character used in the source data to wrap and identify field values. You must specify the character as a quoted string.

To specify the double quotation mark character as the text qualifier, enclose the character in single quotation marks: QUALIFIER '"'.

You can specify that there are no text qualifiers using either of these methods:

◦ QUALIFIER ""
◦ QUALIFIER NONE</td></tr>
<tr><td>CONSECUTIVE

optional</td><td>Consecutive text qualifiers are treated as a single qualifier.</td></tr>
<tr><td>STARTLINE line_number</td><td>The line number on which to start reading the file.

For example, if the first three lines of a file contain header information that you do not want, specify STARTLINE 4 to start reading data on the fourth line.</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
KEEPTITLE optional	◦ **KEEPTITLE used with ALLFIELDS** - Treat the line number specified by STARTLINE as field names instead of data. If you omit KEEPTITLE, generic field names are used and the line number specified by STARTLINE is treated as data. ◦ **KEEPTITLE used with individual FIELD syntax** - Do not import the line number specified by STARTLINE. FIELD *name* specifies the field names. If you omit KEEPTITLE, the line number specified by STARTLINE is treated as data. FIELD *name* specifies the field names.
CRCLEAR optional	Replaces any CR characters (carriage return) that occur between text qualifiers with space characters. You must specify QUALIFIER with a *char* value to use CRCLEAR. If you use both CRCLEAR and LFCLEAR, CRCLEAR must come first.
LFCLEAR optional	Replaces any LF characters (line feed) that occur between text qualifiers with space characters. You must specify QUALIFIER with a *char* value to use LFCLEAR. If you use both CRCLEAR and LFCLEAR, CRCLEAR must come first.
REPLACENULL optional	Replaces any NUL characters that occur in the delimited file with space characters. The number of any replaced NUL characters is recorded in the log.
ALLCHAR optional	The Character data type is automatically assigned to all the imported fields. **Tip** Assigning the Character data type to all the imported fields simplifies the process of importing delimited text files. Once the data is in Analytics, you can assign different data types, such as Numeric or Datetime, to the fields, and specify format details. ALLCHAR is useful if you are importing a table with identifier fields automatically assigned the Numeric data type by Analytics when in fact they should use the Character data type.
ALLFIELDS	All fields in the source data file are imported. For information about how Analytics assigns data types when you use ALLFIELDS, see "Remarks" on page 247. **Note** If you specify ALLFIELDS, do not specify any individual FIELD syntax, or IGNORE.
FIELD *name type*	The individual fields to import from the source data file, including the name and data type of the field. To exclude a field from being imported, do not specify it. For information about *type*, see "Identifiers for field data types" on page 248. **Note** *type* is ignored if you specify ALLCHAR.
AT *start_position*	The starting byte position of the field in the Analytics data file.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>

Note

non-Unicode Analytics	1 byte = 1 character
Unicode Analytics	2 bytes = 1 character

In Unicode Analytics, typically you should specify an odd-numbered starting byte position. Specifying an even-numbered starting position can cause characters to display incorrectly.
</td></tr>
<tr><td>DEC value</td><td>The number of decimals for numeric fields.

Note

DEC is ignored if you specify ALLCHAR.
</td></tr>
<tr><td>WID bytes</td><td>The length in bytes of the field in the Analytics table layout.

Note

non-Unicode Analytics	1 byte = 1 character
Unicode Analytics	2 bytes = 1 character

In Unicode Analytics, specify an even number of bytes only. Specifying an odd number of bytes can cause characters to display incorrectly.
</td></tr>
<tr><td>PIC format</td><td>

Note

Applies to numeric or datetime fields only.

- **numeric fields** - the display format of numeric values in Analytics views and reports
- **datetime fields** - the physical format of datetime values in the source data (order of date and time characters, separators, and so on)

Note

For datetime fields, *format* must exactly match the physical format in the source data. For example, if the source data is 12/31/2014, you must enter the format as "MM/DD/YYYY".

format must be enclosed in quotation marks.

Note

PIC is ignored if you specify ALLCHAR.
</td></tr>
<tr><td>AS display_name</td><td>The display name (alternate column title) for the field in the view in the new Analytics table.

Specify display_name as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title.

AS is required when you are defining FIELD. To make the display name the same as the field name, enter a blank display_name value using the following syntax: AS "". Make sure there is no space between the two double quotation marks.</td></tr>
<tr><td>IGNORE <i>field_num</i> <...<i>n</i>></td><td>Excludes the field from the table layout.</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
optional	*field_num* specifies the position of the excluded field in the source data file. For example, IGNORE 5 excludes the fifth field in the source data file from the Analytics table layout. **Note** The data in the field is still imported, but it is undefined, and does not appear in the new Analytics table. The data can be defined later, if necessary, and added to the table. To completely exclude a field from being imported, do not specify it when you specify fields individually.

Examples

Import all fields

You import all fields from a comma delimited file to an Analytics table named **Employees**. The file uses double quotation marks as the text qualifiers. Data types are automatically assigned based on the set of rules outlined in "Remarks" on the facing page:

```
IMPORT DELIMITED TO Employees "Employees.fil" FROM "Employees.csv" 0 SEPARATOR "," QUALIFIER "" CONSECUTIVE STARTLINE 1 KEEPTITLE ALLFIELDS
```

Import all fields, automatically assign a Character data type

You import all fields from a comma delimited file to an Analytics table named **Employees**. The file uses double quotation marks as the text qualifiers. The Character data type is automatically assigned to all imported fields:

```
IMPORT DELIMITED TO Employees "Employees.fil" FROM "Employees.csv" 0 SEPARATOR "," QUALIFIER "" CONSECUTIVE STARTLINE 1 KEEPTITLE ALLCHAR ALLFIELDS
```

Import specified fields, automatically assign a Character data type

You import specified fields from a tab delimited file to an Analytics table named **Employees**. The file uses double quotation marks as the text qualifiers. The Character data type is automatically assigned to all imported fields:

```
IMPORT DELIMITED TO Employees "Employees.fil" FROM "Employees.csv" 0 SEPARATOR TAB QUALIFIER "" CONSECUTIVE STARTLINE 1 KEEPTITLE ALLCHAR FIELD "First_Name" C AT 1 DEC 0 WID 25 PIC "" AS "First Name" FIELD "Last_Name" C AT 26 DEC 0 WID 25 PIC "" AS "Last Name" FIELD "CardNum" C AT 51 DEC 0 WID 16 PIC "" AS "Card Num" FIELD "EmpNo" C AT 67
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DEC 0 WID 6 PIC "" AS "Emp Num" FIELD "HireDate" C AT 73 DEC 0 WID 10 PIC "" AS "Hire Date"
FIELD "Salary" C AT 83 DEC 0 WID 5 PIC "" AS "" FIELD "Bonus_2016" C AT 88 DEC 0 WID 10 PIC
"" AS "Bonus 2016"
```

Import specified fields, assign data types individually

You import specified fields from a semi-colon delimited file to an Analytics table named **Employees**. The file does not use text qualifiers. You specify the data type of each imported field:

```
IMPORT DELIMITED TO Employees "Employees.fil" FROM "Employees.csv" 0 SEPARATOR ";"
QUALIFIER "" CONSECUTIVE STARTLINE 1 KEEPTITLE FIELD "First_Name" C AT 1 DEC 0 WID
25 PIC "" AS "First Name" FIELD "Last_Name" C AT 26 DEC 0 WID 25 PIC "" AS "Last Name" FIELD
"CardNum" C AT 51 DEC 0 WID 16 PIC "" AS "Card Num" FIELD "EmpNo" C AT 67 DEC 0 WID 6 PIC
"" AS "Emp Num" FIELD "HireDate" D AT 73 DEC 0 WID 10 PIC "MM/DD/YYYY" AS "Hire Date"
FIELD "Salary" N AT 83 DEC 0 WID 5 PIC "" AS "" FIELD "Bonus_2016" N AT 88 DEC 2 WID 10 PIC
"" AS "Bonus 2016"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

How Analytics assigns data types when you use ALLFIELDS

When you use the ALLFIELDS parameter, instead of defining fields individually, Analytics examines a sub-set of records at the beginning of the delimited file and assigns data types to fields based on the set of rules outlined below.

Once the data is in Analytics, if required you can assign different data types, such as Numeric or Datetime, to the fields, and specify format details.

Description of field values in the delimited file	Examples	Data type assigned
Values enclosed by text qualifiers	"ABC Suppliers" "6,990.75"	Character
Values include a non-numeric character anywhere in the field, with the exception of commas and periods used as numeric separators, and the negative sign (-)	$995 (995)	Character
Values include only numbers, numeric separators, or the negative sign	6,990.75 -6,990.75	Numeric

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Description of field values in the delimited file	Examples	Data type assigned
	995	
One or more blank values occur in a field		Character
Datetime values with separators, or alpha months	2016/12/31 31 Dec 2016	Character
Datetime values that are all numbers	20161231	Numeric

Identifiers for field data types

The table below lists the letters that you must use when specifying *type* for `FIELD`. Each letter corresponds to an Analytics data type.

For example, if you are defining a Last Name field, which requires a character data type, you would specify "C": `FIELD "Last_Name" C`.

For more information, see Analytics data types.

> **Note**
>
> When you use the **Data Definition Wizard** to define a table that includes EBCDIC, Unicode, or ASCII fields, the fields are automatically assigned the letter "C" (for the CHARACTER type).
>
> When you enter an IMPORT statement manually, or edit an existing IMPORT statement, you can substitute the more specific letters "E" or "U" for EBCDIC or Unicode fields.

Letter	Analytics Data type
A	ACL
B	BINARY
C	CHARACTER
D	DATETIME
E	EBCDIC
F	FLOAT
G	ACCPAC
I	IBMFLOAT
K	UNSIGNED

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Letter	Analytics Data type
L	LOGICAL
N	PRINT
P	PACKED
Q	BASIC
R	MICRO
S	CUSTOM
T	PCASCII
U	UNICODE
V	VAXFLOAT
X	NUMERIC
Y	UNISYS
Z	ZONED

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT EXCEL command

Creates an Analytics table by defining and importing a Microsoft Excel worksheet or named range.

Syntax

IMPORT EXCEL TO *table import_filename* FROM *source_filename* TABLE *input_worksheet_or_named_range* <KEEPTITLE> <STARTLINE *line_number*> <ALLCHAR> {ALLFIELDS|CHARMAX *max_field_length*|[field_syntax] <...*n*> <IGNORE *field_num*> <...*n*>} <OPEN>

field_syntax ::=
FIELD *import_name type* {PIC *format*|WID *characters* DEC *value*} AS *display_name*

> **Note**
>
> You must specify the IMPORT EXCEL parameters in exactly the same order as above, and in the table below.
>
> Analytics cannot import from an Excel workbook if Protected View is active for the workbook. You must first enable editing in the workbook, save and close the workbook, and then perform the import.

Parameters

Name	Description
TO *table*	The name of the Analytics table to import the data into. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
import_filename	The name of the Analytics data file to create. Specify *import_filename* as a quoted string with a .FIL file extension. For example, "Invoices.FIL". By default, the data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ "C:\data\Invoices.FIL" ◦ "data\Invoices.FIL"
FROM *source_filename*	The name of the source data file. *source_filename* must be a quoted string.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	If the source data file is not located in the same directory as the Analytics project, you must use an absolute path or a relative path to specify the file location: ◦ "C:\data*source_filename*" ◦ "data*source_filename*"
TABLE *worksheet_or_named_range*	The worksheet or the named range to import from the Microsoft Excel source data file. Requirements: ◦ add a "$" sign at the end of a worksheet name For example, TABLE "Corp_Credit_Cards$" ◦ specify a named range exactly as it appears in Excel For example, TABLE "Employees_Sales" ◦ specify *worksheet_or_named_range* as a quoted string
KEEPTITLE optional	◦ **KEEPTITLE used with ALLFIELDS or CHARMAX** – Treat the line number specified by STARTLINE as field names instead of data. If you omit KEEPTITLE, generic field names are used and the line number specified by STARTLINE is treated as data. ◦ **KEEPTITLE used with individual FIELD syntax** – Do not import the line number specified by STARTLINE. FIELD *name* specifies the field names. If you omit KEEPTITLE, the line number specified by STARTLINE is treated as data. FIELD *name* specifies the field names.
STARTLINE *line_number* optional	The line number on which to start reading the worksheet. For example, if the first three lines of a worksheet contain header information that you do not want, specify STARTLINE 4 to start reading data on the fourth line. If you omit STARTLINE, the start line is the first line in the worksheet. **Note** The start line for a named range is always the first line in the named range, regardless of the STARTLINE setting.
ALLCHAR optional	The Character data type is automatically assigned to all the imported fields. **Tip** Assigning the Character data type to all the imported fields simplifies the process of importing Excel files. Once the data is in Analytics, you can assign different data types, such as Numeric or Datetime, to the fields, and specify format details. ALLCHAR is useful if you are importing a table with identifier fields automatically assigned the Numeric data type by Analytics when in fact they should use the Character data type.
ALLFIELDS	All fields in the source data file are imported.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note If you specify ALLFIELDS, do not specify any individual FIELD syntax, CHARMAX, or IGNORE.
CHARMAX *max_field_length*	The maximum length in characters for any field in the Analytics table that originates as character data in the source data file. Source character data that exceeds the maximum is truncated. All fields in the source data file, regardless of data type, are imported. **Note** If you specify CHARMAX, do not specify any individual FIELD syntax, ALLFIELDS, or IGNORE.
FIELD *import_name type*	The individual fields to import from the source data file, including the name and data type of the field. *import_name* becomes the field name in the Analytics table. *import_name* does not need to be the same as the field name in the source data file, although it can be. **Tip** You can additionally use AS to specify a display name that is different from *import_name*. *type* becomes the field date type in the Analytics table. *type* does not need to be the same as the field data type in the source data file, although it can be. For more information about *type*, see "Identifiers for field data types" on page 256. **Note** If you specify ALLCHAR, *type* is ignored. If you specify individual FIELD syntax, do not specify ALLFIELDS or CHARMAX. ## Excluding a field To exclude a field from being imported, do not specify it. You must also specify IGNORE for excluded fields.
PIC *format*	**Note** Applies to numeric or datetime fields only. ◦ **numeric fields** - the display format of numeric values in Analytics views and reports ◦ **datetime fields** - the physical format of datetime values in the source data (order of date and time characters, separators, and so on) **Note** For datetime fields, *format* must exactly match the physical format in the source data. For example, if the source data is 12/31/2014, you must enter the format as "MM/DD/YYYY". *format* must be enclosed in quotation marks.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
WID *characters*	The length in characters of the field in the Analytics table layout.
DEC *value*	The number of decimals for numeric fields.
AS *display_name*	The display name (alternate column title) for the field in the view in the new Analytics table. Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title. AS is required when you are defining FIELD. To make the display name the same as the field name, enter a blank *display_name* value using the following syntax: AS "". Make sure there is no space between the two double quotation marks.
IGNORE *field_num* <...*n*> optional	Excludes the field from the table layout. *field_num* specifies the position of the excluded field in the source data file. For example, IGNORE 5 excludes the fifth field in the source data file from the Analytics table layout. **Note** Be careful to correctly align *field_num* with the position of excluded fields. If you specify *field_num* for an included field (FIELD definition), or for a field position that does not exist, the import does not work correctly. The number of FIELD and IGNORE parameters combined must equal the total number of fields in the source data table. If the total numbers do not match, the import does not work correctly. If you specify ALLFIELDS or CHARMAX, do not specify IGNORE.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.

Examples

Import specified fields

You perform an import that defines a new Analytics table called **Credit_Cards**. It uses the first row of Excel data as the field names.

From the twelve fields in the source table, the Analytics table defines and includes three fields and excludes nine fields:

```
IMPORT EXCEL TO Credit_Cards "Credit Cards.fil" FROM "Credit_Cards_Metaphor.xls" TABLE
"Corp_Credit_Cards$" KEEPTITLE FIELD "CARDNUM" C WID 16 AS "Card Number" FIELD
"EXPDT" D WID 10 PIC "YYYY-MM-DD" AS "Expiry Date" FIELD "PASTDUEAMT" N WID 6 DEC 2
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
AS "Past Due" IGNORE 2 IGNORE 3 IGNORE 5 IGNORE 6 IGNORE 7 IGNORE 9 IGNORE 10
IGNORE 11 IGNORE 12
```

Import all fields

You perform an import that defines a new Analytics table called **May_Transactions**. It uses the first row of Excel data as the field names.

The Analytics table includes all fields from the source table and uses default field definitions.

Field length set to longest value

In the first example, fields that originate as character data in the source data file are set to the length of the longest value in the field:

```
IMPORT EXCEL TO May_Transactions "May_Transactions.fil" FROM "Trans_May.xls" TABLE
"Trans1_May$" KEEPTITLE ALLFIELDS
```

Field length constrained

In the second example, fields that originate as character data in the source data file are set to the length of the longest value in the field, or to the CHARMAX value of 50 characters, whichever is shorter:

```
IMPORT EXCEL TO May_Transactions "May_Transactions.fil" FROM "Trans_May.xls" TABLE
"Trans1_May$" KEEPTITLE CHARMAX 50
```

Import all fields as character data

You perform an import that defines a new Analytics table called **May_Transactions**. All fields, including numbers and dates, are imported as character data.

```
IMPORT EXCEL TO May_Transactions "May_Transactions.fil" FROM "Trans_May.xls" TABLE
"Trans1_May$" KEEPTITLE ALLCHAR ALLFIELDS
```

Import all fields as character data, skip header information

You perform an import that defines a new Analytics table called **Past_Due_Report**.

You skip the first two rows of the Excel file, which contain report header information, and start reading the file on the third row, which contains field names. All fields, including numbers and dates, are imported as character data.

```
IMPORT EXCEL TO Past_Due_Report "Past_Due_Report.fil" FROM "Past_Due_Report.xlsx"
TABLE "Sheet1$" KEEPTITLE STARTLINE 3 ALLCHAR ALLFIELDS
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Define fields individually, or import all fields using a default definition

When you import an Excel file to an Analytics table you can use FIELD parameters to define each field individually, or you can use the ALLFIELDS parameter, or the CHARMAX parameter, to import all fields using default Analytics field definitions.

Different combinations of parameters produce different results. The table below summarizes the different possibilities.

> **Note**
>
> "Define" means manually specifying things like field name, data type, length, datetime format, and so on.

I want to:	Use these parameters:	Do not use these parameters:
◦ import all fields automatically with default definitions ◦ if required, define fields post-import in Analytics	ALLFIELDS	CHARMAX, FIELD
◦ import all fields automatically with default definitions ◦ if required, define fields post-import in Analytics ◦ truncate long character fields	CHARMAX	ALLFIELDS, FIELD
◦ define fields pre-import	FIELD	ALLFIELDS, CHARMAX
◦ define fields pre-import ◦ exclude some fields from being imported	FIELD IGNORE	ALLFIELDS, CHARMAX
◦ partially define fields pre-import ◦ automatically import all fields as character data	ALLCHAR FIELD	ALLFIELDS, CHARMAX
◦ omit blank rows or header information at the top of a worksheet	STARTLINE	
◦ use the first row of the worksheet as field names	KEEPTITLE	
◦ use the row of the worksheet specified by STARTLINE as field names	KEEPTITLE STARTLINE	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

How Analytics assigns data types when you use ALLFIELDS or CHARMAX

When you use the ALLFIELDS or CHARMAX parameters, instead of defining fields individually, Analytics examines a subset of records at the beginning of the Excel file and assigns data types to fields based on a set of internal rules.

Once the data is in Analytics, if required you can assign different data types, such as Numeric or Datetime, to the fields, and specify format details.

Maximum size of data import

File format .xlsx or .xlsm

The maximum number of Excel columns, and the maximum number of characters in a field, that you can import from .xlsx or .xlsm files is not limited to a specific number.

Importing from these Excel file types is governed by the record length limit in Analytics data files (.fil) of 32 KB. If any record in the source Excel file would create an Analytics record longer than 32 KB, the import fails.

File format .xls

The import of .xls (Excel 97 - 2003) files uses a different type of processing, and is subject to maximums of:

- 255 columns
- 255 characters per field
- 32 KB per record
- 65,000 rows

Identifiers for field data types

The table below lists the letters that you must use when specifying *type* for `FIELD`. Each letter corresponds to an Analytics data type.

For example, if you are defining a Last Name field, which requires a character data type, you would specify "C": `FIELD "Last_Name" C`.

For more information, see Analytics data types.

> **Note**
>
> When you use the **Data Definition Wizard** to define a table that includes EBCDIC, Unicode, or ASCII fields, the fields are automatically assigned the letter "C" (for the CHARACTER type).
>
> When you enter an IMPORT statement manually, or edit an existing IMPORT statement, you can substitute the more specific letters "E" or "U" for EBCDIC or Unicode fields.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Letter	Analytics Data type
A	ACL
B	BINARY
C	CHARACTER
D	DATETIME
E	EBCDIC
F	FLOAT
G	ACCPAC
I	IBMFLOAT
K	UNSIGNED
L	LOGICAL
N	PRINT
P	PACKED
Q	BASIC
R	MICRO
S	CUSTOM
T	PCASCII
U	UNICODE
V	VAXFLOAT
X	NUMERIC
Y	UNISYS
Z	ZONED

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT GRCPROJECT command

Creates an Analytics table by importing a HighBond Projects table.

Syntax

IMPORT GRCPROJECT TO *table import_filename* PASSWORD *num* FROM *org_id*/*type_id* <FIELD *name* AS *display_name* <...*n*>>

Parameters

Name	Description
TO *table*	The name of the Analytics table to import the data into. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
import_filename	The name of the Analytics data file to create. Specify *import_filename* as a quoted string with a .FIL file extension. For example, "Invoices.FIL". By default, the data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ "C:\data\Invoices.FIL" ◦ "data\Invoices.FIL"
PASSWORD *num*	The password definition to use. You do not use PASSWORD *num* to prompt for, or specify, an actual password. The password definition refers to a password previously supplied or set using the PASSWORD command, the SET PASSWORD command, or the PASSWORD analytic tag. *num* is the number of the password definition. For example, if two passwords have been previously supplied or set in a script, or when scheduling an analytic, PASSWORD 2 specifies that password #2 is used. For more information about supplying or setting passwords, see: • "PASSWORD command" on page 360 • "SET command" on page 418 • PASSWORD analytic tag The required password value is a HighBond access token. For more information, see

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>"Creating a password definition and specifying a password value" on page 261.

Note
PASSWORD may or may not be required, depending on the environment in which the script runs:
<table><tr><td>Analytics
(online activation)</td><td>PASSWORD is not required.
The current user's HighBond access token is automatically used.</td></tr><tr><td>Analytics
(offline activation)</td><td rowspan="4">PASSWORD is required.</td></tr><tr><td>Robots</td></tr><tr><td>Analytics Exchange</td></tr><tr><td>Analysis App window</td></tr></table></td></tr>
<tr><td>FROM org_id/type_id</td><td>The organization and type of information that defines the data being imported:
◦ org_id - the Projects organization you are importing data from
◦ type_id - the type of information you are importing

The org_id value and the type_id value must be separated by a slash, with no intervening spaces: <code>FROM "125@eu/audits"</code>.

The entire string must be enclosed in quotation marks.

Organization ID

org_id must include the organization ID number, and if you are importing from a data center other than North America, the data center code. The organization ID number and the data center code must be separated by the at sign (@): <code>FROM "125@eu"</code>.

The data center code specifies which regional HighBond server you are importing the data from:
◦ ap - Asia Pacific
◦ au - Australia
◦ ca - Canada
◦ eu - Europe
◦ us - North America

You can use only the data center code or codes authorized for your organization's instance of HighBond. The North America data center is the default, so specifying <code>"@us"</code> is optional.

If you do not know the organization ID number, use the Analytics user interface to import a table from Projects. The organization ID number is contained in the command in the log. For more information, see Import HighBond Projects data.

Type ID</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	type_id specifies the type of information you are importing. Information in Projects is contained in a series of related tables. For *type_id*, use one of the values listed below. Enter the value exactly as it appears and include underscores, if applicable: ◦ audits - Projects ◦ control_test_plans - Control Test Plans ◦ control_tests - Control Test ◦ controls - Controls ◦ finding_actions - Actions ◦ findings - Issues ◦ mitigations - Risk Control Associations ◦ narratives - Narratives ◦ objectives- Objectives ◦ risks - Risks ◦ walkthroughs - Walkthroughs **Tip** For information about how the tables in Projects are related, and the key fields that you can use to join the tables once you have imported them to Analytics, see Import HighBond Projects data.
FIELD *name* AS *display_name* <...*n*> optional	Individual fields in the source data to import. Specify the name. If you omit FIELD, all fields are imported. ◦ *name* must exactly match the physical field name in the Projects table, including matching the case ◦ *display_name* (alternate column title) is the display name for the field in the view in the new Analytics table. You must specify a display name for each FIELD *name*. Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title. Unlike some other IMPORT commands in Analytics, you cannot specify a blank *display_name* as a way of using the FIELD name as the display name. **Tip** To get the physical field names, use the Analytics user interface to import the appropriate table from Projects. The physical field names are contained in the command in the log. Subsequent imports can be scripted.

Examples

Importing all fields from the Projects table

You import all fields from the **Projects** table for all active projects belonging to organization 286 to an Analytics table named **All_Projects**. You include a numbered password definition to authenticate the connection:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
IMPORT GRCPROJECT TO All_Projects "C:\HighBond Projects Data\All_Projects.fil" PASSWORD 1
FROM "286@us/audits"
```

Importing specified fields from the Projects table

You import specified fields from the **Projects** table for all active projects belonging to organization 286 to an Analytics table named **All_Projects**:

```
IMPORT GRCPROJECT TO All_Projects "C:\HighBond Projects Data\All_Projects.fil" FROM
"286@us/audits" FIELD "id" AS "Id" FIELD "description" AS "Description" FIELD "name" AS "Name"
FIELD "start_date" AS "Start date" FIELD "status" AS "Status" FIELD "created_at" AS "Created at"
```

Importing all fields from the Issues table

You import all fields from the **Issues** table for all active projects belonging to organization 286 to an Analytics table named **All_Issues**:

```
IMPORT GRCPROJECT TO All_Issues "C:\HighBond Projects Data\All_Issues.fil" FROM
"286@us/findings"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Creating a password definition and specifying a password value

PASSWORD command

If you use the PASSWORD command to create the numbered password definition for connecting to HighBond, no password value is specified, so a password prompt is displayed when the script attempts to connect.

For more information, see "PASSWORD command" on page 360.

SET PASSWORD command

If you use the SET PASSWORD command to create the numbered password definition for connecting to HighBond, a password value is specified, so no password prompt is displayed, which is appropriate for scripts designed to run unattended.

For more information, see SET PASSWORD command.

Acquire a HighBond access token

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Regardless of which method you use to create the password definition, the required password value is a HighBond access token, which users can generate in Launchpad.

> **Caution**
>
> The generated access token matches the account used to sign in to Launchpad. As a scriptwriter, specifying your own access token in a script may not be appropriate if the script will be used by other people.

1. Do one of the following:
   - From the Analytics main menu, select **Tools > HighBond Access Token**.
   - In the **Script Editor**, right-click and select **Insert > HighBond Token**.

   The **Manage API tokens** page opens in your browser. You may be required to first sign in to Launchpad.
2. Do one of the following:
   - **Use an existing token** – In the **Token** column, click the partially masked token that you want to use and enter your HighBond account password. The unmasked token is displayed.

     > **Tip**
     >
     > Use an existing token unless you have a reason for creating a new one. If the existing token does not work, create a new one.
     >
     > Using an existing token cuts down on the number of tokens you need to manage.

   - **Create a new token** – Click **Create token > Analytics** and enter your HighBond account password.

     A new Analytics token is created.

     > **Note**
     >
     > If you are a Launchpad System Admin, you also have the option of creating an API token. You should reserve API tokens for their intended purpose, which is programmatic access to the HighBond platform.
3. Click **Copy** to copy the token.

   > **Tip**
   >
   > Do not close the dialog box containing the token until you have successfully pasted the token.
4. In Analytics, do one of the following:
   - paste the token into the password prompt
   - paste the token at the appropriate point in the SET PASSWORD command syntax in a script
5. In Launchpad, close the dialog box containing the token.

   If you created a new token, a partially masked version of the token is added to the top of your list of tokens.

   For more information, see Creating and managing access tokens.

Import debug capability

A simple debug capability exists for imports from HighBond.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

The imported data is temporarily stored in a JSON intermediary file in the folder containing the target Analytics project. In any folder containing an Analytics project you can create a text file that causes the JSON file to be retained, instead of being deleting after the data is imported into Analytics.

- **JSON file is present** - If the import from HighBond is failing, but the JSON file is present on your computer, you know that the problem is on the Analytics side, not on the HighBond side.
- **JSON file is not present** - If the import from HighBond is failing, and the JSON file is not present on your computer, you know that the problem is on the HighBond side.

This information can help with troubleshooting.

Configure retention of the JSON intermediary file

In the folder containing the target Analytics project, create an empty text file with exactly this name: `_grc_import_debug.txt`

When you import from either Results or Projects in HighBond, the JSON intermediary file is retained with the name `results.json`. The file is overwritten with each subsequent import from HighBond.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT GRCRESULTS command

Creates an Analytics table by importing a HighBond Results table or interpretation.

Syntax

IMPORT GRCRESULTS TO *table import_filename* PASSWORD *num* FROM *Results_resource_path* <FIELD *name* AS *display_name* <...*n*>>

Parameters

Name	Description
TO *table*	The name of the Analytics table to import the data into. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
import_filename	The name of the Analytics data file to create. Specify *import_filename* as a quoted string with a .FIL file extension. For example, "Invoices.FIL". By default, the data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ "C:\data\Invoices.FIL" ◦ "data\Invoices.FIL"
PASSWORD *num*	The password definition to use. You do not use PASSWORD *num* to prompt for, or specify, an actual password. The password definition refers to a password previously supplied or set using the PASSWORD command, the SET PASSWORD command, or the PASSWORD analytic tag. *num* is the number of the password definition. For example, if two passwords have been previously supplied or set in a script, or when scheduling an analytic, PASSWORD 2 specifies that password #2 is used. For more information about supplying or setting passwords, see: • "PASSWORD command" on page 360 • "SET command" on page 418 • PASSWORD analytic tag The required password value is a HighBond access token. For more information, see

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>"Creating a password definition and specifying a password value" on page 269.

Note
PASSWORD may or may not be required, depending on the environment in which the script runs:
<table><tr><td>Analytics
(online activation)</td><td>PASSWORD is not required.
The current user's HighBond access token is automatically used.</td></tr><tr><td>Analytics
(offline activation)</td><td rowspan="4">PASSWORD is required.</td></tr><tr><td>Robots</td></tr><tr><td>Analytics Exchange</td></tr><tr><td>Analysis App window</td></tr></table></td></tr>
<tr><td>FROM Results_resource_path</td><td>The path to the data you are importing.

The form of the path varies depending on the data you are importing. For details about the form of the path, see "Results path" on page 267.

Note
The form of the Results path is supplied by an API, and is subject to change. The easiest and most reliable way to acquire the correct and current syntax for the path is to perform a manual import of the target data, and copy the path from the command log.</td></tr>
<tr><td>FIELD name AS display_name <...n>

optional</td><td>Individual fields in the source data to import. Specify the name.

If you omit FIELD, all fields are imported.

Name

name must exactly match the physical field name in the Results table, including matching the case. To view the physical field name, do one of the following:

◦ In Results, click a column header in the Table View. The physical field name appears after Field Name.
◦ In Analytics, when you import a Results table, the physical field name appears in parentheses after the display name in the dialog box that allows you to select fields.

Note
The Results physical field name is not the display name used for column headers in the Table View.

Also see "Field name considerations when importing and exporting Results data" on page 268.

Display name</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	display_name (alternate column title) is the display name for the field in the view in the new Analytics table. You must specify a display name for each FIELD *name*. Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title. Unlike some other IMPORT commands in Analytics, you cannot specify a blank *display_name* as a way of using the FIELD name as the display name.

Examples

Importing specified fields from a table in Results

You import specified fields from a table in Results to an Analytics table named **T and E exceptions**:

```
IMPORT GRCRESULTS TO T_and_E_exceptions "C:\Secondary Analysis\T_and_E_exceptions.fil" PASSWORD 1 FROM "results/api/orgs/11594/control_tests/185699/exceptions" FIELD "metadata.status" AS "Status" FIELD "EmpNo" AS "Employee Number" FIELD "DATE" AS "Date" FIELD "CARDNUM" AS "Card Number" FIELD "CODES" AS "MC Codes" FIELD "AMOUNT" AS "Amount" FIELD "DESCRIPTION" AS "Description"
```

Importing all fields from a table in Results

You import all fields from a table in Results to an Analytics table named **T and E exceptions**:

```
IMPORT GRCRESULTS TO T_and_E_exceptions "C:\Secondary Analysis\T_and_E_exceptions.fil" PASSWORD 1 FROM "results/api/orgs/11594/control_tests/185699/exceptions"
```

Importing data from an interpretation in Results

You import an interpretation in Results to an Analytics table named **T and E exceptions filtered**:

```
IMPORT GRCRESULTS TO T_and_E_exceptions_filtered "C:\Secondary Analysis\T_and_E_exceptions_filtered.fil" FROM "results/api/orgs/11594/control_tests/185699/interpretations/22699/exceptions"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Preserving sort order and filters

When you import data from Results, any data customization, such as sorting or filter, is retained or ignored in the resulting Analytics table depending on how you import the data:

- **import a table** – data customization is ignored. All the data in the table is imported, with the exception of any fields you choose to omit.
- **import an interpretation** – data customization is retained

Results path

> **Note**
>
> The form of the Results path is supplied by an API, and is subject to change. The easiest and most reliable way to acquire the correct and current syntax for the path is to perform a manual import of the target data, and copy the path from the command log.

The Results path in the FROM parameter takes the following general form:

FROM "results <-*region code*>/api/orgs/<*org ID*>/control_tests/<*control test ID*>/exceptions

For example: FROM "results/api/orgs/11594/control_tests/4356/exceptions"

The org ID is displayed in the browser address bar when you log in to Launchpad. The control test ID, and the interpretation ID, are displayed in the address bar when you are viewing those tables in Results.

The table below provides all the variations of the Results path.

To import:	Use this form of Results path:
Control test (table) data	FROM "results/api/orgs/11594/control_tests/4356/exceptions"
Control test (table) audit trail	FROM "results/api/orgs/11594/control_tests/4356/audit_trail"
Control test (table) comments	FROM "results/api/orgs/11594/control_tests/4356/comments"
Interpretation	FROM "results/api/orgs/11594/control_tests/4356/interpretations/1192/exceptions"
Data from a HighBond region other than the default region (us)	◦ **Asia Pacific** - FROM "results-ap/api/orgs/11594/control_tests/4356/exceptions" ◦ **Australia** - FROM "results-au/api/orgs/11594/control_tests/4356/exceptions" ◦ **Canada** - FROM "results-ca/api/orgs/11594/control_tests/4356/exceptions" ◦ **Europe** - FROM "results-eu/api/orgs/11594/control_tests/4356/exceptions"

System-generated information columns

When you import data from Results, you have the option of also importing one or more of the system-generated information columns listed below.

The system-generated columns are either:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- part of Results tables, and contain processing information related to individual records
- additional information – collection name, table name, or record ID number

You must specify the field names of the system-generated columns exactly as they appear below. The default display names apply when you import from Results through the Analytics user interface. You are free to change the display names if you are scripting the import process.

Field name	Default display name
metadata.priority	Priority
metadata.status	Status
metadata.publish_date	Published
metadata.publisher	Publisher Name
metadata.assignee	Assignee
metadata.group	Group
metadata.updated_at	Updated
metadata.closed_at	Closed
extras.collection	Collection
extras.results_table	Results Table
extras.record_id	Record ID

Field name considerations when importing and exporting Results data

If you are round-tripping data between Results and Analytics, you need to ensure that all field names in the Results table meet the more stringent Analytics field name requirements. If you do not, you risk mis-aligning your Analytics and Results data.

For example, any special characters in Results field names are automatically converted to underscores when they are imported into Analytics, which means the field names no longer match the original names in Results. If you then export the Analytics data back to the original table in Results, fields are no longer correctly matched.

To avoid this problem with data that you intend to round-trip, make sure that before you upload the data to Results from CSV or Excel files it meets these Analytics field name requirements:

- no special characters or spaces
- does not start with a number
- contains only alphanumeric characters, or the underscore character (_)

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Creating a password definition and specifying a password value

PASSWORD command

If you use the PASSWORD command to create the numbered password definition for connecting to HighBond, no password value is specified, so a password prompt is displayed when the script attempts to connect.

For more information, see "PASSWORD command" on page 360.

SET PASSWORD command

If you use the SET PASSWORD command to create the numbered password definition for connecting to HighBond, a password value is specified, so no password prompt is displayed, which is appropriate for scripts designed to run unattended.

For more information, see SET PASSWORD command.

Acquire a HighBond access token

Regardless of which method you use to create the password definition, the required password value is a HighBond access token, which users can generate in Launchpad.

> **Caution**
>
> The generated access token matches the account used to sign in to Launchpad. As a scriptwriter, specifying your own access token in a script may not be appropriate if the script will be used by other people.

1. Do one of the following:
   - From the Analytics main menu, select **Tools > HighBond Access Token**.
   - In the **Script Editor**, right-click and select **Insert > HighBond Token**.

   The **Manage API tokens** page opens in your browser. You may be required to first sign in to Launchpad.
2. Do one of the following:
   - **Use an existing token** – In the **Token** column, click the partially masked token that you want to use and enter your HighBond account password. The unmasked token is displayed.

     > **Tip**
     >
     > Use an existing token unless you have a reason for creating a new one. If the existing token does not work, create a new one.
     >
     > Using an existing token cuts down on the number of tokens you need to manage.

   - **Create a new token** – Click **Create token > Analytics** and enter your HighBond account password.

     A new Analytics token is created.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> If you are a Launchpad System Admin, you also have the option of creating an API token. You should reserve API tokens for their intended purpose, which is programmatic access to the HighBond platform.

3. Click **Copy** to copy the token.

   > **Tip**
   >
   > Do not close the dialog box containing the token until you have successfully pasted the token.

4. In Analytics, do one of the following:
   - paste the token into the password prompt
   - paste the token at the appropriate point in the SET PASSWORD command syntax in a script
5. In Launchpad, close the dialog box containing the token.

   If you created a new token, a partially masked version of the token is added to the top of your list of tokens.

   For more information, see Creating and managing access tokens.

Import debug capability

A simple debug capability exists for imports from HighBond.

The imported data is temporarily stored in a JSON intermediary file in the folder containing the target Analytics project. In any folder containing an Analytics project you can create a text file that causes the JSON file to be retained, instead of being deleting after the data is imported into Analytics.

- **JSON file is present** – If the import from HighBond is failing, but the JSON file is present on your computer, you know that the problem is on the Analytics side, not on the HighBond side.
- **JSON file is not present** – If the import from HighBond is failing, and the JSON file is not present on your computer, you know that the problem is on the HighBond side.

This information can help with troubleshooting.

Configure retention of the JSON intermediary file

In the folder containing the target Analytics project, create an empty text file with exactly this name: `_grc_import_debug.txt`

When you import from either Results or Projects in HighBond, the JSON intermediary file is retained with the name `results.json`. The file is overwritten with each subsequent import from HighBond.

Importing large tables

Tables that have a large number of fields may not successfully import using a single IMPORT GRCRESULTS command. If you need to work with a single table containing a large number of fields outside of Results, use one of the following approaches:

- **Split the table** – use two or more IMPORT GRCRESULTS commands to import a subset of fields and then join the resulting tables in Analytics using the JOIN command

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- **Export the table to file** - use the export to CSV format and then import the resulting file into Analytics using the IMPORT DELIMITED command

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT LAYOUT command

Imports an external table layout file (.layout) to an Analytics project.

> **Note:**
>
> Prior to version 11 of Analytics, external table layout files used an .fmt file extension. You can still import a table layout file with an .fmt extension by manually specifying the extension.

Syntax

```
IMPORT LAYOUT external_layout_file TO table_layout_name
```

Parameters

Name	Description
external_layout_file	The name of the external table layout file. If the file name or path includes any spaces it must be enclosed in quotation marks – for example, "Ap Trans.layout". The .layout file extension is used by default and does not need to be specified. If required, you can use another file extension, such as .fmt. If the layout file is not located in the same folder as the Analytics project, you must use an absolute path or a relative path to specify the file location – for example, "C:\Saved layouts\Ap_Trans.layout" or "Saved layouts\Ap_Trans.layout".
TO *table_layout_name*	The name of the imported table layout in the Analytics project – for example, "Ap Trans May". You must specify the *table_layout_name* as a quoted string if it contains any spaces. You can specify a *table_layout_name* that is different from the name of the *external_layout_file*.

Example

Importing an external table layout file

You import an external table layout file called **Ap_Trans.layout** and create a new table layout called **Ap_Trans _May** in the Analytics project:

```
IMPORT LAYOUT "C:\Saved layouts\Ap_Trans.layout" TO "Ap_Trans_May"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

When to use IMPORT LAYOUT

Importing an external table layout file and associating it with a data file can save you the labor of creating a new table layout from scratch:

- If the imported table layout specifies an association with a particular Analytics data file (.fil), and a data file of the same name exists in the folder containing the project, the imported table layout is automatically associated with the data file in the folder.
- If there is no data file with the same name in the project folder, you need to link the imported table layout to a new data source.

Table layouts and source data files must match

The imported table layout and the data file it is associated with must match. The structure of the data in the data file must match the field definitions specified by the table layout metadata.

Data structure refers to the data elements (fields) contained in a data file, the number and order of the fields, and the data type and length of the fields. If the table layout and the data file do not match, jumbled or missing data results.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT MULTIDELIMITED command

Creates multiple Analytics tables by defining and importing multiple delimited files.

Syntax

IMPORT MULTIDELIMITED <TO *import_folder*> FROM {*source_filename*|*source_folder*} *source_char_encoding* SEPARATOR {*char*|TAB|SPACE} QUALIFIER {*char*|NONE} <CONSECUTIVE> STARTLINE *line_number* <KEEPTITLE> <CRCLEAR> <LFCLEAR> <REPLACENULL> <ALLCHAR>

> **Note**
>
> You must specify the IMPORT MULTIDELIMITED parameters in exactly the same order as above, and in the table below.
>
> To import multiple delimited files cleanly, the structure of all the files must be consistent before importing.
>
> For more information, see "Consistent file structure required" on page 279.

Parameters

Name	Description
TO *import_folder* optional	The folder to import the data into. To specify the folder, use an absolute file path, or a file path relative to the folder containing the Analytics project. Specify *import_folder* as a quoted string. **Example** TO "C:\Point of sale audit\Data\Transaction working data" TO "Data\Transaction working data" If you omit TO , the data is imported to the folder containing the Analytics project.
FROM *source_filename* | *source_folder*	The name of the source data files, or the folder containing the source data files. Specify *source_filename* or *source_folder* as a quoted string. The command supports importing four types of delimited file:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description

	◦ `*.csv`
	◦ `*.dat`
	◦ `*.del`
	◦ `*.txt`
	
	Source data files in the root Analytics project folder
	
	To specify multiple files, use a wildcard character (*) in place of unique characters in file names. The wildcard character stands for zero (0) or more occurrences of any letter, number, or special character.
	
	Example
	
	`FROM "Transactions_FY*.csv"`
	
	selects:
	
	`Transactions_FY18.csv`
	
	`Transactions_FY17.csv`
	
	You can use a wildcard in more than one location in a file name, and in a file extension.
	
	Example
	
	`FROM "Transactions_FY*.*"`
	
	selects:
	
	`Transactions_FY18.txt`
	
	`Transactions_FY17.csv`
	
	Source data files not in the root Analytics project folder
	
	If the source data files are not located in the same folder as the Analytics project, you must use an absolute file path, or a file path relative to the folder containing the project, to specify the location of the files.
	
	Example
	
	`FROM "C:\Point of sale audit\Data\Transaction master files\Transactions_FY*.csv"`
	
	`FROM "Data\Transaction master files\Transactions_FY*.csv"`
	
	Folder containing source data files
	
	Instead of specifying file names, you can just specify the name of the folder containing source data files. All supported delimited files in the folder are imported (`*.csv`, `*.dat`, `*.del`, `*.txt`).
	
	To specify a source data folder, use an absolute file path, or a file path relative to the folder containing the Analytics project.
	
	Example

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>
<pre>FROM "C:\Point of sale audit\Data\Transaction master files"</pre>
<pre>FROM "Data\Transaction master files"</pre>
</td></tr>
<tr><td>source_char_encoding</td><td>
The character set and encoding of the source data.

Depending on which edition of Analytics you are using, and the encoding of the source data, specify the appropriate code:
<table>
<tr><th>Code</th><th>Analytics edition</th><th>Source data encoding</th></tr>
<tr><td>0</td><td>Non-Unicode edition</td><td>all data</td></tr>
<tr><td>0</td><td>Unicode edition</td><td>ASCII data</td></tr>
<tr><td>2</td><td>Unicode edition</td><td>Unicode data, UTF-16 LE encoding</td></tr>
<tr><td>3 numeric_code</td><td>Unicode edition</td><td>Unicode data that does not use UTF-16 LE encoding

To determine the numeric code that matches the source data encoding, perform an import using the Data Definition Wizard, select the Encoded Text option, and find the matching encoding in the accompanying drop-down list.

To specify the code, specify 3, followed by a space, and then the numeric code.</td></tr>
</table>

Note
If you do not specify a code, Non-Unicode Analytics automatically uses 0 , and Unicode Analytics automatically uses 2 .
</td></tr>
<tr><td>SEPARATOR char | TAB | SPACE</td><td>
The separator character (delimiter) used between fields in the source data. You must specify the character as a quoted string.

You can specify a tab or a space separator by typing the character between double quotation marks, or by using a keyword:
◦ SEPARATOR "	" or SEPARATOR TAB
◦ SEPARATOR " " or SEPARATOR SPACE
</td></tr>
<tr><td>QUALIFIER char | NONE</td><td>
The text qualifier character used in the source data to wrap and identify field values. You must specify the character as a quoted string.

To specify the double quotation mark character as the text qualifier, enclose the character in single quotation marks: QUALIFIER "".

You can specify that there are no text qualifiers using either of these methods:
◦ QUALIFIER ""
◦ QUALIFIER NONE
</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
CONSECUTIVE optional	Consecutive text qualifiers are treated as a single qualifier.
STARTLINE *line_number*	The line the data begins on. For example, if the first four lines of data contain header information that you do not want, specify 5 for *line_number*. **Note** Ideally, the start line of the data should be the same in all the delimited files that you import with a single execution of IMPORT MULTIDELIMITED. If start lines are different, see "Consistent file structure required" on page 279.
KEEPTITLE optional	Treat the line number specified by STARTLINE as field names instead of data. If you omit KEEPTITLE, generic field names are used. **Note** The field names must be on the same line number in all the delimited files that you import with a single execution of IMPORT MULTIDELIMITED. If field names are on different line numbers, see "Consistent file structure required" on page 279.
CRCLEAR optional	Replaces any CR characters (carriage return) that occur between text qualifiers with space characters. You must specify QUALIFIER with a *char* value to use CRCLEAR. If you use both CRCLEAR and LFCLEAR, CRCLEAR must come first.
LFCLEAR optional	Replaces any LF characters (line feed) that occur between text qualifiers with space characters. You must specify QUALIFIER with a *char* value to use LFCLEAR. If you use both CRCLEAR and LFCLEAR, CRCLEAR must come first.
REPLACENULL optional	Replaces any NUL characters that occur in the delimited file with space characters. The number of any replaced NUL characters is recorded in the log.
ALLCHAR optional	The Character data type is automatically assigned to all the imported fields. **Tip** Assigning the Character data type to all the imported fields simplifies the process of importing delimited text files. Once the data is in Analytics, you can assign different data types, such as Numeric or Datetime, to the fields, and specify format details. ALLCHAR is useful if you are importing a table with identifier fields automatically assigned the Numeric data type by Analytics when in fact they should use the Character data type.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

The examples below assume that you have monthly transaction data stored in 12 delimited files:

- `Transactions_Jan.csv` to `Transactions_Dec.csv`

  > **Note**
  >
  > A separate Analytics table is created for each delimited file that you import.

Import all the delimited files

You want to import all 12 delimited files. You use the wildcard symbol (*) where the month occurs in each file name.

Analytics attempts to assign the appropriate data type to each field.

```
IMPORT MULTIDELIMITED FROM "Transactions_*.csv" 0 SEPARATOR "," QUALIFIER ""
CONSECUTIVE STARTLINE 1 KEEPTITLE
```

Import all the delimited files as character data

This example is the same as the one above, except Analytics automatically assigns the Character data type to all the imported fields.

```
IMPORT MULTIDELIMITED FROM "Transactions_*.csv" 0 SEPARATOR "," QUALIFIER ""
CONSECUTIVE STARTLINE 1 KEEPTITLE ALLCHAR
```

Import all the delimited files from the specified folder

You want to import all the delimited files in the `C:\Point of sale audit\Data\Transaction master files` folder.

```
IMPORT MULTIDELIMITED FROM "C:\Point of sale audit\Data\Transaction master files" 0
SEPARATOR "," QUALIFIER "" CONSECUTIVE STARTLINE 1 KEEPTITLE
```

Import all the delimited files from the specified folder, and save the Analytics tables to another folder

This example is the same as the one above, but instead of saving the Analytics tables in the root project folder, you want to save them in the `C:\Point of sale audit\Data\Transaction working data` folder.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
IMPORT MULTIDELIMITED TO "C:\Point of sale audit\Data\Transaction working data" FROM
"C:\Point of sale audit\Data\Transaction master files" 0 SEPARATOR "," QUALIFIER ""
CONSECUTIVE STARTLINE 1 KEEPTITLE
```

Remarks

Consistent file structure required

To import a group of delimited files cleanly using IMPORT MULTIDELIMITED, the structure of all the files in the group must be consistent.

You can import inconsistently structured delimited files, and subsequently perform data cleansing and standardizing in Analytics. However, this approach can be labor intensive. In many cases, it is easier to make the delimited files consistent before importing.

To import multiple delimited files cleanly, the following items need to be consistent across all files:

Item	ACLScript keyword	Problem	Solution
The character set and encoding of the source data	*numeric code*	(Unicode edition of Analytics only) Source delimited files use different character encodings. For example, some files have ASCII encoding and some files have Unicode encoding.	Group source files by encoding type, and do a separate import for each group.
Delimiter character	SEPARATOR	Source delimited files use a different separator character (delimiter) between fields.	Do one of the following: ◦ Standardize the separator character in the source files before importing them. ◦ Group source files by separator character, and do a separate import for each group.
Text qualifier character	QUALIFIER	Source delimited files use a different text qualifier character to wrap and identify field values.	Do one of the following: ◦ Standardize the qualifier character in the source files before importing them. ◦ Group source files by qualifier character, and do a separate import for each group.
Start line of the data	STARTLINE	Source delimited files have different start lines for the data.	Do one of the following: ◦ Standardize the start line in the source files before importing them. ◦ Group source files that have the same start line, and do a separate import for each group.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Item	ACLScript keyword	Problem	Solution
			◦ Make *line_number* equal to the lowest start line across all the files. Once the files have been imported to Analytics tables, you can use the "EXTRACT command" on page 203 to extract only the records from any table with unwanted header information.
Field names	KEEPTITLE	Source delimited files have field names on different line numbers.	Do one of the following: ◦ Standardize the line number with the field names in the source files before importing them. ◦ Group source files that have field names on the same line number, and do a separate import for each group.
Field names	KEEPTITLE	Some source delimited files have field names and some do not.	Do one of the following: ◦ Add field names to the source files that require them before importing all files. ◦ Group source files that have field names, and files that do not have field names, and do a separate import for each group. ◦ Omit KEEPTITLE to import all files using generic field names. Once the files have been imported to Analytics tables, you can use the "EXTRACT command" on page 203 to extract only the data you want from any table.

Multiple IMPORT DELIMITED commands

The IMPORT MULTIDELIMITED command actually performs multiple individual IMPORT DELIMITED commands – one for each file imported. If you double-click the IMPORT MULTIDELIMITED entry in the log, the individual IMPORT DELIMITED commands are displayed in the display area.

Combining multiple delimited files after importing them

After you import multiple delimited files into individual Analytics tables you might want to combine them into a single Analytics table. For example, you could combine the data from twelve monthly tables into a single annual table containing all the data.

For information about combining multiple Analytics tables, see "APPEND command" on page 73.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT MULTIEXCEL command

Creates multiple Analytics tables by defining and importing multiple Microsoft Excel worksheets or named ranges.

Syntax

IMPORT MULTIEXCEL <TO *import_folder*> FROM {*source_filename*|*source_folder*} TABLE *input_worksheets_or_named_ranges* <PREFIX> <KEEPTITLE> <CHARMAX *max_field_length*>

> **Note**
>
> You must specify the IMPORT MULTIEXCEL parameters in exactly the same order as above, and in the table below.
>
> Analytics cannot import from an Excel workbook if Protected View is active for the workbook. You must first enable editing in the workbook, save and close the workbook, and then perform the import.

Parameters

Name	Description
TO *import_folder* optional	The folder to import the data into. To specify the folder, use an absolute file path, or a file path relative to the folder containing the Analytics project. Specify *import_folder* as a quoted string. **Example** TO "C:\Point of sale audit\Data\Transaction working data" TO "Data\Transaction working data" If you omit TO, the data is imported to the folder containing the Analytics project.
FROM *source_filename* | *source_folder*	The name of the source data file or files, or the folder containing the source data file or files. Specify *source_filename* or *source_folder* as a quoted string. **Source data file or files in the root Analytics project folder** ◦ **single Excel file** Specify the compete file name and extension.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	### Example FROM "Transactions_FY18.xlsx" ◦ **multiple Excel files** To specify multiple files, use a wildcard character (*) in place of unique characters in file names. The wildcard character stands for zero (0) or more occurrences of any letter, number, or special character. ### Example FROM "Transactions_FY*.xlsx" selects: Transactions_FY18.xlsx Transactions_FY17.xlsx You can use a wildcard in more than one location in a file name, and in a file extension. ### Example FROM "Transactions_FY*.*" selects: Transactions_FY18.xlsx Transactions_FY17.xls ## Source data file or files not in the root Analytics project folder If the source data file or files are not located in the same folder as the Analytics project, you must use an absolute file path, or a file path relative to the folder containing the project, to specify the file location. ### Example FROM "C:\Point of sale audit\Data\Transaction master files\Transactions_FY18.xlsx" FROM "Data\Transaction master files\Transactions_FY*.xlsx" ## Folder containing source data file or files Instead of specifying a file name, you can just specify the name of the folder containing a source data file or files. To specify a source data folder, use an absolute file path, or a file path relative to the folder containing the Analytics project. ### Example

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	FROM "C:\Point of sale audit\Data\Transaction master files" FROM "Data\Transaction master files" **Note** When you specify a folder, any worksheet in any Excel file in the folder is imported if the worksheet name matches the TABLE value.
TABLE *input_worksheets_or_named_ranges*	The name of the worksheets or named ranges to import. A separate Analytics table is created for each imported worksheet or named range. Specify *input_worksheets_or_named_ranges* as a quoted string. Use a wildcard (*) in place of unique characters in the names of worksheets or ranges. For example, `"Trans_*$"` selects the following worksheets: ◦ Trans_Jan ◦ Trans_Feb ◦ Trans_Mar ◦ and so on **Note** The wildcard character (*) stands for zero (0) or more occurrences of any letter, number, or special character. You can use a wildcard in more than one location. For example, `*Trans*$` selects: • `Trans_Jan` • `Jan_Trans` **The meaning of the dollar sign ($)** In an Excel file, worksheets are identified with a dollar sign ($) appended to the worksheet name (`Trans_Jan$`). The dollar sign is not visible in Excel. Named ranges are identified by the absence of a dollar sign (`Trans_Jan_commercial`). Specifying the dollar sign is not required when using IMPORT MULTIEXCEL. However, you should include it, or exclude it, in the following situations: ◦ **Include "$"** - if you want to import only worksheets, and no named ranges, include the dollar sign at the end of the worksheet name ◦ **Exclude "$"** - if you want to import named ranges, or worksheets and named ranges in a single import operation, do not include the dollar sign
PREFIX optional	Prepend the Excel file name to the name of the Analytics tables. **Tip** If worksheets in different files have the same name, prepending the Excel file name allows you to avoid table name conflicts.
KEEPTITLE optional	Treat the first row of data as field names instead of data. If omitted, generic field names are used.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note All first rows in the worksheets and named ranges that you import should use a consistent approach. First rows should be either field names, or data, across all data sets. Avoid mixing the two approaches in a single import operation. If the data sets have an inconsistent approach to first rows, use two separate import operations.
CHARMAX *max_field_length* optional	The maximum length in characters for any field in an Analytics table that originates as character data in a source data file.

Examples

The examples below assume that you have monthly transaction data for three years stored in three Excel files:

- `Transactions_FY18.xlsx`
- `Transactions_FY17.xlsx`
- `Transactions_FY16.xlsx`

Each Excel file has 12 worksheets – one for each month of the year. The worksheets also include some named ranges identifying various subsets of transactions.

> **Note**
>
> A separate Analytics table is created for each worksheet or named range that you import.

Import worksheets

Import all FY18 worksheets

You want to import all 12 monthly worksheets from the FY18 Excel file, and ignore any named ranges.

- you use the wildcard symbol (*) where the month occurs in each worksheet name
- you include the dollar sign ($) at the end of the worksheet name so that only worksheets are selected, and no named ranges

```
IMPORT MULTIEXCEL FROM "Transactions_FY18.xlsx" TABLE "Trans_*$"
```

Import all FY18 worksheets, keep field names, and specify maximum character field length

This example is the same as the one above, but you want to keep the field names from the Excel files, and

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

also limit the length of character fields.

- you include KEEPTITLE to use the first row of Excel data as the field names
- you include CHARMAX 50 so that fields that originate as character data in the Excel file are limited to 50 characters in the resulting Analytics table

```
IMPORT MULTIEXCEL FROM "Transactions_FY18.xlsx" TABLE "Trans_*$" KEEPTITLE
CHARMAX 50
```

Import all worksheets from all three files

You want to import all 36 monthly worksheets from the three Excel files, and ignore any named ranges.

- you use the wildcard symbol (*) where the month occurs in each worksheet name
- you include the dollar sign ($) at the end of the worksheet name so that only worksheets are selected, and no named ranges
- you use the wildcard symbol (*) where the year occurs in each Excel file name
- as a way of reducing the chance of naming conflicts, you use PREFIX to prepend the name of the source Excel file to each Analytics table name

```
IMPORT MULTIEXCEL FROM "Transactions_FY*.xlsx" TABLE "Trans_*$" PREFIX
```

Import named ranges

Import all FY18 "Commercial_transaction" named ranges

You want to import all "Commercial_transaction" named ranges from the FY18 Excel file, and ignore worksheets, and other named ranges.

- you use the wildcard symbol (*) where a unique identifier occurs in the names of the different ranges
- you exclude the dollar sign ($) so that named ranges can be selected

```
IMPORT MULTIEXCEL FROM "Transactions_FY18.xlsx" TABLE "Commercial_transaction_*"
```

Import worksheets and named ranges

Import all FY18 worksheets and named ranges

You want to import all 12 monthly worksheets, and all named ranges, from the FY18 Excel file.

- with TABLE, you use only the wildcard symbol (*) so that all worksheets and named ranges in the file are selected
- you exclude the dollar sign ($) so that named ranges can be selected

```
IMPORT MULTIEXCEL FROM "Transactions_FY18.xlsx" TABLE "*"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Manage directories

Import all worksheets from all Excel files in the specified folder

You want to import all worksheets from all Excel files in the `C:\Point of sale audit\Data\Transaction master files` folder.

- with TABLE, you use only the wildcard symbol (*) so that all worksheets in each file are selected, and the dollar sign ($) so that only worksheets are selected, and no named ranges
- as a way of reducing the chance of naming conflicts, you use PREFIX to prepend the name of the source Excel file to each Analytics table name

```
IMPORT MULTIEXCEL FROM "C:\Point of sale audit\Data\Transaction master files" TABLE "*$" PREFIX
```

Import all worksheets from all Excel files in the specified folder, and save the Analytics tables to another folder

This example is the same as the one above, but instead of saving the Analytics tables in the root project folder, you want to save them in the `C:\Point of sale audit\Data\Transaction working data` folder.

```
IMPORT MULTIEXCEL TO "C:\Point of sale audit\Data\Transaction working data" FROM "C:\Point of sale audit\Data\Transaction master files" TABLE "*$" PREFIX
```

Remarks

Multiple IMPORT EXCEL commands

The IMPORT MULTIEXCEL command actually performs multiple individual IMPORT EXCEL commands - one for each worksheet imported. If you double-click the IMPORT MULTIEXCEL entry in the log, the individual IMPORT EXCEL commands are displayed in the display area.

Last table imported is automatically opened

IMPORT MULTIEXCEL does not support the OPEN keyword. However, after the command executes, the last table imported is automatically opened.

Combining multiple worksheets after importing them

After you import multiple worksheets into individual Analytics tables you might want to combine them into a single Analytics table. For example, you could combine the data from twelve monthly tables into a single annual table containing all the data.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

For information about combining multiple Analytics tables, see "APPEND command" on page 73.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT ODBC command

Creates an Analytics table by defining and importing data from an ODBC data source.

ODBC stands for Open Database Connectivity, a standard method for accessing databases.

Syntax

IMPORT ODBC SOURCE *source_name* TABLE *table_name* <QUALIFIER *data_qualifier*> <OWNER *user_name*> <USERID *user_id*> <PASSWORD *num*> <WHERE *where_clause*> <TO *table_name*> <WIDTH *max_field_length*> <MAXIMUM *max_field_length*> <FIELDS *field* <,...*n*>>

Parameters

Name	Description
SOURCE *source_name*	The data source name (DSN) of the ODBC data source to connect to. The DSN must already exist and be correctly configured. **Note** You are limited to data sources that use the Windows ODBC drivers that are installed on your computer. The Analytics native data connectors that can be used with the ACCESSDATA command may not be available with IMPORT ODBC.
TABLE *table_name*	The table name in the ODBC data source to import data from. *table_name* usually refers to a database table in the source data, but it can refer to anything Analytics imports as a table. For example, if you use the Microsoft Text Driver, *table_name* refers to the text file you want to import data from.
QUALIFIER *data_qualifier* optional	The character to use as the text qualifier to wrap and identify field values. You must specify the character as a quoted string. Use single quotation marks to specify the double quotation character: `'"'`.
OWNER *user_name* optional	The name of the database user account that owns the table you are connecting to.
USERID *user_id* optional	The username to access the data source.
PASSWORD *num* optional	The password definition to use. You do not use PASSWORD *num* to prompt for, or specify, an actual password. The

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	password definition refers to a password previously supplied or set using the PASSWORD command, the SET PASSWORD command, or the PASSWORD analytic tag. *num* is the number of the password definition. For example, if two passwords have been previously supplied or set in a script, or when scheduling an analytic, PASSWORD 2 specifies that password #2 is used. For more information about supplying or setting passwords, see: • "PASSWORD command" on page 360 • "SET command" on page 418 • PASSWORD analytic tag
WHERE *where_clause* optional	An SQL WHERE clause that limits the records returned based on a criteria you specify. Must be a valid SQL statement and must be entered as a quoted string: `WHERE "SALARY > 50000".`
TO *table_name* optional	The name of the Analytics data file to create. Specify *table_name* as a quoted string with a .FIL file extension. For example, `TO "Invoices.FIL".` By default, the data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ `TO "C:\data\Invoices.FIL"` ◦ `TO "data\Invoices.FIL".`
WIDTH *max_field_length* optional	The maximum length in characters for any field in the Analytics table that originates as character data in the source from which you are importing. You can enter any value between 1 and 254. The default value is 50. Data that exceeds the maximum field length is truncated when imported to Analytics.
MAXIMUM *max_field_length* optional	The maximum length in characters for text, note, or memo fields you are importing. You can enter any value between 1 and 1100. The default value is 100. Data that exceeds the maximum field length is truncated when imported to Analytics.
FIELDS *field* <,...*n*> optional	Individual fields in the source data to import. Specify the name. If you specify multiple fields, each field must be separated by a comma. If you omit FIELDS, all fields are imported. Enclosing the field names in quotation marks makes them case-sensitive. If you use quotation marks, the case of field names must exactly match between FIELDS and the ODBC data source. If you use quotation marks and the case of the field names does not match, the fields are not imported. **Note** FIELDS must be positioned last among the IMPORT ODBC parameters. If FIELDS is not positioned last, the command fails.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Importing data from SQL Server

You import data from a SQL Server database to an Analytics table named **Trans_Dec11**:

```
IMPORT ODBC SOURCE "SQLServerAudit" TABLE "Transactions" OWNER "audit" TO "C:\ACL
DATA\Trans_Dec11.FIL" WIDTH 100 MAXIMUM 200 FIELDS
"CARDNUM","CREDLIM","CUSTNO","PASTDUEAMT"
```

Remarks

Older method of connecting to ODBC data sources

The IMPORT ODBC command is the older method of connecting to ODBC-compliant data sources from Analytics. The new method of connecting to ODBC data sources uses the Data Access window and the ACCESSDATA command.

You can continue to use IMPORT ODBC in Analytics. However, this method of connecting is now available only in scripts and from the Analytics command line. You can no longer access this connection method in the **Data Definition Wizard**.

Suppress the time portion of datetime values

When using the IMPORT ODBC command to define an Analytics table you can suppress the time portion of datetime values by prefacing the command with the SET SUPPRESSTIME ON command.

This capability allows retrofitting Analytics scripts written prior to version 10.0 of Analytics, when the time portion of datetime values was automatically truncated. If SET SUPPRESSTIME ON is not added to these scripts, they do not run in the datetime-enabled version of Analytics.

For more information, see the "SET SUPPRESSTIME" section in "SET command" on page 418.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT PDF command

Creates an Analytics table by defining and importing an Adobe PDF file.

Syntax

IMPORT PDF TO *table* <PASSWORD *num*> *import_filename* FROM *source_filename* <SERVER *profile_name*> *skip_length* <PARSER "VPDF"> <PAGES *page_range*> {[record_syntax] [field_syntax] <...*n*>} <...*n*>

record_syntax ::=
RECORD *record_name record_type lines_in_record transparent* [test_syntax] <...*n*>

test_syntax ::=
TEST *include_exclude match_type* AT *start_line,start_position,range logic text*

field_syntax ::=
FIELD *name type* AT *start_line,start_position* SIZE *length,lines_in_field* DEC *value* WID *bytes* PIC *format* AS *display_name*

Parameters

General parameters

Name	Description
TO *table*	The name of the Analytics table to import the data into. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
PASSWORD *num* optional	Used for password-protected PDF files. The password definition to use. You do not use PASSWORD *num* to prompt for, or specify, an actual password. The password definition refers to a password previously supplied or set using the PASSWORD command, the SET PASSWORD command, or the PASSWORD analytic tag.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	num is the number of the password definition. For example, if two passwords have been previously supplied or set in a script, or when scheduling an analytic, PASSWORD 2 specifies that password #2 is used. For more information about supplying or setting passwords, see: • "PASSWORD command" on page 360 • "SET command" on page 418 • PASSWORD analytic tag
import_filename	The name of the Analytics data file to create. Specify *import_filename* as a quoted string with a .FIL file extension. For example, "Invoices.FIL". By default, the data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ "C:\data\Invoices.FIL" ◦ "data\Invoices.FIL"
FROM *source_filename*	The name of the source data file. *source_filename* must be a quoted string. If the source data file is not located in the same directory as the Analytics project, you must use an absolute path or a relative path to specify the file location: ◦ "C:\data*source_filename*" ◦ "data*source_filename*"
SERVER *profile_name* optional	The profile name for the server that contains the data that you want to import.
skip_length optional	The number of bytes to skip at the start of the file. For example, if the first 32 bytes contains header information, specify a skip length value of 32 to omit this information. **Note** For Unicode data, specify an even number of bytes only. Specifying an odd number of bytes can cause problems with subsequent processing of the imported data.
PARSER "VPDF" optional	Use the VeryPDF parser to parse the PDF file during the file definition process. If you omit PARSER, the default Xpdf parser is used. If you are importing the PDF file for the first time, and you have no reason to do otherwise, use the default Xpdf parser. If you have already encountered data alignment issues when using Xpdf, use the VeryPDF parser to see if the parsing results are better.
PAGES *page_range* optional	The pages to include if you do not want to import all of the pages in the PDF file. *page_range* must be specified as a quoted string. You can specify: ◦ individual pages separated by commas (1,3,5)

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ page ranges (2-7) ◦ a combination of pages and ranges (1, 3, 5-7, 11) If you omit PAGES, all pages in the PDF file are imported.

RECORD parameter

General record definition information.

> **Note**
>
> Some of the record definition information is specified using numeric codes that map to options in the Data Definition Wizard.
>
> **In scripts, specify the numeric code, not the option name.**

Name	Description
RECORD *record_name*	The name of the record in the Data Definition Wizard. Specifying *record_name* is required in the IMPORT PDF command, but the *record_name* value does not appear in the resulting Analytics table. In the Data Definition Wizard, Analytics provides default names based on the type of record: ◦ Detail ◦ Header*n* ◦ Footer*n* You can use the default names, or specify different names.
record_type	The three possible record types when defining a PDF file: ◦ 0 - detail ◦ 1 - header ◦ 2 - footer > **Note** > You can define multiple sets of header and footer records in a single execution of IMPORT PDF, but only one set of detail records.
lines_in_record	The number of lines occupied by a record in the PDF file. You can define single-line or multiline records to match the data in the PDF file.
transparent	The transparency setting for a header record. > **Note** > Applies to header records only. ◦ 0 - not transparent ◦ 1 - transparent Transparent header records do not split multiline detail records.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	If a header record splits a multiline detail record in the source PDF file, which can happen at a page break, specifying 1 (transparent) unifies the detail record in the resulting Analytics table.

TEST parameter

The criteria for defining a set of records in the PDF file. You can have one or more occurrences of TEST (up to 8) for each occurrence of RECORD.

> **Note**
>
> Some of the criteria are specified using numeric codes that map to options in the Data Definition Wizard (option names are shown in parentheses below).
>
> **In scripts, specify the numeric code, not the option name.**

Name	Description
TEST *include_exclude*	How to treat matching data: ◦ 0 - (**Include**) data meeting the criteria is included in the set of records ◦ 1 - (**Exclude**) data meeting the criteria is excluded from the set of records
match_type	The type of matching to perform: ◦ 0 - (**Exact Match**) matching records must contain the specified character, or string of characters, in the specified start line, starting at the specified position ◦ 2 - (**Alpha**) matching records must contain one or more alpha characters, in the specified start line, at the specified start position, or in all positions of the specified range ◦ 3 - (**Numeric**) matching records must contain one or more numeric characters, in the specified start line, at the specified start position, or in all positions of the specified range ◦ 4 - (**Blank**) matching records must contain one or more blank spaces, in the specified start line, at the specified start position, or in all positions of the specified range ◦ 5 - (**Non-Blank**) matching records must contain one or more non-blank characters (includes special characters), in the specified start line, at the specified start position, or in all positions of the specified range ◦ 7 - (**Find in Line**) matching records must contain the specified character, or string of characters, anywhere in the specified start line ◦ 8 - (**Find in Range**) matching records must contain the specified character, or string of characters, in the specified start line, anywhere in the specified range ◦ 10 - (**Custom Map**) matching records must contain characters that match the specified character pattern, in the specified start line, starting at the specified position
AT *start_line*, *start_position*, *range*	◦ ***start_line*** - the line of a record that the criteria apply to For example, if you create a custom map to match zip codes, and the zip codes appear on the third line of a three-line address record, you must specify 3 in *start_line*. > **Note** > For single-line records, the *start_line* value is always 1.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ ***start_position*** - the starting byte position in the PDF file for the comparison against the criteria ◦ ***range*** - the number of bytes from the starting byte position in the PDF file to use in the comparison against the criteria If you are using starting byte position only, without a range, specify `0` for *range*. **Note** non-Unicode Analytics: 1 byte = 1 character Unicode Analytics: 2 bytes = 1 character
logic	The logical relations between criteria: ◦ `0` - (**And**) the current and the next criteria are related with a logical AND ◦ `1` - (**Or**) the current and the next criteria are related with a logical OR ◦ `4` - (**New Group** > **And**) the current criterion is the last in a group of logical criteria, and the current group and the next group are related with a logical AND ◦ `5` - (**New Group** > **Or**) the current criterion is the last in a group of logical criteria, and the current group and the next group are related with a logical OR ◦ `7` - (**End**) the current criterion is the last in a group of logical criteria
text	Literal or wildcard characters to match against: ◦ **For Exact Match, Find in Line, or Find in Range** - specifies the character, or string of characters, that uniquely identifies the set of records in the PDF file ◦ **For Custom Map** - specifies the character pattern that uniquely identifies the set of records in the PDF file The **Custom Map** option uses the same syntax as the "MAP( ) function" on page 643. For other match types, *text* is an empty string `""`.

FIELD parameters

Field definition information.

Name	Description
FIELD *name type*	The individual fields to import from the source data file, including the name and data type of the field. To exclude a field from being imported, do not specify it. For information about *type*, see "Identifiers for field data types" on page 297.
AT *start_line*, *start_position*	◦ ***start_line*** - the start line of the field in the record in the PDF file For multiline records in a PDF file, *start_line* allows you to start a field at any line of the record. *start_line* is always `1` if *lines_in_record* is `1`. ◦ ***start_position*** - the starting byte position of the field in the PDF file

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>

Note

non-Unicode Analytics	1 byte = 1 character
Unicode Analytics	2 bytes = 1 character

In Unicode Analytics, typically you should specify an odd-numbered starting byte position. Specifying an even-numbered starting position can cause characters to display incorrectly.
</td></tr>
<tr><td>SIZE length, lines_in_field</td><td>

- ***length*** - the length in bytes of the field in the Analytics table layout

Note

non-Unicode Analytics	1 byte = 1 character
Unicode Analytics	2 bytes = 1 character

In Unicode Analytics, specify an even number of bytes only. Specifying an odd number of bytes can cause characters to display incorrectly.

- ***lines_in_field*** - the number of lines occupied by a single field value in the PDF file

You can define single-line or multiline fields to match the data in the file.

Note

The number of lines specified for a field cannot exceed the number of lines specified for the record containing the field.
</td></tr>
<tr><td>DEC value</td><td>The number of decimals for numeric fields.</td></tr>
<tr><td>WID bytes</td><td>The display width of the field in bytes.

The specified value controls the display width of the field in Analytics views and reports. The display width never alters data, however it can hide data if it is shorter than the field length.</td></tr>
<tr><td>PIC format</td><td>

Note

Applies to numeric or datetime fields only.

- **numeric fields** - the display format of numeric values in Analytics views and reports
- **datetime fields** - the physical format of datetime values in the source data (order of date and time characters, separators, and so on)

Note

For datetime fields, *format* must exactly match the physical format in the source data. For example, if the source data is 12/31/2014, you must enter the format as "MM/DD/YYYY".

format must be enclosed in quotation marks.
</td></tr>
<tr><td>AS display_name</td><td>The display name (alternate column title) for the field in the view in the new Analytics table.

Specify display_name as a quoted string. Use a semi-colon (;) between words if you want</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	a line break in the column title. AS is required when you are defining FIELD. To make the display name the same as the field name, enter a blank *display_name* value using the following syntax: AS "". Make sure there is no space between the two double quotation marks.

Examples

Importing data from a specific page of a PDF file

You import data from page 1 of the password-protected PDF file, `Vendors.pdf`.

One set of detail records, with three fields, is created in the resulting Analytics table, **Vendor_List**:

```
IMPORT PDF TO Vendor_List PASSWORD 1 "Vendor_List.FIL" FROM "Vendors.pdf" 2 PAGES "1"
RECORD "Detail" 0 1 0 TEST 0 3 AT 1,1,0 7 "" FIELD "Vendor_Number" C AT 1,1 SIZE 10,1 DEC 0
WID 10  PIC "" AS "" FIELD "Vendor_Name" C AT 1,33 SIZE 58,1 DEC 0 WID 58  PIC "" AS "" FIELD
"Last_Active_Date" D AT 1,277 SIZE 20,1 DEC 0 WID 20  PIC "DD/MM/YYYY" AS ""
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Troubleshooting PDF imports in the Unicode edition of Analytics

If you encounter issues when you import a PDF file using the Unicode edition of Analytics, the problem may be related to length specifications:

- If foreign language characters are appearing unexpectedly, or the layout in the resulting Analytics table is skewed, check that SIZE *length* is set to an even number.

  Specifying an odd number of bytes for SIZE *length* can cause problems with processing of the imported data.
- If the Analytics table is created, but contains zero records, trying setting *skip_length* to 2, or some other even number if there is header data at the beginning of the file that you want to skip.

Identifiers for field data types

The table below lists the letters that you must use when specifying *type* for FIELD. Each letter corresponds to an Analytics data type.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

For example, if you are defining a Last Name field, which requires a character data type, you would specify "C": `FIELD "Last_Name" C`.

For more information, see Analytics data types.

> **Note**
>
> When you use the **Data Definition Wizard** to define a table that includes EBCDIC, Unicode, or ASCII fields, the fields are automatically assigned the letter "C" (for the CHARACTER type).
>
> When you enter an IMPORT statement manually, or edit an existing IMPORT statement, you can substitute the more specific letters "E" or "U" for EBCDIC or Unicode fields.

Letter	Analytics Data type
A	ACL
B	BINARY
C	CHARACTER
D	DATETIME
E	EBCDIC
F	FLOAT
G	ACCPAC
I	IBMFLOAT
K	UNSIGNED
L	LOGICAL
N	PRINT
P	PACKED
Q	BASIC
R	MICRO
S	CUSTOM
T	PCASCII
U	UNICODE
V	VAXFLOAT

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Letter	Analytics Data type
X	NUMERIC
Y	UNISYS
Z	ZONED

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT PRINT command

Creates an Analytics table by defining and importing a Print Image (Report) file.

Syntax

IMPORT PRINT TO *table import_filename* FROM *source_filename* <SERVER *profile_name*> *character_set_value* <*code_page_number*> {[record_syntax] [field_syntax] <...*n*>} <...*n*>

record_syntax ::=
RECORD *record_name record_type lines_in_record transparent* [test_syntax] <...*n*>

test_syntax ::=
TEST *include_exclude match_type* AT *start_line*,*start_position*,*range logic text*

field_syntax ::=
FIELD *name type* AT *start_line*,*start_position* SIZE *length*,*lines_in_field* DEC *value* WID *bytes* PIC *format* AS *display_name*

Parameters

General parameters

Name	Description
TO *table*	The name of the Analytics table to import the data into. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
import_filename	The name of the Analytics data file to create. Specify *import_filename* as a quoted string with a .FIL file extension. For example, "Invoices.FIL". By default, the data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ "C:\data\Invoices.FIL" ◦ "data\Invoices.FIL"
FROM *source_filename*	The name of the source data file. *source_filename* must be a quoted string. If the source data file is not located in the same directory as the Analytics project, you must use an absolute path or a relative path to specify the file location: ◦ "C:\data*source_filename*" ◦ "data*source_filename*"
SERVER *profile_name* optional	The profile name for the server that contains the data that you want to import.
character_set_value	The character set used to encode the Print Image (Report) file. The following values are supported: ◦ 0 - ASCII ◦ 1 - EBCDIC ◦ 2 - Unicode ◦ 3 - Encoded text
code_page_number optional	If you specified 3 (Encoded text) for *character_set_value*, you must also enter a code page number.

RECORD parameter

General record definition information.

> **Note**
>
> Some of the record definition information is specified using numeric codes that map to options in the Data Definition Wizard.
>
> **In scripts, specify the numeric code, not the option name.**

Name	Description
RECORD *record_name*	The name of the record in the Data Definition Wizard. Specifying *record_name* is required in the IMPORT PRINT command, but the *record_name* value does not appear in the resulting Analytics table. In the Data Definition Wizard, Analytics provides default names based on the type of record: ◦ Detail ◦ Header*n* ◦ Footer*n* You can use the default names, or specify different names.
record_type	The three possible record types when defining a Print Image file:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ 0 - detail ◦ 1 - header ◦ 2 - footer **Note** You can define multiple sets of header and footer records in a single execution of IMPORT PRINT, but only one set of detail records.
lines_in_record	The number of lines occupied by a record in the Print Image file. You can define single-line or multiline records to match the data in the file.
transparent	The transparency setting for a header record. **Note** Applies to header records only. ◦ 0 - not transparent ◦ 1 - transparent Transparent header records do not split multiline detail records. If a header record splits a multiline detail record in the source Print Image file, which can happen at a page break, specifying 1 (transparent) unifies the detail record in the resulting Analytics table.

TEST parameter

The criteria for defining a set of records in the Print Image file. You can have one or more occurrences of TEST (up to 8) for each occurrence of RECORD.

> **Note**
>
> Some of the criteria are specified using numeric codes that map to options in the Data Definition Wizard (option names are shown in parentheses below).
>
> **In scripts, specify the numeric code, not the option name.**

Name	Description
TEST *include_exclude*	How to treat matching data: ◦ 0 - (**Include**) data meeting the criteria is included in the set of records ◦ 1 - (**Exclude**) data meeting the criteria is excluded from the set of records
match_type	The type of matching to perform: ◦ 0 - (**Exact Match**) matching records must contain the specified character, or string of characters, in the specified start line, starting at the specified position ◦ 2 - (**Alpha**) matching records must contain one or more alpha characters, in the specified start line, at the specified start position, or in all positions of the specified range ◦ 3 - (**Numeric**) matching records must contain one or more numeric characters, in the specified start line, at the specified start position, or in all positions of the specified range

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ 4 - (**Blank**) matching records must contain one or more blank spaces, in the specified start line, at the specified start position, or in all positions of the specified range ◦ 5 - (**Non-Blank**) matching records must contain one or more non-blank characters (includes special characters), in the specified start line, at the specified start position, or in all positions of the specified range ◦ 7 - (**Find in Line**) matching records must contain the specified character, or string of characters, anywhere in the specified start line ◦ 8 - (**Find in Range**) matching records must contain the specified character, or string of characters, in the specified start line, anywhere in the specified range ◦ 10 - (**Custom Map**) matching records must contain characters that match the specified character pattern, in the specified start line, starting at the specified position
AT *start_line*, *start_position*, *range*	◦ ***start_line*** - the line of a record that the criteria apply to For example, if you create a custom map to match zip codes, and the zip codes appear on the third line of a three-line address record, you must specify 3 in *start_line*. **Note** For single-line records, the *start_line* value is always 1. ◦ ***start_position*** - the starting byte position in the Print Image file for the comparison against the criteria ◦ ***range*** - the number of bytes from the starting byte position in the Print Image file to use in the comparison against the criteria If you are using starting byte position only, without a range, specify 0 for *range*. **Note** non-Unicode Analytics: 1 byte = 1 character Unicode Analytics, extended ASCII (ANSI) data: 1 byte = 1 character Unicode Analytics, Unicode data: 2 bytes = 1 character For Unicode data, *range* must be an even number of bytes. For example, 50,59 (10 bytes). Specifying an odd number of bytes can prevent correct matching against criteria.
logic	The logical relations between criteria: ◦ 0 - (**And**) the current and the next criteria are related with a logical AND ◦ 1 - (**Or**) the current and the next criteria are related with a logical OR ◦ 4 - (**New Group** > **And**) the current criterion is the last in a group of logical criteria, and the current group and the next group are related with a logical AND ◦ 5 - (**New Group** > **Or**) the current criterion is the last in a group of logical criteria, and the current group and the next group are related with a logical OR ◦ 7 - (**End**) the current criterion is the last in a group of logical criteria
text	Literal or wildcard characters to match against: ◦ **For Exact Match, Find in Line, or Find in Range** - specifies the character, or string of characters, that uniquely identifies the set of records in the Print Image file ◦ **For Custom Map** - specifies the character pattern that uniquely identifies the set of

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	records in the Print Image file The **Custom Map** option uses the same syntax as the "MAP( ) function" on page 643. For other match types, *text* is an empty string "".

FIELD parameters

Field definition information.

Name	Description
FIELD *name type*	The individual fields to import from the source data file, including the name and data type of the field. To exclude a field from being imported, do not specify it. For information about *type*, see "Identifiers for field data types" on page 306.
AT *start_line*, *start_position*	◦ ***start_line*** - the start line of the field in the record in the Print Image file For multiline records in a Print Image file, *start_line* allows you to start a field at any line of the record. *start_line* is always 1 if *lines_in_record* is 1. ◦ ***start_position*** - the starting byte position of the field in the Print Image file **Note** non-Unicode Analytics: 1 byte = 1 character Unicode Analytics, extended ASCII (ANSI) data: 1 byte = 1 character Unicode Analytics, Unicode data: 2 bytes = 1 character
SIZE *length*, *lines_in_field*	◦ ***length*** - the length in bytes of the field in the Analytics table layout **Note** non-Unicode Analytics: 1 byte = 1 character Unicode Analytics, extended ASCII (ANSI) data: 1 byte = 1 character Unicode Analytics, Unicode data: 2 bytes = 1 character For Unicode data, specify an even number of bytes only. Specifying an odd number of bytes can cause characters to display incorrectly. ◦ ***lines_in_field*** - the number of lines occupied by a single field value in the Print Image file You can define single-line or multiline fields to match the data in the file.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note The number of lines specified for a field cannot exceed the number of lines specified for the record containing the field.
DEC *value*	The number of decimals for numeric fields.
WID *bytes*	The display width of the field in bytes. The specified value controls the display width of the field in Analytics views and reports. The display width never alters data, however it can hide data if it is shorter than the field length.
PIC *format*	**Note** Applies to numeric or datetime fields only. ◦ **numeric fields** - the display format of numeric values in Analytics views and reports ◦ **datetime fields** - the physical format of datetime values in the source data (order of date and time characters, separators, and so on) **Note** For datetime fields, *format* must exactly match the physical format in the source data. For example, if the source data is 12/31/2014, you must enter the format as "MM/DD/YYYY". *format* must be enclosed in quotation marks.
AS *display_name*	The display name (alternate column title) for the field in the view in the new Analytics table. Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title. AS is required when you are defining FIELD. To make the display name the same as the field name, enter a blank *display_name* value using the following syntax: AS "". Make sure there is no space between the two double quotation marks.

Examples

Importing data from a Print Image (Report) file

You import data from the Print Image (Report) file, `Report.txt`.

One header record, and one set of detail records, with five fields, is created in the resulting Analytics table, **Inventory_report**:

```
IMPORT PRINT TO Inventory_report "Inventory_report.FIL" FROM "Report.txt" 0 RECORD "Header1" 1 1 0 TEST 0 0 AT 1,17,0 7 ":" FIELD "Field_1" C AT 1,19 SIZE 2,1 DEC 0 WID 2 PIC "" AS "Prod Class" FIELD "Field_2" C AT 1,24 SIZE 31,1 DEC 0 WID 31 PIC "" AS "Prod Description"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
RECORD "Detail" 0 1 0 TEST 0 0 AT 1,59,59 7 "." FIELD "Field_3" X AT 1,6 SIZE 9,1 DEC 0 WID 9 PIC "" AS "Item ID" FIELD "Field_4" C AT 1,16 SIZE 24,1 DEC 0 WID 24 PIC "" AS "Item Desc." FIELD "Field_5" N AT 1,40 SIZE 10,1 DEC 0 WID 10 PIC "" AS "On Hand" FIELD "Field_6" N AT 1,50 SIZE 12,1 DEC 2 WID 12 PIC "" AS "Cost" FIELD "Field_7" N AT 1,62 SIZE 12,1 DEC 2 WID 12 PIC "" AS "Total"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Identifiers for field data types

The table below lists the letters that you must use when specifying *type* for FIELD. Each letter corresponds to an Analytics data type.

For example, if you are defining a Last Name field, which requires a character data type, you would specify "C": FIELD "Last_Name" C.

For more information, see Analytics data types.

> **Note**
>
> When you use the **Data Definition Wizard** to define a table that includes EBCDIC, Unicode, or ASCII fields, the fields are automatically assigned the letter "C" (for the CHARACTER type).
>
> When you enter an IMPORT statement manually, or edit an existing IMPORT statement, you can substitute the more specific letters "E" or "U" for EBCDIC or Unicode fields.

Letter	Analytics Data type
A	ACL
B	BINARY
C	CHARACTER
D	DATETIME
E	EBCDIC
F	FLOAT
G	ACCPAC
I	IBMFLOAT

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Letter	Analytics Data type
K	UNSIGNED
L	LOGICAL
N	PRINT
P	PACKED
Q	BASIC
R	MICRO
S	CUSTOM
T	PCASCII
U	UNICODE
V	VAXFLOAT
X	NUMERIC
Y	UNISYS
Z	ZONED

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT SAP command

Creates an Analytics table by importing data from an SAP system using *Direct Link*.

Syntax

IMPORT SAP PASSWORD *num* TO *table_name* SAP SOURCE "SAP AGENT" *import_details*

Parameters

Name	Description
PASSWORD *num*	The password definition to use. You do not use PASSWORD *num* to prompt for, or specify, an actual password. The password definition refers to a password previously supplied or set using the PASSWORD command, the SET PASSWORD command, or the PASSWORD analytic tag. *num* is the number of the password definition. For example, if two passwords have been previously supplied or set in a script, or when scheduling an analytic, PASSWORD 2 specifies that password #2 is used. For more information about supplying or setting passwords, see: • "PASSWORD command" on page 360 • "SET command" on page 418 • PASSWORD analytic tag **Note** The password is used to access the SAP system.
TO *table_name*	The name of the Analytics table to import the data into. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
SAP SOURCE "SAP AGENT"	Required for importing SAP data. "SAP AGENT" is the only available option.
import_details	The details of the query. Must be enclosed by the <q></q> tags, and uses the tags listed in "Direct Link query tags" on page 311 to define the query. The physical size of this value can be up to 16 KB.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Performing a multi-table query

This example performs a multi-table query using the IMPORT SAP command.

Correct order and nesting of the tags is necessary to create a valid query string. The tags in the example are ordered and nested correctly. Use this example to determine the required order and nesting of IMPORT SAP query tags.

> **Note**
>
> To assist readability, this example is formatted using multiple lines. In your script, the command and the query string must be entered without any line breaks.

> **Tip**
>
> The syntax of an IMPORT SAP query string is typically complex. The best way to add IMPORT SAP commands with query strings to your scripts is to copy an existing IMPORT SAP command from the **Log** tab in Analytics, then edit the query tags as necessary.

```
IMPORT SAP PASSWORD 1 TO Purchasing_doc SAP SOURCE "SAP AGENT"
<q version="6.0">
 <s>0</s>
 <d>IDES</d>
 <u>mzunini</u>
 <c>800</c>
 <lg>en</lg>
 <cf>C:\ACL Data\Purchasing_doc.fil</cf>
 <sf>E:\Data\DL_JSMITH111107.DAT</sf>
 <jcount>11110701</jcount>
 <jname>DL_JSMITH111107.DAT</jname>
 <dl>75</dl>
 <m>2</m>
 <dt>20140321</dt>
 <tm>033000</tm>
 <r>500</r>
 <ar>0</ar>
 <e>500</e>
 <ts>
  <t>
   <n>EKKO</n>
   <a>T00001</a>
   <td>Purchasing Document Header</td>
   <fs>
    <f>EBELN</f>
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
      <f>BUKRS</f>
      <f>BSTYP</f>
      <f>BSART</f>
      <f>STATU</f>
      <f>WKURS</f>
     </fs>
     <wc>
      <w>
       <f>BUKRS</f>
       <o>0</o>
       <l>1000</l>
       <h></h>
      </w>
     </wc>
    </t>
    <t>
     <n>EKPO</n>
     <a>T00002</a>
     <td>Purchasing Document Item</td>
     <fs>
      <f>EBELP</f>
      <f>WERKS</f>
      <f>MENGE</f>
      <f>BRTWR</f>
     </fs>
     <wc></wc>
    </t>
   </ts>
   <js>
    <jc>
     <pt>
      <pa>T00001</pa>
      <pf>EBELN</pf>
     </pt>
     <ct>
      <ca>T00002</ca>
      <cf>EBELN</cf>
     </ct>
    </jc>
   </js>
  </q>
```

Remarks

The IMPORT SAP command is only supported if Direct Link is installed and configured.

The table in "Direct Link query tags" below lists the tags that can be included in the *import_details* parameter. The **Required** column uses the following values to indicate when tags must be present:

- **Y** - Required
- **N** - Optional
- **M** - Required for multi-table queries only
- **B** - Required, but no value should be passed
- **W** - Optional when filters are used
- **S** - Required when scheduled mode is specified

Direct Link query tags

Name	Tag	Required	Description
Table Alias	<a>	M	The alias that uniquely identifies the table within the query. This allows the same table to be used more than once. The maximum length is 6 characters.
All Rows	<ar>	Y	Indicates that all matching rows should be returned as part of the query's result set. Valid values are: **1** - Overrides the number of records specified in the <r> tag (Maximum Rows) **0** - Returns the number of records specified in the <r> tag (Maximum Rows) This tag always appears after the <r></r> tag.
Client	<c>	N	The client within the SAP system.
Child Table Alias	<ca>	M	The alias of the child table.
Child Table Field	<cf>	M	The field in the child table that the join condition is based on.
Client Filename	<cf>	Y	Identifies the target file on the client system where the results of the query will be stored.
Child Table	<ct>	M	The child table in the join condition.
Destination	<d>	N	Identifies a destination in the SAP RFC library file (`sapnwrfc.ini`) that is used to locate an SAP system.
Data Length	<dl>	B	The number of characters in each row, including carriage return and line feed characters indicating the end of the record (CR+LF, or the hexadecimal characters 0D+0A).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Tag	Required	Description
Date	<dt>	S	Required when using scheduled mode. Specifies the time to run the SAP job. Must be formatted as YYYYMMDD. For example, December 31, 2014 must be specified as 20141231.
Expected Rows	<e>	B	The expected number of rows the query will return.
Field Name	<f>	Y	The native field name.
Filter Field	<f>	W	The native field name that the filter applies to.
Fields	<fs>	Y	The list of fields in the table that will be returned as part of the query results.
High Value	<h>	W	Contains the high value when using the Between operator. Ignored when using any other operator.
Join Condition	<jc>	M	The join condition.
Job Count	<jcount>	B	Used internally by SAP to identify a Background mode query.
Job Name	<jname>	B	Used internally by SAP to identify a Background mode query.
Join Relationships	<js>	Y	The list of join conditions that link tables within the query.
Join Switch	<jw>	N	Numeric equivalent of the join switch enumerated type. Valid values are: **0** - Inner Join **1** - Left Outer Join
Low Value	<l>	W	Contains either the lowest value when using the Between operator or the value when using any other operator.
Language	<lg>	Y	Language identifier used to determine the locale of fields in the SAP database.
Mode	<m>	Y	Numeric equivalent of the submission mode enumerated type. Valid values are: **0** - Extract Now **1** - Background **2** - Scheduled
Table Name	<n>	Y	The native table name.
Operator	<o>	W	Numeric equivalent of the operator enumerated type.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Tag	Required	Description
			Valid values are: **0** - Equal to (=) **1** - Not equal to (<>) **2** - Less than (<) **3** - Less than or equal to (<=) **4** - Greater than (>) **5** - Greater than or equal to (>=) **6** - Between **7** - Contains
Parent Table Alias	<pa>	M	The alias of the parent table.
Parent Table Field	<pf>	M	The field in the parent table the join condition is based on.
Parent Table	<pt>	M	The parent table in the join condition.
Query	<q>	Y	Encloses a query.
Maximum Rows	<r>	Y	The maximum number of rows the query should return.
Selected	<s>	Y	If the <s> tag appears below the <f> tag, it indicates whether the field will be returned as part of the query's result set.
System	<s>	Y	If the <s> tag appears below the <q> tag, it identifies the type of system this query is used against (currently only SAP is supported).
Server Filename	<sf>	B	Identifies the file on the server that holds the results of a Background mode query.
Server Group Name	<sg>	N	The name of the server group. Maximum 20 characters.
Server Name	<sn>	N	The name of the server. Maximum 20 characters.
Table	<t>	Y	The table.
Table Description	<td>	Y	The table description from the SAP data dictionary. It should always appear below the <a> tag.
Time	<tm>	S	Required when using scheduled mode. Specifies the time to run the SAP job. Must be formatted as hhmmss. For example, 2:30 pm must be specified as 143000.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Tag	Required	Description
Tables	<ts>	Y	The list of tables from which the query will extract data.
Table Type	<tt>	Y	The type of SAP table. Valid values are: **0** - clustered **1** - transparent **2** - pooled **3** - view
Username	<u>	N	The user's logon name.
Filter	<w>	W	The filter applied to the table's data.
Filters	<wc>	W	The list of filters that will be applied to the data contained within the table.
Filter Switch	<ws>	N	Numeric equivalent of the filter switch enumerated type. Valid values are: **0** - (Or) And (Or) **1** - (And) Or (And)

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT XBRL command

Creates an Analytics table by defining and importing an XBRL file.

Syntax

IMPORT XBRL TO *table import_filename* FROM *source_filename* CONTEXT *context_name* <...*n*> [field_syntax] <...*n*> <IGNORE *field_num*> <...*n*>

field_syntax ::=
FIELD *name type* AT *start_position* DEC *value* WID *bytes* PIC *format* AS *display_name*

Parameters

Name	Description
TO *table*	The name of the Analytics table to import the data into. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
import_filename	The name of the Analytics data file to create. Specify *import_filename* as a quoted string with a .FIL file extension. For example, "Invoices.FIL". By default, the data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ "C:\data\Invoices.FIL" ◦ "data\Invoices.FIL"
FROM *source_filename*	The name of the source data file. *source_filename* must be a quoted string. If the source data file is not located in the same directory as the Analytics project, you must use an absolute path or a relative path to specify the file location: ◦ "C:\data*source_filename*" ◦ "data*source_filename*"
CONTEXT *context_name*	The XBRL context to define the table from. If you specify more than one context, all contexts must be of the same type (instant, period, or forever).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
FIELD *name type*	The individual fields to import from the source data file, including the name and data type of the field. To exclude a field from being imported, do not specify it. For information about *type*, see "Identifiers for field data types" on the facing page.
AT *start_position*	The starting byte position of the field in the Analytics data file. **Note** non-Unicode Analytics: 1 byte = 1 character Unicode Analytics: 2 bytes = 1 character In Unicode Analytics, typically you should specify an odd-numbered starting byte position. Specifying an even-numbered starting position can cause characters to display incorrectly.
DEC *value*	The number of decimals for numeric fields.
WID *bytes*	The length in bytes of the field in the Analytics table layout. **Note** non-Unicode Analytics: 1 byte = 1 character Unicode Analytics: 2 bytes = 1 character In Unicode Analytics, specify an even number of bytes only. Specifying an odd number of bytes can cause characters to display incorrectly.
PIC *format*	**Note** Applies to numeric or datetime fields only. ◦ **numeric fields** - the display format of numeric values in Analytics views and reports ◦ **datetime fields** - the physical format of datetime values in the source data (order of date and time characters, separators, and so on) **Note** For datetime fields, *format* must exactly match the physical format in the source data. For example, if the source data is 12/31/2014, you must enter the format as "MM/DD/YYYY". *format* must be enclosed in quotation marks.
AS *display_name*	The display name (alternate column title) for the field in the view in the new Analytics table. Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title. AS is required when you are defining FIELD. To make the display name the same as the field name, enter a blank *display_name* value using the following syntax: AS "". Make sure there is no space between the two double quotation marks.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
IGNORE *field_num* optional	Excludes a field from the table layout. *field_num* specifies the position of the field in the source data. For example, IGNORE 5 excludes the fifth field in the source data from the Analytics table layout.

Examples

Importing an XBRL file to an Analytics table

You import data from the **Current_AsOf** context in an XBRL file to an Analytics table called **Financials**:

```
IMPORT XBRL TO Financials "Financials.fil" FROM "FinancialStatemenXBRL.xml" CONTEXT "Current_AsOf" FIELD "Item" C AT 1 DEC 0 WID 57 PIC "" AS "" FIELD "Value" X AT 58 DEC 0 WID 7 PIC "" AS "" IGNORE 1 IGNORE 3
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Identifiers for field data types

The table below lists the letters that you must use when specifying *type* for FIELD. Each letter corresponds to an Analytics data type.

For example, if you are defining a Last Name field, which requires a character data type, you would specify "C": FIELD "Last_Name" C.

For more information, see Analytics data types.

> **Note**
>
> When you use the **Data Definition Wizard** to define a table that includes EBCDIC, Unicode, or ASCII fields, the fields are automatically assigned the letter "C" (for the CHARACTER type).
>
> When you enter an IMPORT statement manually, or edit an existing IMPORT statement, you can substitute the more specific letters "E" or "U" for EBCDIC or Unicode fields.

Letter	Analytics Data type
A	ACL
B	BINARY

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Letter	Analytics Data type
C	CHARACTER
D	DATETIME
E	EBCDIC
F	FLOAT
G	ACCPAC
I	IBMFLOAT
K	UNSIGNED
L	LOGICAL
N	PRINT
P	PACKED
Q	BASIC
R	MICRO
S	CUSTOM
T	PCASCII
U	UNICODE
V	VAXFLOAT
X	NUMERIC
Y	UNISYS
Z	ZONED

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IMPORT XML command

Creates an Analytics table by defining and importing an XML file.

Syntax

IMPORT XML TO *table import_filename* FROM *source_filename* [field_syntax] <...*n*>

field_syntax ::=
FIELD *name type* AT *start_position* DEC *value* WID *bytes* PIC *format* AS *display_name* RULE *xpath_expression*

Parameters

Name	Description
TO *table*	The name of the Analytics table to import the data into. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
import_filename	The name of the Analytics data file to create. Specify *import_filename* as a quoted string with a .FIL file extension. For example, "Invoices.FIL". By default, the data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ "C:\data\Invoices.FIL" ◦ "data\Invoices.FIL"
FROM *source_filename*	The name of the source data file. *source_filename* must be a quoted string. If the source data file is not located in the same directory as the Analytics project, you must use an absolute path or a relative path to specify the file location: ◦ "C:\data*source_filename*" ◦ "data*source_filename*"
FIELD *name type*	The individual fields to import from the source data file, including the name and data type of the field. To exclude a field from being imported, do not specify it. For information about *type*, see "Identifiers for field data types" on page 321.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>AT start_position</td><td>The starting byte position of the field in the Analytics data file.
Note<table><tr><td>non-Unicode Analytics</td><td>1 byte = 1 character</td></tr><tr><td>Unicode Analytics</td><td>2 bytes = 1 character</td></tr></table>In Unicode Analytics, typically you should specify an odd-numbered starting byte position. Specifying an even-numbered starting position can cause characters to display incorrectly.</td></tr>
<tr><td>DEC value</td><td>The number of decimals for numeric fields.</td></tr>
<tr><td>WID bytes</td><td>The length in bytes of the field in the Analytics table layout.
Note<table><tr><td>non-Unicode Analytics</td><td>1 byte = 1 character</td></tr><tr><td>Unicode Analytics</td><td>2 bytes = 1 character</td></tr></table>In Unicode Analytics, specify an even number of bytes only. Specifying an odd number of bytes can cause characters to display incorrectly.</td></tr>
<tr><td>PIC format</td><td>Note
Applies to numeric or datetime fields only.
numeric fields - the display format of numeric values in Analytics views and reports
datetime fields - the physical format of datetime values in the source data (order of date and time characters, separators, and so on)
Note
For datetime fields, format must exactly match the physical format in the source data. For example, if the source data is 12/31/2014, you must enter the format as "MM/DD/YYYY".
format must be enclosed in quotation marks.</td></tr>
<tr><td>AS display_name</td><td>The display name (alternate column title) for the field in the view in the new Analytics table.
Specify display_name as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title.
AS is required when you are defining FIELD. To make the display name the same as the field name, enter a blank display_name value using the following syntax: AS "". Make sure there is no space between the two double quotation marks.</td></tr>
<tr><td>RULE xpath_expression</td><td>The XPath expression used to select the field contents from the XML file.
XPath is a standard way of accessing data from XML files. For example, acct/title/text() retrieves the text within the <title> tag in the XML file.</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Importing data from an XML file to an Analytics table

You import data from an XML file to an Analytics table named **Employees**:

```
IMPORT XML TO Employees "Employees.fil" FROM "emp.XML" FIELD "Empno" C AT 1 DEC 0 WID 6 PIC "" AS "" RULE "/RECORDS/RECORD/Empno/text()" FIELD "First" C AT 7 DEC 0 WID 13 PIC "" AS "" RULE "/RECORDS/RECORD/First/text()" FIELD "Last" C AT 20 DEC 0 WID 20 PIC "" AS "" RULE "/RECORDS/RECORD/Last/text()" FIELD "HireDate" D AT 40 DEC 0 WID 10 PIC "YYYY-MM-DD" AS "" RULE "/RECORDS/RECORD/HireDate/text()" FIELD "Salary" N AT 50 DEC 2 WID 8 PIC "" AS "" RULE "/RECORDS/RECORD/Salary/text()"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Identifiers for field data types

The table below lists the letters that you must use when specifying *type* for FIELD. Each letter corresponds to an Analytics data type.

For example, if you are defining a Last Name field, which requires a character data type, you would specify "C": FIELD "Last_Name" C.

For more information, see Analytics data types.

> **Note**
>
> When you use the **Data Definition Wizard** to define a table that includes EBCDIC, Unicode, or ASCII fields, the fields are automatically assigned the letter "C" (for the CHARACTER type).
>
> When you enter an IMPORT statement manually, or edit an existing IMPORT statement, you can substitute the more specific letters "E" or "U" for EBCDIC or Unicode fields.

Letter	Analytics Data type
A	ACL
B	BINARY
C	CHARACTER
D	DATETIME

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Letter	Analytics Data type
E	EBCDIC
F	FLOAT
G	ACCPAC
I	IBMFLOAT
K	UNSIGNED
L	LOGICAL
N	PRINT
P	PACKED
Q	BASIC
R	MICRO
S	CUSTOM
T	PCASCII
U	UNICODE
V	VAXFLOAT
X	NUMERIC
Y	UNISYS
Z	ZONED

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

INDEX command

Creates an index for an Analytics table that allows access to the records in a sequential order rather than a physical order.

Syntax

INDEX <ON> {*key_field* <D> <...*n*>|ALL} TO *file_name* <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <OPEN> <ISOLOCALE *locale_code*>

Parameters

Name	Description
ON *key_field* D <...*n*> | ALL	The key field or fields, or the expression, to use for indexing. You can index by any type of field, including computed fields and ad hoc expressions, regardless of data type. ◦ ***key_field*** - use the specified field or fields If you index by more than one field, you create nested indexing in the table. The order of nesting follows the order in which you specify the fields. Include D to index the key field in descending order. The default index order is ascending. ◦ **ALL** - use all fields in the table If you index by all the fields in a table you create nested indexing. The order of nesting follows the order in which the fields appear in the table layout. An ascending index order is the only option for ALL.
TO *file_name*	The name of the index and the associated index file. The index file is created with an .INX extension. **Note** In the Analytics user interface, index names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
OPEN optional	Open the table and apply the index to the table.
ISOLOCALE *locale_code* optional	**Note** Applicable in the Unicode edition of Analytics only. The system locale in the format *language_country*. For example, to use Canadian French, enter fr_ca. Use the following codes: ◦ **language** - ISO 639 standard language code ◦ **country** - ISO 3166 standard country code If you do not specify a country code, the default country for the language is used. If you do not use ISOLOCALE, the default system locale is used.

Examples

> **Note**
> For more information about how this command works, see the Analytics Help.

Create an index and open the table

In the Vendor table, you create an index on the **Vendor City** field and open the table:

```
OPEN Vendor
INDEX ON Vendor_City to "CityIndex" OPEN
```

Create an index and apply it to a table

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

In the Vendor table, you create an index on the **Vendor City** field. Later, you apply the index to the table:

```
OPEN Vendor
INDEX ON Vendor_City to "CityIndex"
.
.
.
SET INDEX TO "CityIndex"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

JOIN command

Combines fields from two Analytics tables into a new, single Analytics table.

> **Note**
>
> To use fuzzy matching to join tables, see "FUZZYJOIN command" on page 217.

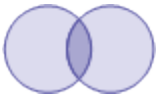
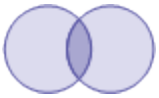
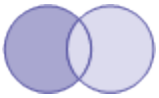
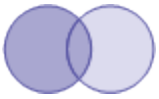
Syntax

```
JOIN {PKEY primary_key_fields|PKEY ALL} {FIELDS primary_fields|FIELDS ALL} {SKEY secondary_key_fields|SKEY ALL} <WITH secondary_fields|WITH ALL> {no_keyword|MANY|UNMATCHED|PRIMARY|SECONDARY|PRIMARY SECONDARY} <IF test> TO table_name <LOCAL> <OPEN> <WHILE test> <FIRST range|NEXT range> <APPEND> <PRESORT> <SECSORT> <ISOLOCALE locale_code>
```

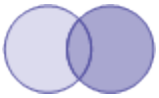
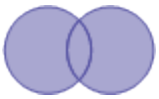
Parameters

Name	Description
PKEY *primary_key_fields* | PKEY ALL	The key field or fields, or expression, in the primary table. ◦ ***primary_key_fields*** - use the specified field or fields ◦ **ALL** - use all fields in the table
FIELDS *primary_fields* | FIELDS ALL	The fields or expressions from the primary table to include in the joined output table. ◦ ***primary_fields*** - include the specified field or fields ◦ **ALL** - include all fields from the table **Note** You must explicitly specify the primary key field or fields if you want to include them in the joined table. Specifying ALL also includes them.
SKEY *secondary_key_fields* | SKEY ALL	The key field or fields, or expression, in the secondary table. ◦ ***secondary_key_fields*** - use the specified field or fields ◦ **ALL** - use all fields in the table
WITH *secondary_fields* | WITH ALL optional	The fields or expressions from the secondary table to include in the joined output table. ◦ ***secondary_fields*** - include the specified field or fields ◦ **ALL** - include all fields from the table **Note** You must explicitly specify the secondary key field or fields if you want to include them in the joined table. Specifying ALL also includes them. You cannot specify WITH if you are using the UNMATCHED join type.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td>no_keyword | MANY | UNMATCHED | PRIMARY | SECONDARY | PRIMARY SECONDARY</td><td>
The type of join to perform.
<h3>no_keyword (omit all join-type keywords)</h3>

<table>
<tr><th>The joined output table contains:</th><th>Corresponding option in the Join dialog box</th></tr>
<tr><td>◦ all matched primary records and the first matched secondary record</td><td>Matched primary and secondary
(1st secondary match)</td></tr>
</table>
<h3>MANY</h3>

<table>
<tr><th>The joined output table contains:</th><th>Corresponding option in the Join dialog box</th></tr>
<tr><td>◦ all matched primary records and all matched secondary records
◦ one record for each match between the primary and secondary tables</td><td>Matched primary and secondary
(all secondary matches)</td></tr>
</table>
<h3>UNMATCHED</h3>

<table>
<tr><th>The joined output table contains:</th><th>Corresponding option in the Join dialog box</th></tr>
<tr><td>◦ unmatched primary records</td><td>Unmatched primary</td></tr>
</table>
<h3>PRIMARY</h3>

</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	<table><tr><th>The joined output table contains:</th><th>Corresponding option in the Join dialog box</th></tr><tr><td>○ all primary records (matched and unmatched) and the first matched secondary record</td><td>**All primary and matched secondary**</td></tr></table> **Note** The keyword BOTH is the same as specifying PRIMARY. **SECONDARY**  <table><tr><th>The joined output table contains:</th><th>Corresponding option in the Join dialog box</th></tr><tr><td>○ all secondary records (matched and unmatched) and all matched primary records Only the first instance of any duplicate secondary matches is joined to a primary record.</td><td>**All secondary and matched primary**</td></tr></table> **PRIMARY SECONDARY**  <table><tr><th>The joined output table contains:</th><th>Corresponding option in the Join dialog box</th></tr><tr><td>○ all primary and all secondary records, matched and unmatched Only the first instance of any duplicate secondary matches is joined to a primary record.</td><td>**All primary and secondary**</td></tr></table>
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note For most join types, an IF condition applies only to the primary table. The one exception is a many-to-many join, in which the IF condition can also reference the secondary table.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
PRESORT optional	Sorts the primary table on the primary key field before executing the command. **Note** You cannot use PRESORT inside the GROUP command. **Indexing instead of sorting** The primary table can be indexed instead of sorted. With large tables, indexing instead of sorting may reduce the time required to join the tables. If you are joining two tables using an indexed common key field, omit PRESORT and SECSORT.
SECSORT optional	Sorts the secondary table on the secondary key field before executing the command. **Note** You cannot use SECSORT inside the GROUP command. **Indexing instead of sorting** The secondary table can be indexed instead of sorted. With large tables, indexing instead of sorting may reduce the time required to join the tables. If you are joining two tables using an indexed common key field, omit PRESORT and SECSORT.
ISOLOCALE *locale_code* optional	**Note** Applicable in the Unicode edition of Analytics only. The system locale in the format *language_country*. For example, to use Canadian French, enter fr_ca. Use the following codes: ◦ **language** - ISO 639 standard language code ◦ **country** - ISO 3166 standard country code If you do not specify a country code, the default country for the language is used. If you do not use ISOLOCALE, the default system locale is used.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Join two tables as a way of discovering employees who may also be vendors

The example below joins the Empmast and Vendor tables using address as the common key field (the Address and Vendor_Street fields).

The JOIN command creates a new table with matched primary and secondary records, which results in a list of any employees and vendors with the same address.

```
OPEN Empmast PRIMARY
OPEN Vendor SECONDARY
JOIN PKEY Address FIELDS Empno First Last Address SKEY Vendor_Street WITH Vendor_No
Vendor_Name Vendor_Street TO "Employee_Vendor_Match" OPEN PRESORT SECSORT
```

This version of the JOIN command includes all fields from the primary and secondary tables in the joined output table.

```
OPEN Empmast PRIMARY
OPEN Vendor SECONDARY
JOIN PKEY Address FIELDS ALL SKEY Vendor_Street WITH ALL TO "Employee_Vendor_Match"
OPEN PRESORT SECSORT
```

Join two tables as a way of discovering accounts receivable records with no matching customer

The example below joins the Ar and Customer tables using Customer Number (CustNo) as the common key field.

The JOIN command uses the UNMATCHED join type to create a new table with unmatched primary records, which results in a list of Ar records that are not associated with any Customer record.

```
OPEN Ar PRIMARY
OPEN Customer SECONDARY
JOIN PKEY CustNo FIELDS CustNo Due Amount SKEY CustNo UNMATCHED TO "Cus-
tomerNotFound.fil" OPEN PRESORT SECSORT
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

LIST command

Outputs the data in one or more fields in an Analytics table to a display formatted in columns.

Syntax

LIST {FIELDS *field_name* <AS *display_name*> <...*n*>|FIELDS ALL} <LINE *number field_list*> <TO {SCREEN|*filename*|PRINT}> <UNFORMATTED> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <HEADER *header_text*> <FOOTER *footer_text*> <SKIP *lines*> <EOF> <APPEND>

Parameters

Name	Description
FIELDS *field_name* <...*n*> | FIELDS ALL	The fields to include in the output. ◦ **FIELDS *field_name*** - include the specified field or fields Fields are included in the order that you list them. ◦ **FIELDS ALL** - include all fields in the table Fields are included in the order that they appear in the table layout.
AS *display_name* optional	Only used when listing data using FIELDS *field_name*. The display name (alternate column title) for the field in the output. If you want the display name to be the same as the field name, or an existing display name in the source table, do not use AS. Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title.
LINE *number field_list* optional	More than one line is used in the output for each record: ◦ ***number*** - the line number, must be between 2 and 60 inclusive ◦ ***field_list*** - the fields to include on that line
TO SCREEN | *filename* | PRINT optional	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Use either an absolute or relative file path to save the file to a different, existing folder: • TO "C:\Output.TXT" • TO "Results\Output.TXT" ◦ **PRINT** - sends the results to the default printer
UNFORMATTED optional	The output is displayed as unformatted text. Output is identical to that created by the EXPORT ASCII command. Unformatted data can be output to a file for further processing by other software programs.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.
SKIP *lines* optional	Inserts the specified number of blank lines between each record in the list. For example, LIST ALL SKIP 1 produces a double spaced list (one blank line between each record).
EOF optional	Execute the command one more time after the end of the file has been reached. This ensures that the final record in the table is processed when inside a GROUP command. Only use EOF if all fields are computed fields referring to earlier records.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it.

Examples

Listing exceptions and saving to a text file

You use LIST to create a report listing exceptions identified in an inventory table. The report is saved as a text file:

```
LIST Product_number Description Quantity Unit_cost Value IF Quantity < 0 OR Unit_cost < 0
HEADER "Negative Values" TO "Exceptions.txt"
```

Remarks

When to use LIST

Use LIST to print data, display data on screen, or export it to a text file.

Formatting and totals

Unless you specify UNFORMATTED, the following information is included automatically:

- page numbers
- date
- time
- user identification
- column headings

Numeric columns are also automatically totaled.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

LOCATE command

Searches for the first record that matches the specified value or condition, or moves to the specified record number.

Syntax

```
LOCATE {IF test <WHILE test> <FIRST range|NEXT range>|RECORD num}
```

Parameters

Name	Description
IF *test*	The value or condition to search for. You must enclose character literal values in quotation marks, and datetime values in backquotes.
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
RECORD *num*	The record number to locate.

Examples

Locate the first record that matches a specified value

The following examples illustrate using LOCATE to find the first occurrence of a specific value in a table:

```
LOCATE IF Vendor_Name = "United Equipment"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
LOCATE IF Vendor_Name = "Uni"
```

```
LOCATE IF Invoice_Amount > 1000
```

```
LOCATE IF Invoice_Date = `20141231`
```

Locate the first record that matches a specified condition or expression

The following examples illustrate using LOCATE to find the first occurrence of a specific condition or expression in a table:

```
LOCATE IF Vendor_Name = "United Equipment" AND Invoice_Amount > 1000 AND Invoice_Date > `20140930`
```

```
LOCATE IF Vendor_City = v_city
```

Locate a record by record number

The following example illustrates using LOCATE to move to a particular record in a table:

```
LOCATE RECORD 50
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

How it works

Use the LOCATE command to move directly to the first record in a table matching the specified value or condition.

If the specified value or condition is found, the first matching record in the table is selected. If the specified value or condition is not found, the table is positioned at the first record.

You can also use LOCATE to move directly to a specific record number

LOCATE compared to FIND and SEEK

Unlike the FIND and SEEK commands, the LOCATE command is not restricted to searching an indexed table, or a single character field. Using LOCATE, you can search for any type of literal, or for an expression

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

that uses any data type, or mix of data types.

When used to search an unindexed table, the LOCATE command may be significantly slower than FIND or SEEK because it must process each record in the table sequentially. The required processing time depends on the size of the table, the location of a matching record, and whether you reduce the scope of the search by using WHILE, FIRST, or NEXT.

Partial matching supported

Partial matching is supported for character searches. The search value can be contained by a longer value in the field or fields being searched. However, search values must appear at the start of fields to constitute a match.

Enabling or disabling partial matching

You can enable or disable partial matching using either the SET command, or a setting in the **Options** dialog box:

Enable partial matching	Disable partial matching
Specify: SET EXACT OFF or **Deselect: Exact Character Comparisons** in the **Options** dialog box (**Tools > Options > Table**) **Result:** The search value can be contained by a longer value in the field or fields being searched. The search value must appear at the start of a field to constitute a match.	**Specify:** SET EXACT ON or **Select: Exact Character Comparisons** in the **Options** dialog box (**Tools > Options > Table**) **Result:** The search value must exactly match a value in a field to constitute a match.

For more information about SET EXACT, see "SET command" on page 418.

For more information about the **Exact Character Comparisons** option, see Table tab (Options dialog box).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

LOOP command

Executes a series of ACLScript commands repeatedly on a record while a specified condition evaluates to true.

> **Note**
>
> The LOOP command must be enclosed inside the GROUP command.

Syntax

```
LOOP WHILE test
 command
 <...n>
END
```

Parameters

Name	Description
WHILE *test*	The test that must evaluate to true for the commands inside the LOOP command to be executed. If the test evaluates to true the commands are executed repeatedly until the test evaluates to false.
command <...*n*>	One or more commands to execute. You can enter multiple commands inside the LOOP command. Each command must start on a new line.
END	The end of the LOOP command.

Examples

Splitting a comma-delimited field

You have a table containing invoice data and you need to isolate specific information for invoice amounts per department. One invoice may be related to more than one department, and department codes are stored in comma-delimited format in the table.

To extract the invoice amounts per department, you:

1. Use a GROUP command to process the table record by record.
2. Calculate the number of departments (n) associated with each record.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

3. Use the LOOP command to iterate n times over the record to extract data for each department associated with the record.

```
COMMENT
use GROUP to count commas in each department code field as a way of identifying how many departments are associated with the record
"LOOP" over each record for each code in the field, extracting each code into its own record
END
GROUP
 v_department_count = OCCURS(Department_Code,',')
 v_counter = 0
 LOOP WHILE v_counter <= v_department_count
  v_dept = SPLIT(Department_Code, ',', (v_counter + 1))
  EXTRACT FIELDS Invoice_Number, Invoice_Amount, v_dept AS "Department" TO result1
  v_counter = v_counter + 1
 END
END
```

Remarks

> **Tip**
>
> For a detailed tutorial covering the LOOP and GROUP commands, see "Grouping and looping" on page 32.

When to use LOOP

Loops are frequently used when a record contains repeated segments of data that you want to process.

How it works

Each LOOP command must specify a WHILE condition to test, and be closed with an END statement. The commands between LOOP and END are executed repeatedly for the current record as long as the specified test is true.

If the test is initially false, the commands are not executed.

Avoiding infinite loops

To avoid creating an infinite loop, make sure that the test you specify eventually returns false. You can also use the SET LOOP command to prevent infinite looping.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

MERGE command

Combines records from two sorted Analytics tables with an identical structure into a new Analytics table that uses the same sort order as the original tables.

Syntax

MERGE {ON *key_fields*|PKEY *primary_key_fields* SKEY *secondary_key_fields*} <IF *test*> TO *table_name* <LOCAL> <OPEN> <WHILE *test*> <FIRST *range*|NEXT *range*> <APPEND> <PRESORT> <ISOLOCALE *locale_code*>

Parameters

Name	Description
ON *key_fields* | PKEY *primary_key_fields* SKEY *secondary_key_fields*	**Note** Only character fields, or character computed fields, can be used as key fields in MERGE. ◦ **ON *key_fields*** – the key field or fields to merge on if the corresponding key fields in the primary and secondary tables have the same name If the corresponding fields have different names, or if they are expressions rather than actual physical fields, you must use PKEY and SKEY. ◦ **PKEY *primary_key_fields*** – the key field or fields, or expression, in the primary table ◦ **SKEY *primary_key_fields*** – the key field or fields, or expression, in the secondary table **Sorting requirement** The key fields in the primary and secondary tables must both be sorted in ascending order. If one or both key fields are unsorted, or sorted in descending order, the MERGE command fails. You can use PRESORT to sort the primary key field. If the secondary key field is unsorted, you must first sort it in a separate sort operation before performing the merge. **Indexing instead of sorting** The primary and secondary tables can be indexed instead of sorted. With large tables, indexing instead of sorting may reduce the time required to merge the tables.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
PRESORT optional	Sorts the primary table on the primary key field before executing the command. **Note** You cannot use PRESORT inside the GROUP command. Omit PRESORT: ◦ If the primary key field is already sorted ◦ If you are merging two tables using an indexed common key field
ISOLOCALE *locale_code* optional	**Note** Applicable in the Unicode edition of Analytics only. The system locale in the format *language_country*. For example, to use Canadian French, enter fr_ca. Use the following codes: ◦ **language** - ISO 639 standard language code ◦ **country** - ISO 3166 standard country code If you do not specify a country code, the default country for the language is used. If you do not use ISOLOCALE, the default system locale is used.

Examples

Merge tables with identical key field names

The following example merges two tables with identical key field names:

```
OPEN Employees_Location_1 PRIMARY
OPEN Employees_Location_2 SECONDARY
MERGE ON Last_Name TO "AllEmployees" PRESORT
```

Merge tables with different key field names

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

The following example merges two tables with different key field names:

```
OPEN Employees_Location_1 PRIMARY
OPEN Employees_Location_2 SECONDARY
MERGE PKEY Last_Name SKEY Surname TO "AllEmployees" PRESORT
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Alternatives to merging

Merging can be tricky to perform correctly. You can get the same result by appending, or by extracting and appending, and then sorting.

For more information, see "APPEND command" on page 73, and "EXTRACT command" on page 203.

If the two source tables are already sorted, merging is more efficient and can execute more quickly.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

NOTES command

Creates, modifies, or removes a note associated with an individual record in an Analytics table.

Syntax

```
NOTES <IF test> <TEXT note_text> <APPEND> <CLEAR>
```

Parameters

Name	Description
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT). ◦ If you do not specify an IF test, the note text is added to each record in the table ◦ If you specify an IF test and CLEAR, the notes for those records that meet the condition are deleted
TEXT *note_text* optional	The text to add as a note. *note_text* must be either a string enclosed in quotation marks, or a character expression.
APPEND optional	The note text is added to the end of any existing notes. If omitted, any existing notes are overwritten.
CLEAR optional	Notes are deleted. Even if all record notes in a table are deleted, the auto-generated **RecordNote** field is not deleted from the table layout.

Examples

Adding the same note to multiple records

Any existing notes for the specified records are overwritten:

```
NOTES IF MATCH(RECNO(),1,3,5,7) TEXT "note text"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Adding or append the same note to multiple records

Any existing notes for the specified records have the new note text appended:

```
NOTES IF MATCH(RECNO(),1,3,5,7) TEXT "note text" APPEND
```

Deleting notes from multiple records

All record notes in the table are deleted:

```
NOTES CLEAR
```

Notes for the specified records are deleted:

```
NOTES IF MATCH(RECNO(),1,3,5,7) CLEAR
```

Notes for records 1-100 are deleted:

```
NOTES IF RECNO() <= 100 CLEAR
```

Remarks

Deleting the RecordNote field

To delete the **RecordNote** field from the table layout, and all notes in the table, use the DELETE NOTES command without any of the options specified.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

NOTIFY command

Sends an email notification message.

Syntax

NOTIFY USER *username* <PASSWORD *pwd*> MAILBOX *pathname* ADDRESS *recipient* <CC *cc_recipient*> <BCC *bcc_recipient*> <SUBJECT *subject*> MESSAGE *message* <ATTACHMENT *pathname*>

Parameters

Name	Description
USER *username*	The email address of the sender.
PASSWORD *pwd* optional	The password for the mail server.
MAILBOX *pathname*	The SMTP server name to use to send the email message. For example: `MAILBOX "mailserver.example.com"`
ADDRESS *recipient*	The email address of one or more recipients. Separate multiple email addresses with a comma. Enter a maximum of 1020 characters.
CC *cc_recipient* optional	The email address of one or more carbon copy recipients. Separate multiple email addresses with a comma. Enter a maximum of 1000 characters.
BCC *bcc_recipient* optional	The email address of one or more blind carbon copy recipients. Separate multiple email addresses with a comma.
SUBJECT *subject* optional	The subject line of the email message.
MESSAGE *message*	The body text of the email message. The message is plain text and does not support HTML. If you want to insert a line break in your message, use two carat characters: `^^`.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
ATTACHMENT *pathname* optional	The path and filename of one or more attachments. Must be a quoted string. Specify multiple attachments by entering a comma separated list of files for *pathname*: `ATTACHMENT "result1,result2"`

Examples

Sending an error report email

You are running a script, and you want to send a notification email if the script fails. Using NOTIFY, you define the email message and include two attachments:

- the log file
- a .fil file containing recorded errors

```
NOTIFY USER "support@company.com" MAILBOX "mail.company.com" ADDRESS "script_admin@example.com" SUBJECT "Error Report" MESSAGE "Failed to process script. Details attached." ATTACHMENT "Errors.fil,ACL_Demo.log"
```

Remarks

Recipients and attachments

You can use the NOTIFY command to send email notification messages to one or more recipients. Messages can include attached data files and Analytics projects.

The NOTIFY command can be used to notify the appropriate personnel when a script fails unexpectedly.

Protocols and ports

The command can be used with any mail server that supports SMTP (Simple Mail Transfer Protocol), which is used by Microsoft Exchange and many other mail servers. The NOTIFY command can also be used with older email applications, from Microsoft and others, that send mail locally.

NOTIFY uses port 25, so this port must be open on the mail server or the command fails. The port number used by the command is not configurable. If NOTIFY fails with an error message, contact your IT department to find out if port 25 is blocked on your network.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Error handling

If Analytics is unable to connect with the mail server, it makes five additional attempts to connect, with a 10-second pause between each attempt. If all connection attempts are unsuccessful, the NOTIFY command is canceled, with a message written to the log, but the script continues processing.

You can use the SET command to change this default behavior. You can specify a different number of connection attempts and a different amount of time between attempts, or you can turn off additional connection attempts. You can also specify that Analytics stops processing a script if the NOTIFY command is canceled. For more information, see "SET command" on page 418.

An invalid email recipient is not considered a failure of the NOTIFY command and does not cause a script to stop regardless of the associated setting.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

OPEN command

Opens an Analytics table and the associated data file.

Syntax

OPEN {*table_name*|*data_file* <FORMAT *layout_name*>} <BUFFERLENGTH *length*> <CRLF> <DBASE> <INDEX *index_file*> <PRIMARY|SECONDARY> <SKIP *bytes*> <RELATION *key_field*>

Parameters

Name	Description
table_name	The name of the Analytics table to open.
data_file	The data file to associate with the table specified by FORMAT *layout_name*. Analytics assumes a file extension of .fil if no extension is specified. To open a file with no extension, insert a period (.) at the end of the file name.
FORMAT *layout_name* optional	The Analytics table layout to apply to the data file that you open as a table.
BUFFERLENGTH *n* optional	The length in bytes of the input buffer area to be allocated to the table. The default value is 33,000 bytes. Larger buffer areas may improve processing speed at the expense of RAM available for storing Analytics commands. If any IBM variable length blocks are read which exceed the buffer length, Analytics displays an error message and stops processing. The default value is set in the **Buffer Size** field in the **Table** tab in the **Options** dialog box. You will seldom need to change BUFFERLENGTH *n*, because the default is sufficient to handle almost all situations.
CRLF optional	Specifies that a variable length ASCII file is to be read. Analytics automatically adjusts for the varying record lengths. By default, files are assumed to be fixed-length files.
DBASE optional	Specifies that the data source is a dBASE file. Analytics recognizes the type of dBASE file and automatically creates a table from the file description. Can be omitted for dBASE files with a .dbf extension.
INDEX *index_file* optional	The index file to apply to the table when it is opened. The file extension for the index file name is assumed to be .inx when none is specified.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	You can specify INDEX with either primary or secondary tables.
PRIMARY | SECONDARY optional	Specifies that a table is opened as either a primary table or a secondary table. If omitted, the table is opened as a primary table.
SKIP *bytes* optional	The number of bytes to bypass at the physical start of the table. SKIP can be used to ignore table header records or leading portions of the table that do not follow the layout of the remainder of the table. If omitted, the table is read starting at the first byte. **Note** non-Unicode Analytics: 1 byte = 1 character Unicode Analytics, extended ASCII (ANSI) data: 1 byte = 1 character Unicode Analytics, Unicode data: 2 bytes = 1 character For Unicode data, specify an even number of bytes only. Specifying an odd number of bytes can cause characters to display incorrectly.
RELATION *key_field* optional	Specifies that the table is to be opened as an ad hoc related table. Analytics does not retain this relation when the table is closed. You must also specify the INDEX parameter when you use RELATION. *key_field* is the key field or expression used to create the relation between two tables.

Examples

Opening a table while specifying a table layout

You open the **April_2012** table using the **March_2012** table layout:

```
OPEN April_2012 FORMAT March_2012
```

Opening a dBASE file

You open a dBASE file named `Inventory.dbf` for which there is no existing table:

```
OPEN Inventory
```

Opening a table and apply a pre-existing index

To open either a primary or a secondary table, and apply an index that already exists for the table, use the following syntax:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
OPEN Accounts_receivable INDEX Customer_number_AR
```

```
OPEN Customer SECONDARY INDEX Customer_number
```

Opening a table and establish an ad hoc relation to another table

You need to establish a temporary relation between an open table named **Customers** (the primary table) and a table named **Accounts_receivable** (the secondary table).

You use an index named **Customer_index** and a key field in the primary table named **Last_name**:

```
OPEN Accounts_receivable INDEX Customer_index RELATION Last_name
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

OUTLIERS command

Identifies statistical outliers in a numeric field. Outliers can be identified for the field as a whole, or for separate groups based on identical values in one or more character, numeric, or datetime key fields.

Syntax

OUTLIERS {AVERAGE|MEDIAN} {PKEY *key_field* <...*n*>|NOKEY} ON *numeric_field* <OTHER *field* <...*n*>> NUMSTDEV *number_of_std_devs* <IF *test*> <TO {SCREEN|*table_name*}> <PRESORT> <WHILE *test*> <FIRST *range*|NEXT *range*> <OPEN>

> **Note**
>
> You cannot run the OUTLIERS command locally against a server table.
>
> You must specify the OUTLIERS command name in full. You cannot abbreviate it.

Parameters

Name	Description
AVERAGE | MEDIAN	The method for calculating the center point of the values in *numeric_field* (the outlier field). ◦ **AVERAGE** - calculate the average (mean) of the values ◦ **MEDIAN** - calculate the median of the values The center point is calculated for either: ◦ the numeric field as a whole ◦ the numeric values for each key field group The center point is subsequently used in calculating the standard deviation of the numeric field, or of each group. **Note** If you specify MEDIAN, *numeric_field* must be sorted. Use PRESORT if *numeric_field* is not already sorted. **Tip** If the data you are examining for outliers is significantly skewed, MEDIAN might produce results that are more representative of the bulk of the data.
PKEY *key_field* <...*n*> | NOKEY	If you specify PKEY, outliers are identified at the group level. If you specify NOKEY, outliers are identified at the field level. ◦ **PKEY *key_field*** - the field or fields to use for grouping the data in the table Key fields can be character, numeric, or datetime. Multiple fields must be separated

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	by spaces, and can be different data types. If you specify more than one field, you created nested groups. The nesting follows the order in which you specify the fields. For each key field group, a standard deviation is calculated for the group's numeric values in *numeric_field*. The group standard deviation is used as the basis for identifying group outliers. **Note** The key field or fields must be sorted. Use PRESORT if one or more fields are not already sorted. ◦ **NOKEY** - do not group the data in the table A standard deviation is calculated for *numeric_field* as a whole. The field standard deviation is used as the basis for identifying field outliers.
ON *numeric_field*	The numeric field to examine for outliers. You can examine only one field at a time. Outliers are values that fall outside the upper and lower boundaries established by the field or group standard deviation, or by a specified multiple of the standard deviation.
OTHER *field* <...*n*> optional	One or more additional fields to include in the output. **Note** Key fields and the outlier field are automatically included in the output table, and do not need to be specified using OTHER.

NUMSTDEV *number_of_std_devs*	In *numeric_field*, the number of standard deviations from the mean or the median to the upper and lower outlier boundaries. You can specify any positive integer or decimal numeral (0.5, 1, 1.5, 2 . . .)
	The formula for creating outlier boundaries is:
	`mean/median ± (number_of_std_devs * standard deviation)`
	Note
	Standard deviation is a measure of the dispersion of a data set - that is, how spread out the values are. The outliers calculation uses population standard deviation.
	Example of outlier boundaries
	`NUMSTDEV 2`
	establishes, for *numeric_field* as a whole, or for each key field group:
	• an upper outlier boundary 2 standard deviations greater than the mean or the median
	`mean/median + (2 * SD)`
	• a lower outlier boundary 2 standard deviations less than the mean or the median
	`mean/median - (2 * SD)`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>Any value greater than the upper boundary, or less than the lower boundary, is included as an outlier in the output results.

Note
For the same set of data, as you increase the value in number_of_std_devs you potentially decrease the number of outliers returned.</td></tr>
<tr><td>IF test
optional</td><td>A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition.

Note
The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).</td></tr>
<tr><td>TO SCREEN | table_name
optional</td><td>The location to send the results of the command to:
◦ SCREEN - displays the results in the Analytics display area

Tip
You can click any linked result value in the display area to drill down to the associated record or records in the source table.
◦ table_name - saves the results to an Analytics table
Specify table_name as a quoted string with a .FIL file extension. For example: <code>TO "Output.FIL"</code>
By default, the table data file (.FIL) is saved to the folder containing the Analytics project.
Use either an absolute or relative file path to save the data file to a different, existing folder:
• <code>TO "C:\Output.FIL"</code>
• <code>TO "Results\Output.FIL"</code>

Note
Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.</td></tr>
<tr><td>PRESORT
optional</td><td>Performs a sorting operation before executing the command.
<table><tr><th>If you specify PRESORT and:</th><th>Sorts by:</th></tr><tr><td>PKEY, AVERAGE</td><td>◦ key field or fields
◦ key field or fields, then by numeric_field (if numeric_field is computed)</td></tr></table></td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><th>If you specify PRESORT and:</th><th>Sorts by:</th></tr>
<tr><td></td><td>Note
Sorting a computed numeric_field is an internal, technical requirement of Analytics.</td></tr>
<tr><td>PKEY, MEDIAN</td><td>key field or fields, then by numeric_field</td></tr>
<tr><td>NOKEY, AVERAGE</td><td>no sorting</td></tr>
<tr><td>NOKEY, MEDIAN</td><td>numeric_field</td></tr>
</table>
Tip
If the appropriate field or fields in the input table are already sorted, you can save processing time by not specifying PRESORT.
Note
You cannot use PRESORT inside the GROUP command.
</td></tr>
<tr><td>WHILE test
optional</td><td>A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached.
Note
If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.</td></tr>
<tr><td>FIRST range | NEXT range
optional</td><td>The number of records to process:
◦ FIRST - start processing from the first record until the specified number of records is reached
◦ NEXT - start processing from the currently selected record until the specified number of records is reached
Use range to specify the number of records to process.
If you omit FIRST and NEXT, all records are processed by default.</td></tr>
<tr><td>OPEN
optional</td><td>Opens the table created by the command after the command executes. Only valid if the command creates an output table.</td></tr>
</table>

Examples

Identifying transaction amounts that are out of the ordinary

You want to identify transaction amounts that are out of the ordinary across the entire **Ar** table in `Sample`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

`Project.acl`.

You decide to set the outlier boundaries at 3 times the standard deviation of the **Amount** field. The test returns 16 outliers in the table of 772 records.

```
OPEN Ar
OUTLIERS AVERAGE NOKEY ON Amount NUMSTDEV 3 PRESORT TO "Outliers_AR.fil" OPEN
```

You repeat the test, but increase the standard deviation multiple to 3.5. The test now returns only 6 outliers because the outlier boundaries are farther away from the center point of the values in the **Amount** field.

```
OPEN Ar
OUTLIERS AVERAGE NOKEY ON Amount NUMSTDEV 3.5 PRESORT TO "Outliers_AR.fil"
OPEN
```

Identifying transaction amounts that are out of the ordinary for each customer

For each customer in the **Ar** table in `Sample Project.acl`, you want to identify any transaction amounts that are out of the ordinary.

You decide to set the outlier boundaries at 3 times the standard deviation of each customer's group of transactions.

```
OPEN Ar
OUTLIERS AVERAGE PKEY No ON Amount NUMSTDEV 3 PRESORT TO "Outliers_Customer_
AR.fil" OPEN
```

The test returns 7 outliers. The standard deviation and the average are reported for each customer's group of transactions:

	Cust Number (No)	Trans Amount	STDEV	AVERAGE	Group Number
1	065003	4,954.64	1015.58	833.83	1
2	**262001**	3,567.34	772.44	438.81	2
3	**262001**	(2,044.82)	772.44	438.81	2
4	376005	(931.55)	411.18	484.57	3
5	501657	5,549.19	1332.80	441.14	4
6	811002	3,409.82	634.20	672.10	5
7	925007	3,393.87	736.48	906.16	6

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

How outliers are identified for customer 262001

Customer 262001 has 101 transactions in the **Ar** table, and two of them are reported as outliers because they exceed the outlier boundaries for that customer:

Outlier	Lower boundary	Upper boundary	Outlier
(2,044.82)	(1,878.51)	2,756.13	3,567.34

How outlier boundaries are calculated for customer 262001

The outlier boundaries are the average of all customer 262001 transactions, plus or minus the specified multiple of the standard deviation of the transactions:

Average of all customer 262001 transactions	438.81
Specified multiple of the standard deviation	3
Standard deviation of the transactions	772.44
	438.81 ± (3 * 772.44) = 438.81 ± 2,317.32 = **(1,878.51) (lower boundary)** = **2,756.13 (upper boundary)**

Using MEDIAN to identify transaction amounts that are out of the ordinary for each customer

You use MEDIAN, instead of AVERAGE, to perform the same outlier test that you performed in the example above.

```
OPEN Ar
OUTLIERS MEDIAN PKEY No ON Amount NUMSTDEV 3 PRESORT TO "Outliers_Customer_AR_
Median.fil" OPEN
```

The test returns 10 outliers instead of the 7 that are returned in the previous test. Depending on the nature of the data, MEDIAN and AVERAGE can return somewhat different results:

	Cust Number (No)	Trans Amount	STDEV	MEDIAN	Group Number
1	065003	4,954.64	1015.58	663.68	1
2	262001	(2,044.82)	772.44	450.67	2
3	262001	3,567.34	772.44	450.67	2

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

	Cust Number (No)	Trans Amount	STDEV	MEDIAN	Group Number
4	376005	(931.55)	411.18	517.16	3
5	501657	4,426.14	1332.80	146.80	4
6	501657	5,549.19	1332.80	146.80	4
7	811002	3,409.82	634.20	624.53	5
8	925007	2,972.78	736.48	717.88	6
9	925007	3,030.71	736.48	717.88	6
10	925007	3,393.87	736.48	717.88	6

How outlier boundaries are calculated for each customer

The outlier boundaries are the median value of each customer's transactions, plus or minus the specified multiple of the standard deviation of the transactions.

For example, for customer 262001: 450.67 ± (3 * 772.44)

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Add outlier boundary fields to the results table

Analytics automatically adds the **STDEV** and **AVERAGE** or **MEDIAN** calculated fields to the outliers results table. You may find it useful to also add two computed fields that show the outliers boundaries used to identify the outliers in the results table.

1. Open the outliers results table.
2. Paste this expression into the Analytics command line, edit it as required, and press Enter:

   ```
   DEFINE FIELD Lower_Boundary COMPUTED AVERAGE - (number_of_std_devs * STDEV)
   ```

   - For *number_of_std_devs*, substitute the actual standard deviation multiple you used.
   - If you used median as a center point rather than average, substitute MEDIAN for AVERAGE.
3. Paste this expression into the Analytics command line, edit it as required, and press Enter:

   ```
   DEFINE FIELD Upper_Boundary COMPUTED AVERAGE + (number_of_std_devs * STDEV)
   ```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- For *number_of_std_devs*, substitute the actual standard deviation multiple you used.
- If you used median as a center point rather than average, substitute MEDIAN for AVERAGE.

4. Right-click the view and select **Add Columns**.
5. From the **Available Fields** list, double-click **Lower_Boundary** and **Upper_Boundary** to add them to the **Selected Fields** list.
6. Click **OK**.
7. Optional. Reposition the added fields by dragging the column headers.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PASSWORD command

Creates a password definition, without a password value, that prompts users for a password while a script is running.

Syntax

```
PASSWORD num <prompt>
```

Parameters

Name	Description
num	A value from 1 to 10 that uniquely identifies the password definition.
prompt optional	A valid character expression to display in the dialog box used to prompt for the password. Enclose literal strings in quotation marks. If *prompt* is omitted, a default dialog box with no message is displayed.

Examples

Prompting for password information

You use the PASSWORD command to prompt the user for the three passwords required in a script. Once the user enters the required passwords, the script can complete the remaining processing without interruption:

```
PASSWORD 1 "Enter the password for the Receivables database"
PASSWORD 2 "Enter the password for the Payables database"
PASSWORD 3 "Enter the password for the Customer database"
```

Specifying a password when refreshing an Analytics table

You combine the PASSWORD command with the REFRESH command to update a password-protected data file:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
PASSWORD 1 "Password:"
REFRESH Abc PASSWORD 1
```

Specifying a password to define a server table

You use the PASSWORD command with the DEFINE TABLE DB command to define a server table via AX Connector, which requires one password for the database profile, and another for the associated server profile:

```
DEFINE TABLE DB SOURCE Inventory_DBProfile PASSWORD 9 PASSWORD 3
```

Remarks

When to use PASSWORD

Use the PASSWORD command to prompt a user to enter password information before a script accesses, imports, or refreshes password-protected data.

You can create up to ten different password definitions in a script .

PASSWORD is useful if:

- you want to avoid typing an actual password in a script, which the SET PASSWORD command requires
- individual users need to enter distinct passwords

How passwords are stored

User-entered passwords are temporarily and securely stored in memory.

When a user types a password into the prompt dialog box, the characters are masked using asterisks (*). The password does not appear in either the script or the log.

Storing passwords for server-based analytics

The PASSWORD command is not supported in analytics run in Robots or on AX Server, or in legacy server scripts.

You can use the PASSWORD tag to prompt for a password when a user schedules an analytic in Robots or on AX Server.

You can use the SET PASSWORD command to specify passwords in legacy server scripts.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PAUSE command

Pauses a script, and displays information in a dialog box for users.

Syntax

```
PAUSE message <IF test>
```

Parameters

Name	Description
message	A message to display in the dialog box. The maximum length is 199 characters. *message* must be enclosed in quotation marks. If the message contains double quotation marks, enclose it in single quotation marks.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).

Examples

Displaying an error message

You require user input to meet specific requirements. When you detect that the input does not meet those requirements, you use the PAUSE command and display an error message in a dialog box:

```
PAUSE "The product class must be a 2-digit value."
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

When to use PAUSE

Use PAUSE to display read-only messages on screen in the course of running a script. You can display error messages or information such as the result of an analytic operation.

How it works

While the message dialog box is displayed, execution of the script is halted and only resumes once the user clicks **OK** to close the message dialog box. For this reason, you cannot use PAUSE in scripts or analytics that must run unattended.

Limitations

PAUSE has the following limitations:

- cannot be included inside the GROUP command
- cannot be used in analytics run in Robots, or on AX Server

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PREDICT command

Applies a predictive model to an unlabeled data set to predict classes or numeric values associated with individual records.

> **Note**
>
> The PREDICT command is not supported if you are running Analytics on a 32-bit computer. The computation required by the command is processor-intensive and better suited to 64-bit computers.

Syntax

```
PREDICT MODEL model_name TO table_name <IF test> <WHILE test> <FIRST range|NEXT range>
```

Parameters

Name	Description
MODEL *model_name*	The name of the model file to use for predicting classes or values. You use a model file previously generated by the TRAIN command. You must specify the `*.model` file extension. For example: `MODEL "Loan_default_prediction.model"` **Note** The model file must have been trained on a data set with the same fields as the unlabeled data set - or substantially the same fields.
TO *table_name*	The name of the Analytics table output by the prediction process. The table contains the key fields you specified during the training process, and either one or two fields generated by the prediction process: ◦ **Predicted** - the predicted classes or numeric values associated with each record in the unlabeled data set ◦ **Probability** - (classification only) the probability that a predicted class is accurate Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Loan_applicants_default_predicted.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ `TO "C:\Loan_applicants_default_predicted.FIL"`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ TO "ML Predict output\Loan_applicants_default_predicted.FIL" **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.

Examples

Use a classification model to make predictions

You input a classification model to the PREDICT command to make predictions about which current loan applicants will default if given a loan.

You previously produced the classification model using the TRAIN command with a set of historical loan data, including loan default information.

```
OPEN "Loan_applicants_current"
PREDICT MODEL "Loan_default_prediction.model" TO "Loan_applicants_default_predicted.FIL"
```

Use a regression model to make predictions

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You input a regression model to the PREDICT command to make predictions about the future sale price of houses.

You previously produced the regression model using the TRAIN command with a set of recent house sales data, including the sale price.

```
OPEN "House_price_evaluation"
PREDICT MODEL "House_price_prediction.model" TO "House_prices_predicted.FIL"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PRINT command

Prints a text file, an Analytics log file, or an Analytics project item that has been exported as an external file – a script (.aclscript), a table layout (.layout), or a workspace (.wsp). You can also print a graph that has been generated by a command.

Syntax

```
PRINT {file_name|GRAPH}
```

Parameters

Name	Description
file_name | GRAPH	The item to print: ◦ ***file_name*** – the relative or absolute path and file name of the file to print For example, "C:\ACL Data\Sample Data Files\ACL_Demo.log" or "Sample Data Files\ACL_Demo.log". If the path or file name includes spaces you must enclose *file_name* in quotation marks. ◦ **GRAPH** – the graph previously output as the result of a command

Examples

Printing a log file

To print the log file for the `ACL_Demo.acl` project, specify the following command:

```
PRINT "C:\ACL Data\Sample Data Files\ACL_Demo.log"
```

Printing a graph

To print the graph produced from the BENFORD command, specify the following commands:

```
OPEN Metaphor_APTrans_2002
BENFORD ON Invoice_Amount LEADING 1 TO GRAPH
PRINT GRAPH
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Selecting a printer

The printer used is the default printer configured in Microsoft Windows. To change the printer you need to change the default printer in Windows.

Related commands

To print the contents of an Analytics table in a project, use the DO REPORT command.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PROFILE command

Generates summary statistics for one or more numeric fields, or numeric expressions, in an Analytics table.

Syntax

PROFILE {<FIELDS> *numeric_field* <...*n*>|<FIELDS> ALL} <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*>

Parameters

Name	Description
FIELDS *numeric_field* <...*n*> | FIELDS ALL	Specify individual fields to profile, or specify ALL to profile all numeric fields in the Analytics table.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.

Examples

Profiling a single field

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You profile the **Salary** field:

```
OPEN Employee_Payroll
PROFILE FIELDS Salary
```

The command generates the following output:

Field Name	Total Value	Absolute Value	Minimum	Maximum
SALARY	1,152,525	1,152,525	15,340	52,750

Remarks

Statistics displayed in output

The following statistics are displayed for each numeric field or numeric expression specified for the command:

- total value
- absolute value
- minimum value
- maximum value

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

QUIT command

Ends the current session and closes Analytics.

Syntax

```
QUIT
```

Examples

Check if a file exists and close Analytics if it does not

You have created a script for others to run, but if a required file does not exist, you want to close Analytics.

The example below checks if the required `Inventory.csv` file exists, and closes Analytics if it does not:

```
IF FILESIZE("Inventory.csv") = -1 QUIT
```

Automatically close Analytics after a script completes

The script below summarizes the Inventory table, and produces output results, then automatically closes Analytics:

```
OPEN Inventory
SUMMARIZE ON Location ProdCls SUBTOTAL Value TO "Inventory_value_by_location_class.FIL"
PRESORT CPERCENT
QUIT
```

Remarks

Changes are saved

When QUIT executes, any Analytics tables that are open are saved and closed before quitting.

If you modified the active view or a script and did not save the changes, Analytics prompts you to save the changes before quitting.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RANDOM command

Generates a set of random numbers.

Syntax

RANDOM NUMBER *n* <SEED *seed_value*> MINIMUM *min_value* MAXIMUM *max_value* <COLUMNS *n*> <UNIQUE> <SORTED> <TO {SCREEN|*filename*}> <APPEND>

Parameters

Name	Description
NUMBER *n*	The size of the set of random numbers to be generated. A maximum of 32767 numbers can be generated.
SEED *seed_value* optional	The value used to initialize the random number generator. If you specify a seed value, it can be any number. Each unique seed value results in a different set of random numbers. If you respecify the same seed value, the same set of random numbers is generated. Regenerating the same set of random numbers can be required if you need to replicate analysis. ◦ **Seed value** - explicitly specify a seed value, and save the value, if you want the option of replicating a particular set of random numbers. ◦ **No seed value** - enter a seed value of '0', or leave the seed value blank, if you want Analytics to randomly select a seed value.
MINIMUM *min_value*	The smallest possible number in the set of random numbers. Any valid numeric value or expression is allowed.
MAXIMUM *max_value*	The greatest possible number in the set of random numbers. Any valid numeric value or expression is allowed.
COLUMNS *n* optional	The number of columns used to display the set of random numbers. If you omit COLUMNS, the default is 6 columns.
UNIQUE optional	Include only unique numbers in the set of random numbers. If you omit UNIQUE, duplicate values are allowed in the set of random numbers. **Note** Do not specify UNIQUE when the specified size of the set of random numbers exceeds 75 percent of the range between MINIMUM and MAXIMUM. Doing so can result in too many random number selections being discarded.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
SORTED optional	Displays the set of random numbers in ascending order. If you omit SORTED, the numbers are displayed in the order in which they are randomly selected.
TO SCREEN | *filename* optional	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"` If you omit TO, the set of random numbers is output to screen.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.

Examples

Generate a text file with 100 random numbers

You want to pull 100 hard copy files at random from a set of files with numbering that ranges from 10,000 to 20,000.

You can use the RANDOM command to generate a text file with 100 random numbers between 10,000 and 20,000. You then pull the hard copy files that match the random numbers. The numbers are arranged in 10 columns, are unique, and are sorted in ascending order:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
RANDOM NUMBER 100 SEED 45387 MINIMUM 10000 MAXIMUM 20000 COLUMNS 10 UNIQUE
SORTED TO "Random_Numbers.txt"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Random number algorithm

The RANDOM command uses the default Analytics random number algorithm. Unlike the SAMPLE command, the RANDOM command does not have the option of using the Mersenne-Twister random number algorithm.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RCOMMAND command

Passes an Analytics table to an external R script as a **data frame** and creates a new table in the Analytics project using output from the external R script.

Syntax

RCOMMAND FIELDS *field <...n>* RSCRIPT *path_to_script* TO *table_name* <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <KEEPTITLE> <SEPARATOR *character*> <QUALIFIER *character*> <OPEN>

Parameters

Name	Description
FIELDS *field_name <...n>*	The fields from the source Analytics table, or the expressions, to include in the data frame that is sent to the R script. Depending on the edition of Analytics that you are using, you may encounter errors when sending data containing some special characters to R: ◦ **non-Unicode** - "\" ◦ **Unicode** - "ÿ" or "Ŝ" ◦ **Both** - box drawing characters such as blocks, black squares, and vertical broken bars **Note** Mixed language data is also not supported, for example a table containing both Japanese and Chinese characters.
RSCRIPT *path_to_script*	The full or relative path to the R script on the file system. Enclose *path_to_script* in quotation marks.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. The output table is created from the data frame or matrix that the R script returns.
IF *test* optional	A condition that must be met to process the current record. The data frame passed to the R script contains only those records that meet the condition.
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default. **Caution** There is a known issue in the current version with NEXT when running the RCOMMAND. Avoid using this option as the record reference may reset to the first record regardless of which record is selected.
KEEPTITLE optional	Treat the first row of data as field names instead of data. If omitted, generic field names are used. This option is required if you want to retrieve data using column names in the R script.
SEPARATOR *character* optional	The character to use as the separator between fields. You must specify the character as a quoted string. The default character is a comma. **Note** Avoid using any characters that appear in the input fields. If the SEPARATOR character appears in the input data, the results may be affected.
QUALIFIER *character* optional	The character to use as the text qualifier to wrap and identify field values. You must specify the character as a quoted string. The default character is a double quotation mark.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note Avoid using any characters that appear in the input fields. If the QUALIFIER character appears in the input data, the results may be affected.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.

Examples

Getting R up and running (Hello world)

You create a hello world script to test your connection between Analytics and R:

Analytics command

```
RCOMMAND FIELDS "Hello", ", world!" TO "r_result" RSCRIPT "C:\scripts\r_scripts\analysis.r"
```

R script (analysis.r)

```
srcTable<-acl.readData()

# create table to send back to ACL
output<-data.frame(
 c(srcTable[1,1]),
 c(srcTable[1,2])
)

# add column names and send table back to ACL
colnames(output) <- c("Greeting","Subject")
acl.output<-output
```

Accessing field data using row and column coordinates

You send a number of invoice fields to an R script for analysis outside Analytics:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics command

```
RCOMMAND FIELDS Department_Code Invoice_Amount Invoice_Date Invoice_Number Vendor_
Number TO "r_result" RSCRIPT "C:\scripts\r_scripts\analysis.r"
```

R script (analysis.r)

```
# Retrieves invoice number from second row of data frame in R script
srcTable<-acl.readData()
srcTable[2,4]
```

Accessing field data using column names

You send a number of invoice fields to an R script for analysis outside Analytics. You use the KEEPTITLE option so that columns can be referenced by name in R:

Analytics command

```
RCOMMAND FIELDS Department_Code Invoice_Amount Invoice_Number TO "r_result" RSCRIPT
"C:\scripts\r_scripts\analysis.r" KEEPTITLE
```

R script (analysis.r)

```
# Retrieves invoice number from second row of data frame in R script
srcTable<-acl.readData()
srcTable["2","Invoice_Number"]
```

Sending invoice records that exceed 1000.00 value to R script

You send a number of invoice fields to an R script for analysis outside Analytics. You use IF to limit the records sent to R. Only those records with an invoice amount exceeding 1000.00 are sent:

Analytics command

```
RCOMMAND FIELDS Department_Code Invoice_Amount Invoice_Number TO "r_result" IF Invoice_
Amount > 1000.00 RSCRIPT "C:\scripts\r_scripts\analysis.r" KEEPTITLE
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

R script (analysis.r)

```
# Retrieves invoice number from second row of data frame in R script
srcTable<-acl.readData()
srcTable["2","Invoice_Number"]
```

Sending invoice records and returns multiplied invoice amounts

You send a number of invoice fields to an R script for analysis outside Analytics. The R script performs a single action against every cell in the named column:

Analytics command

```
RCOMMAND FIELDS Department_Code Invoice_Amount Invoice_Number TO "r_result" RSCRIPT
"C:\scripts\r_scripts\analysis.r" KEEPTITLE
```

R script (analysis.r)

```
# Returns slice of ACL table with value doubled
srcTable<-acl.readData()
acl.output<-srcTable["Invoice_Amount"] * 2
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Referencing Analytics data in the R script

The Analytics table is passed to the script as an R **data frame**. Data frames are tabular data objects that may contain columns of different modes, or types, of data.

To work with the data frame created by Analytics in an R script, invoke the `acl.readData()` function and store the returned data frame in a variable:

```
# stores the Analytics table in a data frame called myTable that can be referenced throughout the script
myTable<-acl.readData()
```

To retrieve data from a cell in the data frame, you can use one of the following approaches:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- Using row and column coordinates:

```
# Retrieves the value in the first row and second column of the data frame
myTable[1,2]
```

> **Note**
>
> Coordinates are based on the order of fields specified in the command, not the table layout or view that is currently open.

- Using row and column names:

```
# Retrieves the value in the first row and "myColumnTitle" column of the data frame
myTable["1","myColumnTitle"]
```

You must specify the **KEEPTITLE** option of the command to use column names.

Rows are named "1", "2", "3", and increment accordingly. You may also use a combination of names and coordinates.

Passing data back to Analytics

To return a data frame or matrix back to Analytics and create a new table, use the following syntax:

```
# Passes myNewTable data frame back to Analytics to create a new table
acl.output<-myNewTable
```

> **Note**
>
> You must return a data frame or a matrix to Analytics when the R script terminates. Ensure the columns in the data frame or matrix contain only atomic values and not lists, matrices, arrays, or non-atomic objects. If the values cannot be translated into Analytics data types, the command fails.

Data type mappings

Analytics data types are translated into R data types using a translation process between the Analytics project and the R script:

Analytics data type	R data type(s)
Logical	Logical
Numeric	Numeric
Character	Character

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics data type	R data type(s)
Datetime	Date, POSIXct, POSIXlt

Performance and file size limits

The time it takes to run your R script and process the data that is returned increases for input data exceeding 1 GB. R does not support input files of 2 GB or higher.

The number of records sent to R also affects performance. For two tables with the same file size but a differing record count, processing the table with fewer records is faster.

Handling multi-byte character data

If you are sending data to R in a multi-byte character set, such as Chinese, you must set the system locale appropriately in your R script. To successfully send a table of multi-byte data to R, the first line of the R script must contain the following function:

```
# Example that sets locale to Chinese
Sys.setlocale("LC_ALL","Chinese")
```

For more information about Sys.setlocale( ), see the R documentation.

R log file

Analytics logs R language messages to an `aclrlang.log` file in the project folder. Use this log file for debugging R errors.

> **Tip**
>
> The log file is available in the Results folder of Analytics Exchange analytic jobs.

Running R scripts on AX Server

If you are writing an analysis app to run on AX Server and you want to work with external R scripts:

1. Upload the file as a related file with the analysis app.
2. Use the FILE analytic tag to identify the file(s).
3. Reference the file(s) using the relative path `./filename.r`.

> **Note**
>
> Using a related file ensures that the TomEE application server account has sufficient permissions to access the file when running R with Analytics Exchange.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

REFRESH command

Updates the data in an Analytics table from its associated data source.

Syntax

```
REFRESH <table_name> <PASSWORD num>
```

Parameters

Name	Description
table_name optional	The name of the Analytics table to refresh. If you do not specify a *table_name*, the open table is refreshed.
PASSWORD *num* optional	The password definition to use. You do not use PASSWORD *num* to prompt for, or specify, an actual password. The password definition refers to a password previously supplied or set using the PASSWORD command, the SET PASSWORD command, or the PASSWORD analytic tag. *num* is the number of the password definition. For example, if two passwords have been previously supplied or set in a script, or when scheduling an analytic, PASSWORD 2 specifies that password #2 is used. For more information about supplying or setting passwords, see: • "PASSWORD command" on page 360 • "SET command" on page 418 • PASSWORD analytic tag **Note** The password is used to access the original source data system. You cannot use REFRESH with a password for file-based data sources, with the exception of PDFs.

Examples

Refreshing a table with no password required

If a password is not required for the data source, just specify the REFRESH command and the name of the Analytics table to refresh.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
REFRESH Invoices
```

Refreshing a table with a password in an interactive script

If you are creating an interactive script, you can prompt the user for the password:

```
PASSWORD 1 "Enter your password:"
REFRESH Invoices PASSWORD 1
```

If you are refreshing a table originally imported from a password-protected data source using the ACCESSDATA command, the password prompt is automatic and does not need to be separately specified:

```
REFRESH Invoices
```

Refreshing a table with a password in a non-interactive script

You can set the password in a script if you do not want to prompt the user for the value:

```
SET PASSWORD 1 TO "password"
REFRESH Invoices PASSWORD 1
```

The disadvantage of this method is that the password appears as clear text in the script.

Refreshing a table with a password in an AX Server analytic

If you are creating an AX Server analytic, you can prompt the user for the password when the analytic is scheduled, or run ad-hoc:

```
COMMENT
//ANALYTIC Refresh Table
//PASSWORD 1 "Enter your password:"
END
REFRESH Invoices PASSWORD 1
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

How it works

The REFRESH command updates the contents of a table by re-running the IMPORT command, or the ACCESSDATA command, initially used to define and import the table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

REFRESH updates table content only

The REFRESH command updates only the content of existing fields in an Analytics table. It cannot update an Analytics table layout.

You cannot use REFRESH if the structure of the source data has changed – for example, if fields have been added or removed. You must re-import the data.

Data sources that support refreshing

You can use the REFRESH command to update the content of an Analytics table created using any of the following commands:

- IMPORT ACCESS
- IMPORT DELIMITED
- IMPORT EXCEL
- IMPORT ODBC (legacy ODBC command)
- IMPORT PDF
- IMPORT PRINT
- IMPORT SAP
- IMPORT XBRL
- IMPORT XML
- ACCESSDATA (ODBC data sources)

REFRESH and ACCESSDATA

The following guidelines apply when refreshing a table imported from an ODBC data source using the ACCESSDATA command.

- **Open table** – If the table is open when you refresh it, you temporarily need disk space equal to twice the size of the table. If you have limited disk space, close the table first before refreshing it.
- **Analytics 12** – Tables that were imported using the ACCESSDATA command in version 12 of Analytics are not refreshable, even if you are using a more recent version of Analytics.

  If you want to be able to refresh these tables, re-import them using Analytics 12.5 or later.

REFRESH and passwords

You can use the REFRESH command with password-protected data sources that exist in a database, or in a cloud data service.

You cannot use the REFRESH command with password-protected file-based data sources, such as Excel files. The one exception is password-protected PDFs.

REFRESH and the Analysis App window

Do not use the REFRESH command in scripts that you intend to run in the Analysis App window.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Depending on how a table is imported, refreshing the data in the table is either not supported, or produces unpredictable results, if attempted in the Analysis App window.

If you want to refresh data as part of a script run in the Analysis App window, use either an IMPORT command, or the ACCESSDATA command, and overwrite the table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RENAME command

Renames an Analytics project item or a file.

Syntax

RENAME *item_type name* <AS|TO> *new_name* <OK>

Parameters

Name	Description
item_type name	The type and name of the project item or file that you want to rename. **Note** In most cases, you cannot rename an item or file if it is active, open, or in use. Specify one of the following valid types: ◦ **FIELD** - physical data field, computed field, or variable • The table containing the field must be open. However, the active view cannot include the field. • You cannot rename a field that is referenced by a computed field. ◦ **FORMAT** - Analytics table ◦ **INDEX** - index ◦ **REPORT** - report or view ◦ **WORKSPACE** - workspace ◦ **SCRIPT (or BATCH)** - script ◦ **DATA** - Analytics data file (.fil) ◦ **FILE** - data file in the file system ◦ **LOG** - Analytics log file (.log) ◦ **TEXT** - text file
AS | TO *new_name*	The new name for the project item or file. **Note** Length limitations apply to most Analytics project item names. For more information, see Character and size limits in Analytics.
OK optional	Deletes or overwrites items without asking you to confirm the action.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Renaming a field

You need to rename the **ProdNo** field to **ProdNum**. You use OK to perform the action without additional confirmation:

```
OPEN Inventory
RENAME FIELD ProdNo AS ProdNum OK
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

REPORT command

Formats and generates a report based on the open Analytics table.

Syntax

REPORT <ON *break_field* <PAGE> <NODUPS> <WIDTH *characters*> <AS *display_name*>> <...*n*> FIELD *other_fields* <WIDTH *characters*> <AS *display_name*> <...*n*> <SUPPRESS> <NOZEROS> <LINE *n other_fields*> <PRESORT <*sort_field*>> <...*n*> <SUMMARIZED> <SKIP *n*> <EOF> <TO {SCREEN|PRINT|*filename* <HTML>}> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <HEADER *header_text*> <FOOTER *footer_text*> <APPEND>

Parameters

Name	Description
ON *break_field* PAGE NODUPS WIDTH *characters* AS *display_name* <...*n*> optional	The character field or fields used to break the report into sections. A new report section and subtotal is created each time the value in *break_field* changes. Breaking reports into sections can make them easier to scan. ◦ ***break_field*** - the field to use as a break field To run a report based on a view (DO REPORT), the break field must be the leftmost character field in the view. ◦ **PAGE** - inserts a page break when the break field value changes ◦ **NODUPS** - suppresses duplicate display values in the break field For example, if the customer name is listed for each invoice record, you can make the report more readable by listing only the first instance of each customer name. ◦ **WIDTH *characters*** - the output length of the field in characters ◦ **AS *display_name*** - the display name (alternate column title) for the field in the report Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title. If you want the display name to be the same as the field name, or an existing display name in the source table, do not use AS. **Note** You must specify ON to use *break_field* , PAGE, NODUPS, or PRESORT.
FIELD *other_fields* WIDTH *characters* AS *display_name* <...*n*>	The fields to be included in the report. ◦ **WIDTH *characters*** - the output length of the field in characters ◦ **AS *display_name*** - the display name (alternate column title) for the field in the report Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title. If you want the display name to be the same as the field name, or an existing display name in the source table, do not use AS.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	The SUBTOTAL and ACCUMULATE keywords are synonyms for FIELD, and have been deprecated. All numeric fields are automatically subtotaled. **Note** Break fields are automatically included in the report and do not need to be specified as *other_fields*.
SUPPRESS optional	Excludes blank detail lines from the report.
NOZEROS optional	Substitutes blank values for zero values in the field. For example, if a report includes a large number of zero values in a field, the report is easier to read if it only displays non-zero values.
LINE *n other_fields* optional	Specifies the number of output lines in the column and the fields to appear on the line number *n*. If no value is specified, the column defaults to a single line. The value of *n* must be between 2 and 60 inclusive. Column headings on the report are determined solely by the fields on the first line. *other_fields* specifies appropriate fields or expressions for the report.
PRESORT *sort_field* <...*n*> optional	◦ Sorts *break_field* , if one or more break fields are specified. ◦ Sorts *sort_field*, if one or more sort fields are specified. PRESORT does not sort the fields listed as *other_fields* unless they are also listed as *sort_field*.
SUMMARIZED optional	Produces a report with subtotals and totals only, and no detail lines. Subtotals are generated for the unique break field values. If SUMMARIZED is not specified, Analytics produces a report that includes detail lines, as well as subtotals for each of the specified key break fields.
SKIP *n* optional	Inserts blank lines between detail lines in the report. *n* must be an integer specifying the number of lines to insert. For example, `SKIP 1` produces a double-spaced report.
EOF optional	Execute the command one more time after the end of the file has been reached. This ensures that the final record in the table is processed when inside a GROUP command. Only use EOF if all fields are computed fields referring to earlier records.
TO SCREEN | PRINT| *filename* <HTML> optional	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"` ◦ **PRINT** - sends the results to the default printer By default, reports output to a file are saved as ASCII text files. Specify `HTML` if you want to output the report as an HTML file (.htm). If you omit TO, the report is output to screen.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.

Examples

Generating an HTML report

You generate a report from the **Ar** table and output the report to a formatted HTML file:

```
OPEN Ar
REPORT ON No FIELD Due Type Amount TO "C:\Reports\AR.htm" HTML
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RETRIEVE command

Retrieves the result of a *Direct Link* query submitted for background processing.

Syntax

```
RETRIEVE table_name PASSWORD num
```

Parameters

Name	Description
table_name	The name of the table originally created in Analytics by the Direct Link query. The table must already exist before you use RETRIEVE.
PASSWORD *num*	The password definition to use. You do not use PASSWORD *num* to prompt for, or specify, an actual password. The password definition refers to a password previously supplied or set using the PASSWORD command, the SET PASSWORD command, or the PASSWORD analytic tag. *num* is the number of the password definition. For example, if two passwords have been previously supplied or set in a script, or when scheduling an analytic, PASSWORD 2 specifies that password #2 is used. For more information about supplying or setting passwords, see: • "PASSWORD command" on page 360 • "SET command" on page 418 • PASSWORD analytic tag **Note** The password is used to access the SAP system.

Examples

Retrieving the Background mode query result

You set the password and then retrieve the Background mode query result for an Analytics table named **DD02T_Data**:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
SET PASSWORD 1 TO "pwd"
RETRIEVE DD02T_Data PASSWORD 1
```

Remarks

Before you begin

This command is only supported if Direct Link is installed and configured.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SAMPLE command

Draws a sample of records using either the record sampling or monetary unit sampling method.

Record samplingMonetary unit sampling

Syntax

> **Note**
>
> The syntax does not include filtering (IF statements) or scope parameters because applying these options compromises the validity of a sample.

Fixed interval selection method

SAMPLE <ON> RECORD INTERVAL *interval_value* <FIXED *initial_value*> {RECORD|FIELDS *field_name* <*...n*>} TO *table_name* <LOCAL> <OPEN> <APPEND>

Cell selection method

SAMPLE <ON> RECORD CELL INTERVAL *interval_value* <RANDOM *seed_value*> {RECORD|FIELDS *field_name* <*...n*>} TO *table_name* <LOCAL> <OPEN> <APPEND> <MERSENNE_TWISTER>

Random selection method

SAMPLE <ON> RECORD NUMBER *sample_size* <RANDOM *seed_value*> <ORDER> {RECORD|FIELDS *field_name* <*...n*>} TO *table_name* <LOCAL> <OPEN> <APPEND> <MERSENNE_TWISTER>

Parameters

> **Note**
>
> Do not include thousands separators when you specify values.

Name	Description
ON RECORD	Use record sampling.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
INTERVAL *interval_value* FIXED *initial_value* | CELL INTERVAL *interval_value* | NUMBER *sample_size*	**INTERVAL *interval_value* FIXED *initial_value*** Use the **fixed interval** selection method. An initial record is selected and all subsequent selections are a fixed interval or distance apart - for example, every 20th record after the initial selection. ◦ **INTERVAL *interval_value*** - specify the interval value that was generated by calculating the sample size ◦ **FIXED *initial_value*** - specify the initial record number selected If you specify an *initial_value* of zero ('0'), or omit FIXED, Analytics randomly selects the initial record. **CELL INTERVAL *interval_value*** Use the **cell** selection method. The data set is divided into multiple equal-sized cells or groups, and one record is randomly selected from each cell. The *interval_value* dictates the size of each cell. Specify the interval value that was generated by calculating the sample size. **NUMBER *sample_size*** Use the **random** selection method. All records are randomly selected from the entire data set. Specify the sample size that was generated by calculating the sample size.
RANDOM *seed_value* optional	**Note** Cell and random selection methods only. The seed value to use to initialize the random number generator in Analytics. If you specify a value of zero ('0'), or omit RANDOM, Analytics randomly selects the seed value.
ORDER optional	**Note** Random selection method only. You can only use ORDER when specify FIELDS. Adds the ORDER field to the output results. This field displays the order in which each record is randomly selected.
RECORD | FIELDS *field_name <...n>*	◦ **RECORD** - the entire record is included in the output table ◦ **FIELDS** - individual fields, rather than the entire record, are included in the output table Specify the field(s) or expressions to include. If you specify multiple fields, they must be separated by spaces.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
MERSENNE_TWISTER optional	**Note** Cell and random selection methods only. The random number generator in Analytics uses the Mersenne-Twister algorithm. If you omit MERSENNE_TWISTER, the default Analytics algorithm is used. **Note** You should only use the default Analytics algorithm if you require backward compatibility with Analytics scripts or sampling results created prior to Analytics version 12.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Draw a record sample

You are going to use record sampling to estimate the rate of deviation from a prescribed control in an account containing invoices.

After calculating a statistically valid sample size, you are ready to draw the sample. You are going to use the random selection method.

The example below:

- Samples the open Analytics table
- Uses the random selection method with a seed value of 123456
- Specifies a sample size of 95 records
- Includes only specified fields in the output table
- Specifies that the random number generator in Analytics uses the Mersenne-Twister algorithm

```
SAMPLE ON RECORD RANDOM 123456 NUMBER 95 FIELDS RefNum CustNum Amount Date
Type TO "Ar_record_sample" OPEN MERSENNE_TWISTER
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Syntax

> **Note**
>
> The syntax does not include filtering (IF statements) or scope parameters because applying these options compromises the validity of a sample.

Fixed interval selection method

```
SAMPLE <ON> mus_numeric_field INTERVAL interval_value <FIXED initial_value> <CUTOFF top_
stratum_cutoff_value> <SUBSAMPLE> <NOREPLACEMENT> {RECORD|FIELDS field_name
<...n>} TO table_name <LOCAL> <OPEN> <APPEND>
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Cell selection method

SAMPLE <ON> *mus_numeric_field* CELL INTERVAL *interval_value* <CUTOFF *top_stratum_cutoff_value*> <RANDOM *seed_value*> <SUBSAMPLE> <NOREPLACEMENT> {RECORD|FIELDS *field_name* <...*n*>} TO *table_name* <LOCAL> <OPEN> <APPEND> <MERSENNE_TWISTER>

Random selection method

SAMPLE <ON> *mus_numeric_field* NUMBER *sample_size* POPULATION *absolute_value* <RANDOM *seed_value*> <SUBSAMPLE> <NOREPLACEMENT> <ORDER> {RECORD|FIELDS *field_name* <...*n*>} TO *table_name* <LOCAL> <OPEN> <APPEND> <MERSENNE_TWISTER>

Parameters

> **Note**
>
> Do not include thousands separators when you specify values.

Name	Description
ON *mus_numeric_field*	Use monetary unit sampling (MUS). *mus_numeric_field* is the numeric field or expression to use as the basis for the sampling.
INTERVAL *interval_value* FIXED *initial_value* | CELL INTERVAL *interval_value* | NUMBER *sample_size* POPULATION *absolute_value*	**INTERVAL *interval_value* FIXED *initial_value*** Use the **fixed interval** selection method. An initial monetary unit is selected and all subsequent selections are a fixed interval or distance apart - for example, every 5000th monetary unit after the initial selection. ◦ **INTERVAL *interval_value*** - specify the interval value that was generated by calculating the sample size ◦ **FIXED *initial_value*** - specify the initial monetary unit selected If you specify an *initial_value* of zero ('0'), or omit FIXED, Analytics randomly selects the initial monetary unit. **CELL INTERVAL *interval_value*** Use the **cell** selection method. The data set is divided into multiple equal-sized cells or groups, and one monetary unit is randomly selected from each cell. The *interval_value* dictates the size of each cell. Specify the interval value that was generated by calculating the sample size.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	NUMBER *sample_size* POPULATION *absolute_value* Use the **random** selection method. All monetary units are randomly selected from the entire data set. ◦ **NUMBER *sample_size*** - specify the sample size that was generated by calculating the sample size ◦ **POPULATION *absolute_value*** - specify the total absolute value of *mus_numeric_field*, which is the population from which the sample will be selected
CUTOFF *top_stratum_cutoff_value* optional	**Note** Fixed interval and cell selection methods only. A top stratum cutoff value. Amounts in the *mus_numeric_field* greater than or equal to the cutoff value are automatically selected and included in the sample. If you omit CUTOFF, a default cutoff value equal to the *interval_value* is used.
RANDOM *seed_value* optional	**Note** Cell and random selection methods only. The seed value to use to initialize the random number generator in Analytics. If you specify a value of zero ('0'), or omit RANDOM, Analytics randomly selects the seed value.
SUBSAMPLE optional	**Note** You can only use SUBSAMPLE when specify FIELDS. Adds the SUBSAMPLE field to the output results. If each amount in a sample field represents a total of several separate transactions, and you want to perform audit procedures on only one transaction from each sampled total amount, you can use the values in the SUBSAMPLE field to randomly select the individual transactions. For more information, see Performing monetary unit sampling.
NOREPLACEMENT optional	The same record is not selected more than once. As a result, the sample may contain fewer records than calculated by the SIZE command. If NOREPLACEMENT is omitted, or if you specify REPLACEMENT, records can be selected more than once.
ORDER optional	**Note** Random selection method only. You can only use ORDER when specify FIELDS. Adds the ORDER field to the output results. This field displays the order in which each record is randomly selected.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
RECORD | FIELDS *field_name* <...n>	◦ **RECORD** - the entire record is included in the output table ◦ **FIELDS** - individual fields, rather than the entire record, are included in the output table Specify the field(s) or expressions to include. If you specify multiple fields, they must be separated by spaces.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
MERSENNE_TWISTER optional	**Note** Cell and random selection methods only. The random number generator in Analytics uses the Mersenne-Twister algorithm. If you omit MERSENNE_TWISTER, the default Analytics algorithm is used. **Note** You should only use the default Analytics algorithm if you require backward compatibility with Analytics scripts or sampling results created prior to Analytics version 12.

Examples

Draw a monetary unit sample

You are going to use monetary unit sampling to estimate the total amount of monetary misstatement in an account containing invoices.

After calculating a statistically valid sample size, you are ready to draw the sample. You are going to use the fixed interval selection method.

The example below:

- Samples the open Analytics table based on a transaction amount field
- Uses the fixed interval selection method with an interval value of $6,283.33
- Specifies that the first record selected contains the 100,000th monetary unit (the number of cents in $1,000)
- Uses a top stratum cutoff of $5,000
- Includes the entire record in the output table

```
SAMPLE ON Amount INTERVAL 6283.33 FIXED 1000.00 CUTOFF 5000.00 RECORD TO "Ar_monetary_unit_sample" OPEN
```

Remarks

Note

For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SAVE command

Copies an Analytics table and saves it with a different name, or saves an Analytics project.

Syntax

To create a copy of an Analytics table and save it with a different name:

```
SAVE new_table FORMAT ACL_table
```

To save changes to the current project:

```
SAVE
```

Parameters

Name	Description
new_table	The name of the new Analytics table to create and save. **Note** Table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
FORMAT *ACL_table*	The name of the existing Analytics table. Use the name of the table layout, not the name of an associated data file.

Examples

Creating a new table based on an existing one

You create a new table called **Payables_March** based on the existing table **Payables_master**. **Payables_March** can then be linked to the March payables data file:

```
SAVE Payables_March FORMAT Payables_master
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

How it works

SAVE FORMAT produces a result similar to copying and pasting an Analytics table in the **Overview** tab in the **Navigator**. A new Analytics table is created and associated to the same data file or data source as the original table.

If required, you can link the newly created table to a different data source.

Using SAVE to avoid prompts

At certain points, Analytics prompts you to save changes to the current project. To avoid interruptions in the execution of scripts, you can use the SAVE command to save changes before Analytics prompts you.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SAVE LAYOUT command

Saves an Analytics table layout to an external table layout file (.layout), or saves table layout metadata to an Analytics table.

> **Note**
>
> Prior to version 11 of Analytics, external table layout files used an .fmt file extension. You can still save a table layout file with an .fmt extension by manually specifying the extension.

Syntax

```
SAVE LAYOUT {FILE|TABLE} TO {file_name|table_name}
```

Parameters

Name	Description
FILE | TABLE	◦ **FILE** – save an Analytics table layout to an external table layout file (.layout) ◦ **TABLE** – save table layout metadata to an Analytics table (.fil)
TO *file_name* | *table_name*	The name of the output file, and the output location: ◦ ***filename*** – the name of the .layout file Specify the *filename* as a quoted string. For example: `TO "Ap_Trans.layout"`. The .layout file extension is used by default, so specifying it is optional. By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Ap_Trans.layout"` • `TO "Table Layouts\Ap_Trans.layout"` **Note** Limit the table layout name to 64 alphanumeric characters, not including the .layout extension, to ensure that the name is not truncated when the table layout is imported back into Analytics. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. ◦ ***table_name*** –the name of the Analytics table and .fil file Specify the *table_name* as a quoted string. For example: `TO "Ap_Trans_layout_metadata.fil"`.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	The .fil file extension is used by default, so specifying it is optional. By default, the table data file (.fil) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • TO "C:\Ap_Trans_layout_metadata.fil" • TO "Layout Metadata\Ap_Trans_layout_metadata.fil" **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.

Examples

Saving a table layout to an external table layout file (.layout)

The following examples save the table layout used by the open table to an external table layout file called **Ap_Trans.layout**:

Here, the table layout file is saved in the Analytics project folder:

```
SAVE LAYOUT FILE TO Ap_Trans.layout
```

Here, the table layout file is saved in the specified folder:

```
SAVE LAYOUT FILE TO "C:\ACL_DATA\AP Audit 2013\Ap_Trans.layout"
```

Saving a copy of table layout metadata to a new Analytics table

The following examples save a copy of the metadata in the table layout used by the open table to a new Analytics table called **Ap_Trans_layout_metadata**.

Here, the new Analytics table is saved in the Analytics project folder:

```
SAVE LAYOUT TABLE TO Ap_Trans_layout_metadata
```

Here, the new Analytics table is saved in the specified folder:

```
SAVE LAYOUT TABLE TO "C:\ACL_DATA\AP Audit 2013\Ap_Trans_layout_metadata"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

SAVE LAYOUT file vs table

The SAVE LAYOUT command is used for two different purposes:

- **FILE** - saves the table layout of the open Analytics table to an external table layout file with a `.layout` extension
- **TABLE** - extracts the metadata from the table layout of the open Analytics table and saves it to a new Analytics table

SAVE LAYOUT FILE

How it works

SAVE LAYOUT FILE saves the table layout of the open Analytics table to an external table layout file with a .layout extension.

A table layout contains metadata that provides a structured interpretation of the raw data in an associated source data file. A table layout does not contain any source data itself.

When to use SAVE LAYOUT FILE

Saving a table layout as a .layout file makes the table layout and its metadata portable and reusable.

The .layout file can be imported into any Analytics project and associated with a matching source data file. The data elements in the source data file must match the field definitions specified by the table layout metadata.

For example, you could save the table layout of a transactions file from March, and associate it with a source data file containing transactions from April, assuming the structure of the data in the March and April source data files is identical. Used in this manner, .layout files can save you the labor of creating a new table layout from scratch.

For more information about the structure of Analytics tables, see the Analytics Help.

SAVE LAYOUT TABLE

How it works

SAVE LAYOUT TABLE extracts the metadata from the table layout of the open Analytics table and saves it to a new Analytics table.

The new table is not the table layout itself, but rather a regular Analytics table that contains a summary of the table layout metadata for the original table. Having access to this summary in an Analytics script can allow you to make decisions in the script based on the information.

For each field in the original table, the following pieces of table layout metadata are extracted into the new table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> The field names in the new table are always generated in English regardless of which localized version of Analytics you are using.

Field name in new table	Table layout metadata
field_name	The name of the field
data_type	The data type of the field
category	The data category of the field
start_position	The start position of the field
field_length	The length of the field
decimals	The number of decimal places in the field (numeric fields only)
format	The format of the field (datetime and numeric fields only)
alternate_title	The alternate column title of the field
column_width	The width of the column in the view

Additional details

Computed fields	Computed fields are included in the extracted metadata, but the expression used by the computed field, and any conditions, are not recorded. Start position, field length, and decimal places are also not recorded for computed fields.
Related fields	Related fields are not included because they are not part of the table layout.
Field-level filters Field notes	Field-level filters and field notes are not included.
Alternate column title Column width	The values recorded for alternate column title and column width are the ones specified in the table layout, not the view-level values that can be specified for columns.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SAVE LOG command

Saves the entire command log, or the log entries for the current Analytics session, to an external file.

Syntax

```
SAVE LOG <SESSION> AS filename {<ASCII>|HTML} <OK>
```

Parameters

Name	Description
SESSION optional	Only log entries for the current Analytics session are saved.
AS *filename*	The name of the output file. Specify the *filename* as a quoted string. For example: AS "Command Log". You can specify a file extension (.txt, or .htm or .html), but it is not required. By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: ◦ AS "C:\Command Log.TXT" ◦ AS "Results\Command Log.TXT"
ASCII | HTML	The format of the output file: ◦ **ASCII (or no keyword)** - a plain text ASCII file. ◦ **HTML** - an HTML file.
OK optional	If a file with the same name as *filename* already exists, it is overwritten without confirmation.

Examples

Save the command log from payables analysis

You have performed data analysis on the March payables file and you want to save the associated command log as part of your working papers.

The following example saves the entries from the current Analytics session to an HTML file. If a file with the same name already exists it is overwritten without confirmation:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
SAVE LOG SESSION AS "C:\Payables_March_Log.htm" HTML OK
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SAVE TABLELIST command

Saves a list of all tables in an Analytics project to an Analytics table or a CSV file.

Syntax

```
SAVE TABLELIST {FILE|TABLE} TO {table_name|file_name}
```

Parameters

Name	Description
FILE | TABLE	◦ **FILE** - saves the table list to a CSV file (.csv) ◦ **TABLE** - saves the table list to an Analytics table
TO *table_name* | *file_name*	The location to save the table list: ◦ ***table_name*** - the name of the output Analytics table and the associated .fil file when using TABLE The .fil file extension is used by default and does not need to be specified. The table is saved in the same folder as the Analytics project, and cannot be saved in any other folder. **Note** Analytics table names are limited to 64 alphanumeric characters. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. ◦ ***file_name*** - the name of the .csv file when using FILE The .csv file extension is used by default and does not need to be specified. You can specify an absolute or relative path to save the CSV file in an existing folder other than the folder containing the Analytics project. If you specify a relative path, it is relative to the Analytics working directory. You must specify values as quoted strings if they contain any spaces.

Examples

Creating a new table

You create a new table in the Analytics project called **Table_list_complete**:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
SAVE TABLELIST TABLE TO Table_list_complete
```

Creating a CSV file

You create a new CSV file in the `C:\ACL Data` folder called `Table_list_complete.csv`:

```
SAVE TABLELIST FILE TO "C:\ACL Data\Table_list_complete"
```

Remarks

Output columns

The output Analytics table or CSV file contains three columns:

- **table_name** - the name of the Analytics table layout
- **type** - an indication whether the Analytics table is a local table or a server table
- **Data_file_Path** - the full path to the source data file

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SAVE WORKSPACE command

Creates and saves a workspace.

Syntax

```
SAVE WORKSPACE workspace_name {field_name <...n>}
```

Parameters

Name	Description
workspace_name	The name of the workspace to create and add to the current Analytics project.
field_name <...n>	The name of the field to add to the workspace. You can include multiple field names separated by spaces.

Example

Activating a workspace

You create a workspace called **Inventory_margin** with two computed fields from the **Metaphor_Inventory_2002** table. Then you activate the workspace so that the fields are available in the **Inventory** table:

```
OPEN Metaphor_Inventory_2002
SAVE WORKSPACE Inventory_margin Gross_unit_margin Percent_unit_margin
OPEN Inventory
ACTIVATE WORKSPACE Inventory_margin OK
```

Remarks

Field names used to create computed fields must match

The names of any fields used in expressions that create a computed field that is saved in a workspace must match the names of the fields in the table that uses the workspace.

For example, if a workspace contains the computed field `Value=Sale_price*Quantity`, the active table must also contain fields called **Sale_price** and **Quantity**.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SEEK command

Searches an indexed character field for the first value that matches the specified character expression or character string.

Syntax

```
SEEK search_expression
```

Parameters

Name	Description
search_expression	The character expression to search for. You can use any valid character expression, character variable, or quoted string. *search_expression* is case-sensitive, and can include leading spaces, which are treated like characters.

Examples

Locate the first value in a field that matches a character variable

The Card_Number field has been defined as a character field and is indexed in ascending order.

The example below locates the first value in the field that exactly matches, or starts with, the value contained in the *v_card_num* variable.

```
INDEX ON Card_Number TO "CardNum" OPEN
SET INDEX TO "CardNum"
SEEK v_card_num
```

Locate the first value in a field that matches a character string

The Card_Number field has been defined as a character field and is indexed in ascending order.

The example below locates the first value in the field that exactly matches, or starts with, the character literal "AB-123":

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
INDEX ON Card_Number TO "CardNum" OPEN
SET INDEX TO "CardNum"
SEEK "AB-123"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

How it works

Use the SEEK command to move directly to the first record in a table containing the specified *search_expression* in the indexed character field.

- **If the *search_expression* is found** – the first matching record in the table is selected.
- **If the *search expression* is not found** – the message "No index matched key" is displayed, and the table is positioned at the first record with a greater value than the search expression.

  If there are no values in the indexed field greater than the search expression, the table is positioned at the first record.

Index required

To use SEEK to search a character field, you must first index the field in ascending order. If multiple character fields are indexed in ascending order, only the first field specified in the index is searched.

SEEK cannot be used to search non-character index fields, or character fields indexed in descending order.

Partial matching supported

Partial matching is supported. The search expression can be contained by a longer value in the indexed field. However, the search expression must appear at the start of the field to constitute a match.

The SEEK command is not affected by the **Exact Character Comparisons** option (SET EXACT ON/OFF).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SEQUENCE command

Determines if one or more fields in an Analytics table are in sequential order, and identifies out-of-sequence items.

Syntax

SEQUENCE <ON> {<FIELDS> *field* <D> <...*n*>|<FIELDS> ALL} <UNFORMATTED> <ERRORLIMIT *n*> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <TO {SCREEN|*filename*|PRINT}> <APPEND> <HEADER *header_text*> <FOOTER *footer_text*> <PRESORT> <ISOLOCALE *locale_code*>

Parameters

Name	Description
ON FIELDS *field* D <...*n*> | FIELDS ALL	The fields or expressions to check for sequential order. Specify ALL to check all fields in the Analytics table. Include D to sort the key field in descending order. The default sort order is ascending.
UNFORMATTED optional	Suppresses page headings and page breaks when the results are output to a file.
ERRORLIMIT *n* optional	The number of errors allowed before the command is terminated. The default value is 10.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
TO SCREEN | *filename* | PRINT optional	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"` ◦ **PRINT** - sends the results to the default printer
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.
PRESORT optional	Sorts the table on the key field before executing the command. **Note** You cannot use PRESORT inside the GROUP command.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
ISOLOCALE *locale_code* optional	**Note** Applicable in the Unicode edition of Analytics only. The system locale in the format *language_country*. For example, to use Canadian French, enter fr_ca. Use the following codes: ◦ **language** - ISO 639 standard language code ◦ **country** - ISO 3166 standard country code If you do not specify a country code, the default country for the language is used. If you do not use ISOLOCALE, the default system locale is used.

Analytics output variables

Name	Contains
WRITE*n*	The total number of sequence errors identified by the command.

Examples

Testing for out of sequence employee IDs and hire dates

You write any sequence errors identified in the **EmployeeID** and **HireDate** fields to a text file:

```
SEQUENCE ON EmployeeID HireDate ERRORLIMIT 10 TO "SequenceErrors.txt"
```

Remarks

> **Note**
> For more information about how this command works, see the Analytics Help.

Using SEQUENCE inside a GROUP

If you use SEQUENCE inside a GROUP command, the command executes to avoid interfering with the processing of the group, but no further data sequence errors are reported.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SET command

Sets a configurable Analytics option.

> **Note**
>
> The SET command sets an Analytics option for the duration of the Analytics session only. This behavior applies whether you use the SET command in the Analytics command line or in an Analytics script.
>
> To set Analytics options so that they persist between Analytics sessions, you must use the **Options** dialog box. For more information, see Configuring ACL options.

Syntax

Syntax	Examples and remarks
SET BEEP *value*	SET BEEP 2 Specifies the number of beeps to sound when command processing is completed. The *value* parameter must be between 1 and 255.
SET CENTURY *value*	SET CENTURY 40 Specifies the start-of-century year for two-digit years. The *value* parameter must be from 0 to 99. Setting the start-of-century value to 40 means that two-digit years 40 to 99 are interpreted as 1940 to 1999, and two-digit years 00 to 39 are interpreted as 2000 to 2039.
SET CLEAN {ON | OFF}	SET CLEAN ON When this option is turned on, Analytics replaces invalid character data with blanks and invalid numeric data with zeros.
SET DATE <TO> {0 | 1 | 2 | *string*}	SET DATE "YYYY/MM/DD" Specifies how Analytics displays dates, and the date portion of datetimes, in views, reports, and exported files. ◦ SET DATE 0 sets the date to MM/DD/YYYY format ◦ SET DATE 1 sets the date to MM/DD/YY format ◦ SET DATE 2 sets the date to DD/MM/YY format ◦ SET DATE "<*string*>" sets the date to the custom format you specify

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Syntax	Examples and remarks
	When using the SET DATE command to specify custom date formats, you must use 'D' for Day, 'M' for Month, and 'Y' for Year, even if you have specified different date format characters in the **Options** dialog box. For example: `SET DATE "DD MMM YYYY"`
SET DELETE_FILE {ON | OFF}	`SET DELETE_FILE ON` Default setting: OFF Specify ON to automatically delete the associated data file when you delete a table layout. Specify OFF to prevent the associated data file from being deleted when you delete a table layout. You must include the underscore (_) in DELETE_FILE. Specifying SET DELETE_FILE, without any parameter, in the command line displays whether DELETE_FILE is currently on or off. **Caution** Use caution when turning this option on. It may be an original data file that is deleted along with the table. Data files are deleted outright. They are not sent to the Windows Recycle Bin.
SET DESIGNATION *value*	`SET DESIGNATION "Produced by ABC Corporation"` The *value* parameter is a quoted string that specifies the label to display at the top of each printed page.

SET ECHO {ON | NONE}	`SET ECHO NONE`
	`COM Commands and results in scripts excluded from log.`
	`SET ECHO ON`
	Specify NONE to stop writing commands and results in scripts to the Analytics command log. Specify ON to resume logging.
	The SET ECHO command applies only to the logging of commands and results in scripts. Commands performed through the user interface or issued from the command line, and any results they produce, are always logged, regardless of how ECHO is set.
	You can issue the SET ECHO NONE/ON command in a script or from the command line, but regardless of where you issue the command, it affects only the logging of commands and results in scripts.
	Specifying SET ECHO, without any parameter, in the command line displays whether the logging of commands and results in scripts is currently on or off.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Syntax</th><th>Examples and remarks</th></tr>
<tr><td>SET EXACT {ON | OFF}</td><td>
<pre>SET EXACT ON</pre>
Default setting: OFF
Controls how Analytics compares character fields, expressions, or literal values.
Note
Blank spaces are treated like characters.
◦ SET EXACT is OFF - Analytics uses the shorter string when comparing two strings of unequal length. The comparison starts with the leftmost characters and moves to the right.
For example, "AB" is equal to "AB", and it is also considered equal to "ABC".
◦ SET EXACT is ON - comparison strings must be exactly identical to be a match. When comparing two strings of unequal length, Analytics pads the shorter string with trailing blank spaces to match the length of the longer string.
For example, "AB" is equal to "AB", but it is not considered equal to "ABC".
For more examples illustrating SET EXACT, see "Exact Character Comparisons" in Table tab (Options dialog box).
You can use the ALLTRIM( ) function to remove leading and trailing blank spaces and ensure that only text characters and internal spaces are compared.
For example: ALLTRIM(" AB") = ALLTRIM("AB") is True when the values are wrapped with ALLTRIM( ), but False otherwise.
Some Analytics commands and functions are affected by SET EXACT and some are not:
<table>
<tr><th>Affected</th><th>Not affected</th></tr>
<tr><td>◦ LOCATE command
◦ MATCH( ) function
◦ BETWEEN( ) function</td><td>◦ JOIN command
◦ DEFINE RELATION command
◦ FIND( ) function
◦ FINDMULTI( ) function</td></tr>
</table>
</td></tr>
<tr><td>SET FILTER <TO> {test | filter_name}</td><td>
<pre>SET FILTER TO ProdNo = "070104347"</pre>
<pre>SET FILTER TO ProdNoFilter</pre>
Creates a global filter (view filter) on the open table, and specifies either a logical test, or the name of an existing saved filter.
Specifying SET FILTER, without any parameter, removes any filter from the open table.
</td></tr>
<tr><td>SET FOLDER folder path</td><td>Specifies the Analytics project folder in the Overview tab for command output. The default output folder is the folder containing the active table.
This a DOS-style path using the format /foldername/subfoldername, in which the initial slash (/) indicates the root level in the Overview tab. You must specify a full file path.</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Syntax	Examples and remarks
	◦ SET FOLDER /Tables/Results sets the output folder to the Results subfolder. If the Results subfolder does not exist, it is created. ◦ SET FOLDER / sets the output folder to the root level in the **Overview** tab ◦ SET FOLDER sets the output folder to the default (the folder containing the active table) The output folder remains as whatever you set it – until you reset it, or close the project. Upon opening the project, the output folder reverts to the default of the active table folder.
SET FORMAT {ON | OFF}	SET FORMAT ON Default setting: OFF If you use the ON parameter, Analytics automatically displays the current table layout and computed field definitions when you open a new table. The results appear in the command log.
SET FUZZYGROUPSIZE <TO> *num*	SET FUZZYGROUPSIZE TO 10 Specifies the maximum number of items that can appear in a fuzzy duplicate group in the output results. The *num* parameter cannot be less than 2 or greater than 100. The default size is 20. The specified size remains in effect for the duration of the Analytics session.
SET GRAPH *type*	SET GRAPH LINE Specifies the graph type to use for all subsequently generated graphs. The commands run must be compatible with the specified graph type. For example, the BENFORD command cannot produce a PIE2D or PIE3D chart. If an incompatible graph type is specified the default graph type is used (BAR3D). The *type* parameter must be one of the following: ◦ PIE2D ◦ PIE3D ◦ BAR2D ◦ BAR3D – This is the default graph type. ◦ STACKED2D ◦ STACKED3D ◦ LAYERED ◦ LINE ◦ BENFORD – Combines 2D bar graph and 2D line graph.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Syntax	Examples and remarks
SET HISTORY <TO> *value*	SET HISTORY TO 50 Specifies the maximum number of table history entries to retain. The *value* parameter must be between 1 and 100.
SET INDEX <TO> *value*	SET INDEX TO "CustomerCode.INX" Specifies the index to apply to the active table.
SET LEARN <TO> *script*	SET LEARN TO InventoryRec Specifies the name of the script file that the **Script Recorder** uses to record commands.
SET LOG <TO> {*file* | OFF}	SET LOG TO "analysis.log" SET LOG OFF The first command switches logging to the specified log. If the specified log does not exist, it is created. The second command restores logging to the original Analytics command log. **Note** The maximum length of an Analytics project path and log name is 259 characters, which includes the file path, the log name, and the file extension (.log).
SET LOOP <TO> *num*	SET LOOP TO 20 Specifies the maximum number of loops that can be executed by the LOOP command before the command is terminated. The *num* range is 0 to 32767, where 0 turns off loop testing.
SET MARGIN *side* <TO> *value*	SET MARGIN TOP TO 100 Specify LEFT, RIGHT, TOP, or BOTTOM for the *side* parameter. If you want to change the margin on all sides, you need to specify each margin with a separate SET MARGIN command. Specifying a *value* of 100 creates a margin of 1 inch.
SET MATH <TO> {FIRST | LAST | MIN | MAX}	SET MATH TO MIN Default setting: MAX

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Syntax	Examples and remarks
	Specifies how decimal precision works when two operands are evaluated in a numeric expression. ◦ **FIRST** - use the number of decimal places of the first operand in a pair of operands ◦ **LAST** - use the number of decimal places of the last operand in a pair of operands ◦ **MIN** - use the minimum number of decimal places in a pair of operands ◦ **MAX** - use the maximum number of decimal places in a pair of operands In multi-operand expressions, the SET MATH setting works on a pairwise basis, applying the specified setting to each pair of operands, rounding as necessary, as they are evaluated in the standard mathematical order (BOMDAS). If the SET MATH setting reduces the number of decimal places in a result, the result is rounded, not truncated. For more information, see Controlling rounding and decimal precision in numeric expressions. **Note** You cannot use SET MATH while an Analytics table is open.
SET MONTHS <TO> *string*	Specifies the default three-character abbreviations for month names. The *string* parameter is the list of month abbreviations separated by commas.
SET NOTIFYFAILSTOP {ON | OFF}	SET NOTIFYFAILSTOP ON Default setting: OFF ◦ **NOTIFYFAILSTOP is OFF** - Analytics allows a script to continue even if a NOTIFY command in the script fails. ◦ **NOTIFYFAILSTOP is ON** - Analytics stops processing a script, and writes a message to the log, if a NOTIFY command in the script fails. The script stops after the initial failure, or after the specified number of NOTIFYRETRYATTEMPTS, if none of the attempts are successful.
SET NOTIFYRETRYATTEMPTS <TO> *num*	SET NOTIFYRETRYATTEMPTS TO 10 Specifies the number of times the NOTIFY command will attempt to send an email if the initial attempt is unsuccessful. Enter a number from 0 to 255. If you enter 0, no additional attempts are made after an initial failure. The default is 5. One possible reason for the NOTIFY command failing to send an email is that the email server is unavailable.
SET NOTIFYRETRYINTERVAL <TO> *seconds*	SET NOTIFYRETRYINTERVAL TO 30 Specifies the amount of time in seconds between NOTIFYRETRYATTEMPTS. Enter a number from 1 to 255. The default is 10 seconds.
SET ORDER <TO> *values*	Specifies the sort sequence for character fields. The *values* parameter lists all of the character for the selected sort order.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Syntax	Examples and remarks
SET OVERFLOW {ON | OFF}	SET OVERFLOW OFF Default setting: ON If OFF is specified Analytics does not stop processing when an overflow error occurs.
SET PASSWORD *num* <TO> *string*	SET PASSWORD 1 TO "password123" Used to create a password definition, and specify a password value, for unattended script execution. The *num* parameter uniquely identifies the password definition and must be a value from 1 to 10. Specify the password value as a quoted string.
SET PERIODS <TO> *value* <,...*n*>	SET PERIODS TO "0,30,90,180,10000" Specifies the default aging periods used by the AGE command.
SET PICTURE *format*	SET PICTURE "(9,999,999.99)" Specifies the default formatting for numeric values.
SET READAHEAD <TO> *size*	Specifies the size of the data block read. You should only change this setting if you are advised to do so by Support.
SET RETRY <TO> *num* SET RETRYIMPORT <TO> *num*	SET RETRY TO 50 Specifies the number of times Analytics attempts to import or export data if the initial attempt is unsuccessful. Enter a number from 0 to 255. If you enter 0, no additional attempts are made after an initial failure. The default is 0. There is no waiting period between retry attempts. Each successive attempt is made immediately after a preceding failure. The ability to specify retry attempts is useful when connecting to databases or cloud data services, which can be temporarily unavailable. Applies to the following commands: ◦ ACCESSDATA ◦ IMPORT GRCPROJECT ◦ IMPORT GRCRESULTS ◦ IMPORT SAP ◦ RETRIEVE ◦ REFRESH (for tables initially created using ACCESSDATA or IMPORT SAP only) ◦ EXPORT . . . ACLGRC (export to HighBond Results)

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Syntax	Examples and remarks
	Note SET RETRYIMPORT is retained for backward compatibility. SET RETRYIMPORT and SET RETRY perform identical actions.
SET SAFETY {ON | OFF}	SET SAFETY OFF Specify ON to display a confirmation dialog box when overwriting any of the following: ◦ fields in table layouts ◦ Analytics tables ◦ files, including Analytics data files (.fil) Specify OFF to prevent the dialog box from being displayed. Specifying SET SAFETY, without any parameter, in the command line displays whether SAFETY is currently on or off.
SET SEPARATORS <TO> *values*	SET SEPARATORS TO ".,," Specifies the default decimal, thousands, and list separators used by Analytics. The SET SEPARATORS values must be three valid separator characters in the following order: ◦ decimal (period, comma, or space) ◦ thousands (period, comma, or space) ◦ list (semi-colon, comma, or space) Among the three separators, the decimal separator must be unique. You must specify all three separators when you use the command. The list separator is used primarily to separate function parameters.
SET SESSION <*session_name*>	SET SESSION SET SESSION "Analysis" Creates a new session in the Analytics command log. The session is identified by the current timestamp. The optional *session_name* allows you to add up to 30 characters of additional identifying information. Quotation marks are permitted but not required.
SET SORTMEMORY *num*	SET SORTMEMORY 800 Specifies the maximum amount of memory allocated for sorting and indexing processes. The *num* parameter must be a value from 0 to 2000 megabytes (MB), to be entered in 20MB increments. If the sort memory is set to 0, Analytics uses the memory currently available.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Syntax	Examples and remarks
SET SUPPRESSTIME {ON | OFF}	SET SUPPRESSTIME ON Default setting: OFF Only for use when defining an Analytics table that uses an ODBC data source (IMPORT ODBC command), or direct database access (DEFINE TABLE DB command). If you use the ON parameter, when defining the table Analytics suppresses the time portion of datetime values. For example, 20141231 235959 is read, displayed in views, and subsequently processed as 20141231. Including this command in a pre-datetime Analytics script (pre v.10.0) that assumes the time portion of datetime data will be truncated allows the script to run in the datetime-enabled version of Analytics. Analytics suppresses the time portion by using only the date portion of the datetime format. The time data is still present in the .fil file or the database table. If required, you can redefine the field or define a new field to include the time portion of the data. If SET SUPPRESSTIME = OFF, Analytics tables defined using ODBC or direct database access include full datetime values. You can issue the SET SUPPRESSTIME ON/OFF command in a script or from the command line. Specifying SET SUPPRESSTIME, without any parameter, in the command line displays whether the suppression of the time portion of datetime data is currently on or off.
SET SUPPRESSXML {ON | OFF}	SET SUPPRESSXML ON Default setting: OFF Specifies that command output is in plain text rather than formatted text.
SET TEST {ON | OFF}	SET TEST ON Specifies whether the results of IF, WHILE, FOR, and NEXT tests associated with GROUP commands should be recorded in the log.
SET TIME <TO> *string*	SET TIME "hh:mm:ss PM" Specifies how Analytics displays the time portion of datetimes, and standalone time values, in views, reports, and exported files. When using the SET TIME command to specify time formats, you must use 'h' for Hour, 'm' for Minute, and 's' for Second, even if you have specified different time format characters in the **Options** dialog box. For example: SET TIME TO "hh:mm"

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Syntax	Examples and remarks
SET UTCZONE {ON | OFF}	SET UTCZONE OFF Default setting: ON ◦ **UTCZONE is ON** - Analytics changes the display of local times with a UTC offset to the UTC equivalent of the local time. (UTC is Coordinated Universal Time, the time at zero degrees longitude.) ◦ **UTCZONE is OFF** - Analytics displays local times with a UTC offset without converting them to UTC. For example: ◦ 01 Jan 2015 04:59:59 (SET UTCZONE ON) ◦ 31 Dec 2014 23:59:59-05:00 (SET UTCZONE OFF) Conversion of local time to UTC is for display purposes only, and does not affect the source data. You can change back and forth between the two different display modes whenever you want to.
SET VERIFY {ON | OFF | BLANK}	SET VERIFY ON When ON is specified, Analytics automatically checks whether the contents of a data field correspond to the field's data type in the table layout whenever a table is opened. When BLANK is specified, Analytics replaces invalid character data with blanks and invalid numeric data with zeros, in addition to the verification described for the ON parameter.
SET WIDTH <TO> *characters*	SET WIDTH TO 20 Specifies the default display width in characters for numeric computed fields or ad hoc numeric expressions when Analytics cannot determine the maximum width.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SIZE command

Calculates a statistically valid sample size, and sample interval, for record sampling or monetary unit sampling.

Record samplingMonetary unit sampling

Syntax

SIZE RECORD CONFIDENCE *confidence_level* POPULATION *population_size* PRECISION *tolerable_rate* <ERRORLIMIT *expected_rate*> <TO {SCREEN|*filename*}>

Parameters

> **Note**
>
> Do not include thousands separators, or percentage signs, when you specify values.

Name	Description
RECORD	Calculate sample size for a record sample. ATTRIBUTE is a deprecated parameter that does the same thing as RECORD.
CONFIDENCE *confidence_level*	The desired confidence level that the resulting sample is representative of the entire population. For example, specifying 95 means that you want to be confident that 95% of the time the sample will in fact be representative. Confidence is the complement of "sampling risk". A 95% confidence level is the same as a 5% sampling risk.
POPULATION *population_size*	The number of records in the table you are sampling.
PRECISION *tolerable_rate*	The tolerable deviation rate, which is the maximum rate of deviation from a prescribed control that can occur and you still consider the control effective. For example, specifying 5 means that the deviation rate must be greater than 5% for you to consider the control not effective.
ERRORLIMIT *expected_rate* optional	The expected population deviation rate. This is the rate of deviation from a prescribed control that you expect to find. For example, specifying 1 means that you expect the deviation rate to be 1%. If you omit this parameter, an expected population deviation rate of 0% is used.
TO SCREEN | *filename*	The location to send the results of the command to:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"`

Analytics output variables

Name	Contains
SAMPINT*n*	The required sample interval calculated by the command.
SAMPSIZE*n*	The required sample size calculated by the command.

Examples

Calculate the required size and interval for a record sample

You have decided to use record sampling to estimate the rate of deviation from a prescribed control in an account containing invoices.

Before drawing the sample, you must first calculate the statistically valid sample size and sample interval.

You want to be confident that 95% of the time the sample drawn by Analytics will be representative of the population as a whole.

Using your specified confidence level, the example below calculates a sample size of 95, and a sample interval value of 8.12, to use when drawing a record sample:

```
SIZE RECORD CONFIDENCE 95 POPULATION 772 PRECISION 5 ERRORLIMIT 1 TO SCREEN
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Syntax

SIZE MONETARY CONFIDENCE *confidence_level* POPULATION *population_size* MATERIALITY *tolerable_misstatement* <ERRORLIMIT *expected_misstatement*> <TO {SCREEN|*filename*}>

Parameters

> **Note**
>
> Do not include thousands separators, or percentage signs, when you specify values.

Name	Description
MONETARY	Calculate sample size for a monetary unit sample.
CONFIDENCE *confidence_level*	The desired confidence level that the resulting sample is representative of the entire population. For example, specifying 95 means that you want to be confident that 95% of the time the sample will in fact be representative. Confidence is the complement of "sampling risk". A 95% confidence level is the same as a 5% sampling risk.
POPULATION *population_size*	The total absolute value of the numeric sample field.
MATERIALITY *tolerable_misstatement*	The tolerable misstatement, which is the maximum total amount of misstatement that can occur in the sample field without being considered a material misstatement. For example, specifying 29000 means that the total amount of misstatement must be greater than $29,000 to be considered a material misstatement.
ERRORLIMIT *expected_misstatement* optional	The expected misstatement. This is the total amount of misstatement that you expect the sample field to contain. For example, specifying 5800 means that you expect the total amount of misstatement to be $5,800. If you omit this parameter, an expected misstatement of $0.00 is used.
TO SCREEN | *filename*	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Tip You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"`

Analytics output variables

Name	Contains
SAMPINT*n*	The required sample interval calculated by the command.
SAMPSIZE*n*	The required sample size calculated by the command.

Examples

Calculate the required size and interval for a monetary unit sample

You have decided to use monetary unit sampling to estimate the total amount of monetary misstatement in an account containing invoices.

Before drawing the sample, you must first calculate the statistically valid sample size and sample interval.

You want to be confident that 95% of the time the sample drawn by Analytics will be representative of the population as a whole.

Using your specified confidence level, the example below calculates a sample size of 93, and a sample interval value of 6,283.33, to use when drawing a monetary unit sample:

```
SIZE MONETARY CONFIDENCE 95 POPULATION 585674.41 MATERIALITY 29000 ERRORLIMIT 5800 TO SCREEN
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SORT command

Sorts records in an Analytics table into an ascending or descending sequential order, based on a specified key field or fields. The results are output to a new, physically reordered Analytics table.

Syntax

SORT ON {*key_field* <D> <...*n*>|ALL} <FIELDS *field_name* <AS *display_name*> <...*n*>|FIELDS ALL> TO *tablename* <LOCAL> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <APPEND> <OPEN> <ISOLOCALE *locale_code*>

Parameters

Name	Description
ON *key_field* D <...*n*> | ALL	The key field or fields, or the expression, to use for sorting. You can sort by any type of field, including computed fields and ad hoc expressions, regardless of data type. ◦ ***key_field*** - use the specified field or fields If you sort by more than one field, you create a nested sort in the output table. The order of nesting follows the order in which you specify the fields. Include D to sort the key field in descending order. The default sort order is ascending. ◦ **ALL** - use all fields in the table If you sort by all the fields in a table you create a nested sort in the output table. The order of nesting follows the order in which the fields appear in the table layout. An ascending sort order is the only option for ALL.
FIELDS *field_name* <...*n*> | FIELDS ALL optional	**Note** Key fields are automatically included in the output table, and do not need to be specified using FIELDS. The fields to include in the output: ◦ **FIELDS *field_name*** - use the specified fields Fields are used in the order that you list them. Converts computed fields to physical fields of the appropriate data type in the destination table - ASCII or Unicode (depending on the edition of Analytics), ACL (the native numeric data type), Datetime, or Logical. Populates the physical fields with the actual computed values. ◦ **FIELDS ALL** - use all fields in the table Fields are used in the order that they appear in the table layout.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Converts computed fields to physical fields of the appropriate data type in the destination table - ASCII or Unicode (depending on the edition of Analytics), ACL (the native numeric data type), Datetime, or Logical. Populates the physical fields with the actual computed values. ◦ **omit FIELDS** - the entire record is included in the sorted output table: all fields, and any undefined portions of the record Fields are used in the order that they appear in the table layout. Computed fields are preserved. **Tip** If you need only a portion of the data contained in a record, do not include all fields, or the entire record, in the sorted output table. Select only the fields you need, which in most cases speeds up the sorting process.
AS *display_name* optional	Only used when sorting using FIELDS. The display name (alternate column title) for the field in the view in the new Analytics table. If you want the display name to be the same as the field name, or an existing display name in the source table, do not use AS. Specify *display_name* as a quoted string. Use a semi-colon (;) between words if you want a line break in the column title. **Note** AS works only when outputting to a new table. If you are appending to an existing table, the alternate column titles in the existing table take precedence.
TO *table_name*	The location to send the results of the command to: ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
LOCAL optional	Saves the output file in the same location as the Analytics project.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
OPEN optional	Open the table and apply the index to the table.
ISOLOCALE *locale_code* optional	**Note** Applicable in the Unicode edition of Analytics only. The system locale in the format *language_country*. For example, to use Canadian

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	French, enter fr_ca. Use the following codes: ◦ **language** - ISO 639 standard language code ◦ **country** - ISO 3166 standard country code If you do not specify a country code, the default country for the language is used. If you do not use ISOLOCALE, the default system locale is used.

Examples

Sort on a single field, output entire records

You want to sort the records in the sample **Inventory** table by product number. The sorted records are extracted to a new Analytics table called **Inventory_Product_Number**.

Entire records are included in the output table:

```
SORT ON ProdNo TO "Inventory_Product_Number"
```

To switch from the default ascending sort order to a descending sort order, you add D after the key field name:

```
SORT ON ProdNo D TO "Inventory_Product_Number"
```

Sort on a single field, output a subset of fields

You want to sort the records in the sample **Inventory** table by product number. Only the key field and the specified non-key fields are extracted to a new Analytics table called **Inventory_Quantity_on_Hand**.

The third non-key field, **QtyOH**, is given the display name **Qty on Hand** in the output table:

```
SORT ON ProdNo FIELDS ProdDesc ProdStat QtyOH AS "Qty on Hand" TO "Inventory_Quantity_
on_Hand"
```

Sort on a single field, output all fields

You want to sort the records in the sample **Inventory** table by product number. All fields are extracted to a new Analytics table called **Inventory_Product_Number**.

The difference between using **FIELDS ALL** and outputting the entire record is that **FIELDS ALL** converts any computed fields in the source table to physical fields in the output table, and populates the fields with the actual computed values:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
SORT ON ProdNo FIELDS ALL TO "Inventory_Product_Number"
```

Sort on multiple fields (nested sort)

You want to sort the records in the sample **Inventory** table by location, then by product class, and then by product number. The sorted records are extracted to a new Analytics table called **Inventory_Location_Class_Number**.

```
SORT ON Location ProdCls ProdNo TO "Inventory_Location_Class_Number"
```

Sort using related fields

You want to sort the records in the sample **Ap_Trans** table by the following fields:

- vendor state (related **Vendor** table)
- vendor city (related **Vendor** table)
- vendor number (**Ap_Trans** table)

All three key fields and the specified non-key fields, including the related field **Vendor.Vendor_Name**, are extracted to a new Analytics table called **Ap_Trans_State_City**:

```
SORT ON Vendor.Vendor_State Vendor.Vendor_City Vendor_No FIELDS Vendor.Vendor_Name
Invoice_No Invoice_Date Invoice_Amount Prodno Quantity Unit_Cost TO "Ap_Trans_State_City"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Sorting on related fields

You can sort on related fields, and include related fields as non-key fields in a sorted output table. To reference a related field in the SORT command specify *child table name.field name*.

Fixed-length vs variable-length data files

The SORT command works on both fixed-length and variable-length data files.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

STATISTICS command

Calculates statistics for one or more numeric or datetime fields in an Analytics table.

Syntax

STATISTICS <ON> {*field* <...*n*>|ALL} <STD> <MODMEDQ> <NUMBER *n*> <TO {SCREEN|*filename*|PRINT}> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <APPEND>

Parameters

Name	Description
ON *field* <...*n*> | ALL	Specify one or more numeric or datetime fields to generate statistics for, or specify ALL to generate statistics for all numeric and datetime fields in the Analytics table.
STD optional	Calculates the standard deviation of the fields specified, in addition to the other statistics.
MODMEDQ optional	Calculates the mode, median, first quartile, and third quartile values of the fields specified, in addition to the other statistics.
NUMBER *n* optional	The number of high and low values to retain during processing. The default value is 5.
TO SCREEN | *filename* | PRINT optional	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"` ◦ **PRINT** - sends the results to the default printer

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.

Analytics output variables

Note

If you generate statistics for more than one field in a table, the system-generated output variables contain values for the first listed field only.

Name	Contains
ABS*n*	The absolute value calculated by the command.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Contains
AVERAGE*n*	The mean value calculated by the command.
COUNT*n*	The record count calculated by the command. ◦ If the variable name is COUNT1, it is storing the record count for the most recent command executed. ◦ If the variable name is COUNT*n*, where *n* is greater than 1, the variable is storing the record count for a command executed within a GROUP command. The value of *n* is assigned based on the line number of the command in the GROUP. For example, if the command is one line below the GROUP command it is assigned the value COUNT2. If the command is four lines below the GROUP command, it is assigned the value COUNT5.
HIGH*n*	The 5th highest value identified by the command. The 5th highest is the default setting. The setting can be changed using the NUMBER parameter. For example, NUMBER 3 specifies that the 3rd highest value is stored. **Note** When Analytics identifies the highest value, duplicate values are not factored out. For example, if values in descending order are 100, 100, 99, 98, the 3rd highest value is 99, not 98.
LOW*n*	The 5th lowest value identified by the command. The 5th lowest is the default setting. The setting can be changed using the NUMBER parameter. For example, NUMBER 3 specifies that the 3rd lowest value is stored. **Note** When Analytics identifies the lowest value, duplicate values are not factored out. For example, if values in ascending order are 1, 1, 2, 3, the 3rd lowest value is 2, not 3.
MAX*n*	The maximum value identified by the command.
MEDIAN*n*	The median value identified by the command.
MIN*n*	The minimum value identified by the command.
MODE*n*	The most frequently occurring value identified by the command.
Q25*n*	The first quartile value (lower quartile value) calculated by the command.
Q75*n*	The third quartile value (upper quartile value) calculated by the command.
RANGE*n*	The difference between the maximum and minimum values calculated by the command.
STDDEV*n*	The standard deviation value calculated by the command.
TOTAL*n*	The total value calculated by the command.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Contains
	The value of *n* is 1 unless the TOTAL command is inside a GROUP command, in which case the value of *n* corresponds to the line number of the TOTAL command in the GROUP command. For more information, see "GROUP command" on page 227.

Examples

Generating conditional statistics

You generate statistics for the **Quantity** field in records where the product class ID is 01:

```
STATISTICS ON Quantity IF ProdCls = "01"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

STRATIFY command

Groups records into numeric intervals based on values in a numeric field. Counts the number of records in each interval, and also subtotals specified numeric fields for each interval.

Syntax

STRATIFY <ON> *numeric_field* MINIMUM *value* MAXIMUM *value* {<INTERVALS *number*>|FREE *interval_value* <...n> *last_interval*} <SUPPRESS> <SUBTOTAL *numeric_field* <...n>|SUBTOTAL ALL> <KEY *break_field*> <TO {SCREEN|*table_name*|*filename*|GRAPH|PRINT}> <LOCAL> <IF *test*> <FIRST *range*|NEXT *range*> <WHILE *test*> <APPEND> <OPEN> <HEADER *header_text*> <FOOTER *footer_text*> <STATISTICS>

Parameters

Name	Description
ON *numeric_field*	The numeric field or expression to be stratified.
MINIMUM *value*	Applies to numeric fields only. The minimum value of the first numeric interval. MINIMUM is optional if you are using FREE, otherwise it is required.
MAXIMUM *value*	Applies to numeric fields only. The maximum value of the last numeric interval. MAXIMUM is optional if you are using FREE, otherwise it is required.
INTERVALS *number* optional	Applies to numeric fields only. The number of equal-sized intervals Analytics produces over the range specified by the MINIMUM and MAXIMUM values. If you do not specify a number of intervals, the default number is used. The default is specified by the **Intervals** number on the **Command** tab in the **Options** dialog box.
FREE *interval_value* <...n> *last_interval* optional	Applies to numeric fields only. Creates custom-sized intervals by specifying the start point of each interval and the end point of the last interval. If you specify MINIMUM and MAXIMUM values, those values are the start point of the first interval and the end point of the last interval, and each *interval_value* creates an additional interval within the range. The interval values you specify must be greater than the MINIMUM value, and equal to or less than the MAXIMUM value. Interval values must be in numeric sequence and cannot contain duplicate values:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	FREE -1000, 0, 1000, 2000, 3000 If you specify both FREE and INTERVALS, then INTERVALS is ignored.
SUPPRESS optional	Values above the MAXIMUM value and below the MINIMUM value are excluded from the command output.
SUBTOTAL *numeric_field <...n>* | SUBTOTAL ALL optional	One or more numeric fields or expressions to subtotal for each group. Multiple fields must be separated by spaces. Specify ALL to subtotal all the numeric fields in the table. If you do not select a subtotal field, the field you are stratifying on is automatically sub-totaled. You must explicitly specify the stratify field if you want to subtotal it along with one or more other fields, or if you want to include statistics for the subtotaled stratify field.
KEY *break_field* optional	The field or expression that groups subtotal calculations. A subtotal is calculated each time the value of *break_field* changes. *break_field* must be a character field or expression. You can specify only one field, but you can use an expression that contains more than one field.
TO SCREEN *table_name* | *filename* | GRAPH | PRINT	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Use either an absolute or relative file path to save the file to a different, existing folder: • TO "C:\Output.TXT" • TO "Results\Output.TXT" ◦ **GRAPH** - displays the results in a graph in the Analytics display area ◦ **PRINT** - sends the results to the default printer
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics FOOTER system variable.
STATISTICS optional	**Note** Cannot be used unless SUBTOTAL is also specified. Calculates average, minimum, and maximum values for all SUBTOTAL fields.

Examples

Stratifying on invoice amount

You need to stratify an accounts receivable table on the **Invoice_Amount** field. The invoice amount is also automatically subtotaled.

The output is grouped into $1000 intervals:

- from $0 to $999.99
- from $1,000 to $1,999.99
- so on

The total invoice amount is included for each interval.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
OPEN Ar
STRATIFY ON Invoice_Amount MINIMUM 0 MAXIMUM 10000 INTERVALS 10 TO "Stratified_
invoices.FIL"
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

How it works

STRATIFY groups records into equal-sized, or custom-sized, numeric intervals based on values in a numeric field.

The output contains a single record for each interval, with a count of the number of records in the source table that fall into each interval.

Automatically populate the MINIMUM and MAXIMUM values

You can run the STATISTICS or PROFILE commands on the stratify field before running the STRATIFY command to automatically populate the MINIMUM and MAXIMUM parameter values with the lowest and highest values in the field.

Names of auto-generated subtotal and statistics fields

If you use STATISTICS to perform statistical calculations on one or more SUBTOTAL fields, and output the results to an Analytics table, the fields auto-generated by the parameters have the following names:

Description of auto-generated field	Field name in output table	Alternate column title (display name) in output table
Subtotal field	*subtotaled field name in source table*	**Total** + *subtotaled alternate column title in source table*
Average field	**a_***subtotaled field name in source table*	**Average** + *subtotaled alternate column title in source table*
Minimum field	**m_***subtotaled field name in source table*	**Minimum** + *subtotaled alternate column title in source table*
Maximum field	**x_***subtotaled field name in source table*	**Maximum** + *subtotaled alternate column title in source table*

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SUMMARIZE command

Groups records based on identical values in one or more character, numeric, or datetime fields. Counts the number of records in each group, and also subtotals specified numeric fields for each group.

Syntax

SUMMARIZE ON *key_field <...n>* <SUBTOTAL *numeric_field <...n>*|SUBTOTAL ALL> <OTHER *field <...n>*|OTHER ALL> <TO {SCREEN|*table_name*|PRINT}> <LOCAL> <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <PRESORT> <APPEND> <OPEN> <HEADER *header_text*> <FOOTER *footer_text*> <STATISTICS> <MODMEDQ> <STDEV> <CPERCENT> <ISOLOCALE *locale_code*>

Parameters

Name	Description
ON *key_field <...n>*	One or more character, numeric, or datetime fields to summarize. Multiple fields must be separated by spaces, and can be different data types.
SUBTOTAL *numeric_field <...n>* | SUBTOTAL ALL optional	One or more numeric fields or expressions to subtotal for each group. Multiple fields must be separated by spaces. Specify ALL to subtotal all the numeric fields in the table.
OTHER *field <...n>* | OTHER ALL optional	One or more additional fields to include in the output. ◦ ***field <...n>*** – include the specified field or fields ◦ **ALL** – include all fields in the table that are not specified as key fields or subtotal fields Use OTHER only with fields that contain the same value for all records in each summarized group. If you specify a field that contains values that are different for a summarized group, only the value for the first record in the group is displayed, which is not meaningful. For example: ◦ **summarize a table on customer number** – an appropriate "other field" is **Customer Name**. Typically, the customer name is identical for all records with the same customer number. ◦ **summarize a vendor table by state** – an inappropriate "other field" is **City**. Only the first city listed for each state appears in the output. In this instance, the better approach is to summarize using both state and city as key fields, in that order.
TO SCREEN *table_name* | PRINT	The location to send the results of the command to: ◦ **SCREEN** – displays the results in the Analytics display area

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Tip You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***table_name*** - saves the results to an Analytics table Specify *table_name* as a quoted string with a .FIL file extension. For example: `TO "Output.FIL"` By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: • `TO "C:\Output.FIL"` • `TO "Results\Output.FIL"` **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number. ◦ **PRINT** - sends the results to the default printer
LOCAL optional	Saves the output file in the same location as the Analytics project. **Note** Applicable only when running the command against a server table with an output file that is an Analytics table. The LOCAL parameter must immediately follow the TO parameter.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.
PRESORT optional	Sorts the table on the key field before executing the command. **Note** You cannot use PRESORT inside the GROUP command. **If you use PRESORT** If you use **PRESORT**, the output is sorted and contains a single, unique group for each set of identical values, or identical combination of values, in the key field or fields. **Tip** If the input table is already sorted, you can save processing time by not specifying **PRESORT**. **If you do not use PRESORT** If you do not use **PRESORT**, the output results use the sort order of the input table. If the key field or fields contain non-sequential identical values, the output results contain more than one group for each set of identical values, or identical combination of values. **Note** Depending on the context, more than one group for each set of identical values, or identical combination of values, can defeat the purpose of summarizing.
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.
OPEN optional	Opens the table created by the command after the command executes. Only valid if the command creates an output table.
HEADER *header_text* optional	The text to insert at the top of each page of a report. *header_text* must be specified as a quoted string. The value overrides the Analytics HEADER system variable.
FOOTER *footer_text* optional	The text to insert at the bottom of each page of a report. *footer_text* must be specified as a quoted string. The value overrides the Analytics

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	FOOTER system variable.
STATISTICS optional	**Note** Cannot be used unless SUBTOTAL is also specified. Calculates average, minimum, and maximum values for all SUBTOTAL fields.
MODMEDQ optional	**Note** Cannot be used unless SUBTOTAL is also specified. Calculates mode, median, first quartile, and third quartile values for all SUBTOTAL fields.
STDEV optional	**Note** Cannot be used unless SUBTOTAL is also specified. Calculates standard deviation and percentage of total for all SUBTOTAL fields.
CPERCENT optional	Calculates percentage of record count for each group.
ISOLOCALE optional	**Note** Applicable in the Unicode edition of Analytics only. The system locale in the format *language_country*. For example, to use Canadian French, enter fr_ca. Use the following codes: ◦ **language** - ISO 639 standard language code ◦ **country** - ISO 3166 standard country code If you do not specify a country code, the default country for the language is used. If you do not use ISOLOCALE, the default system locale is used.

Examples

Total transaction amount per customer

You summarize an accounts receivable table on the **Customer_Number** field, and subtotals the **Trans_Amount** field. The output is grouped by customer and includes the total transaction amount for each customer:

```
OPEN Ar
SUMMARIZE ON Customer_Number SUBTOTAL Trans_Amount TO "Customer_total.FIL"
PRESORT
```

Total transaction amount per customer per transaction date

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You summarize an accounts receivable table on the **Customer_Number** and **Trans_Date** fields. You subtotal the **Trans_Amount** field.

The output is grouped by customer, and within customer by date, and includes the total transaction amount for each customer for each date the customer had transactions.

```
OPEN Ar
SUMMARIZE ON Customer_Number Trans_Date SUBTOTAL Trans_Amount TO "Customer_total_
by_date.FIL" PRESORT
```

Total, average, minimum, and maximum transaction amounts per customer per transaction date

You add STATISTICS to the previous example.

In addition to the subtotaled transaction amount for each customer for each date the customer had transactions, you also calculate the average, minimum, and maximum transaction amounts for each customer for each date:

```
OPEN Ar
SUMMARIZE ON Customer_Number Trans_Date SUBTOTAL Trans_Amount TO "Customer_stats_
by_date.FIL" PRESORT STATISTICS
```

Identical transaction amounts, same date

You summarize a credit card transactions table on the **Trans_Date** and **Trans_Amount** fields.

The output is grouped by date, and within date by amount. You can use the associated count to identify transactions with identical amounts and identical dates:

```
OPEN CC_Trans
SUMMARIZE ON Trans_Date Trans_Amount TO "Transactions_by_date_amount.FIL" OPEN
PRESORT
SET FILTER TO COUNT > 1
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

How it works

SUMMARIZE groups records that have the same value, or the same combination of values, in one or more character, numeric, or datetime fields. The output contains a single record for each group, with a count of the number of records in the source table that belong to the group.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Subtotal and statistics: calculations and field names in the output results

You can use one or more optional parameters to perform statistical calculations on any SUBTOTAL field you specify. The statistical calculations are broken down by group in the output:

Optional Parameter	Alternate column title (display name) in output table	Field name in output table	Calculation performed on subtotaled field
SUBTOTAL	**Total** + *subtotaled alternate column title*	*subtotaled field name*	Subtotaled values for each group
STATISTICS	**Average** + *subtotaled alternate column title*	**a**_*subtotaled field name*	The average value for each group
	Minimum + *subtotaled alternate column title*	**m**_*subtotaled field name*	The minimum value for each group
	Maximum + *subtotaled alternate column title*	**x**_*subtotaled field name*	The maximum value for each group
MODMEDQ	**Median** + *subtotaled alternate column title*	**c**_*subtotaled field name*	The median value for each group ◦ Odd-numbered sets of values: the middle value ◦ Even-numbered sets of values: the average of the two values at the middle
	Mode + *subtotaled alternate column title*	**o**_*subtotaled field name*	The most frequently occurring value for each group ◦ Displays "N/A" if no value occurs more than once ◦ In the event of a tie, displays the lowest value
	Q25 + *subtotaled alternate column title*	**q**_*subtotaled field name*	The first quartile value for each group (lower quartile value) ◦ The result is an interpolated value based on an Analytics algorithm ◦ Produces the same result as the QUARTILE and QUARTILE.INC functions in Microsoft Excel

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Optional Parameter	Alternate column title (display name) in output table	Field name in output table	Calculation performed on subtotaled field
	Q75 + *subtotaled alternate column title*	**p_***subtotaled field name*	The third quartile value for each group (upper quartile value) ◦ The result is an interpolated value based on an Analytics algorithm ◦ Produces the same result as the QUARTILE and QUARTILE.INC functions in Microsoft Excel
STDEV	**STDDEV** + *subtotaled alternate column title*	**d_***subtotaled field name*	The standard deviation for each group
	% Field + *subtotaled alternate column title*	**f_***subtotaled field name*	Each group's subtotal expressed as a percentage of the field total
CPERCENT	**Percent of Count**	**COUNT_PERCENTAGE**	The percentage of source table records belonging to each group **Note** Does not require a subtotal field

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

TOP command

Moves to the first record in an Analytics table.

Syntax

```
TOP
```

Parameters

This command does not have any parameters.

Remarks

When to use TOP

Use TOP to move to the first record in a table if a previous command, such as FIND, selected another record in the table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

TOTAL command

Calculates the total value of one or more fields in an Analytics table.

Syntax

TOTAL {<FIELDS> *numeric_field* <...*n*>|<FIELDS> ALL} <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*>

Parameters

Name	Description
FIELDS *numeric_field* <...*n*> | FIELDS ALL	The numeric fields to total. Specify ALL to total each of the numeric fields in the table.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics output variables

> **Note**
>
> If you total more than one field in a table, the system-generated output variable contains the total for the first listed field only.

Name	Contains
TOTAL*n*	The total value calculated by the command. The value of *n* is 1 unless the TOTAL command is inside a GROUP command, in which case the value of *n* corresponds to the line number of the TOTAL command in the GROUP command. For more information, see "GROUP command" on page 227.

Examples

Totaling the first 25 records

You calculate the total amount of the **MKTVAL** field for the first 25 records in the table:

```
TOTAL FIELDS MKTVAL FIRST 25
```

Remarks

When to use TOTAL

Use TOTAL to verify the completeness and accuracy of the source data and to produce control totals. The command calculates the arithmetic sum of the specified fields or expressions.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

TRAIN command

Uses automated machine learning to create an optimum predictive model using a training data set.

> **Note**
>
> The TRAIN command is not supported if you are running Analytics on a 32-bit computer. The computation required by the command is processor-intensive and better suited to 64-bit computers.

Syntax

```
TRAIN {CLASSIFIER|REGRESSOR} <ON> key_field <...n> TARGET labeled_field SCORER
{ACCURACY|AUC|F1|LOGLOSS|PRECISION|RECALL|MAE|MSE|R2} SEARCHTIME minutes
MAXEVALTIME minutes MODEL model_name TO table_name <IF test> <WHILE test> <FIRST
range|NEXT range> FOLDS number_of_folds <SEED seed_value> <LINEAR> <NOFP>
```

> **Note**
>
> The maximum supported size of the data set used with the TRAIN command is 1 GB.

Parameters

Name	Description
CLASSIFIER | REGRESSOR	The prediction type to use when training a predictive model: ◦ **CLASSIFIER** - use classification algorithms to train a model Use classification if you want to predict which class or category records belong to. ◦ **REGRESSOR** - use regression algorithms to train a model Use regression if you want to predict numeric values associated with records.
ON *key_field* <...*n*>	One or more training input fields. Fields can be character, numeric, or logical. Multiple fields must be separated by spaces. **Note** Character fields must be "categorical". That is, they must identify categories or classes, and contain a maximum number of unique values. The maximum is specified by the **Maximum Categories** option (**Tools > Options > Command**).
TARGET *labeled_field*	The field that the model is being trained to predict based on the training input fields. The different prediction types (classification or regression) work with different field data types:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><td>Valid with CLASSIFIER</td><td>a character or logical target field</td></tr>
<tr><td>Valid with REGRESSOR</td><td>a numeric target field</td></tr>
</table>
</td></tr>
<tr><td>SCORER ACCURACY | AUC | F1 | LOGLOSS | PRECISION | RECALL | MAE | MSE | R2</td><td>
The metric to use when scoring (tuning and ranking) the generated models.

The generated model with the best value for this metric is kept, and the rest of the models are discarded.

A different subset of metrics is valid depending on the prediction type you are using (classification or regression):
<table>
<tr><td>Valid with CLASSIFIER</td><td>ACCURACY | AUC | F1 | LOGLOSS | PRECISION | RECALL</td></tr>
<tr><td>Valid with REGRESSOR</td><td>MAE | MSE | R2</td></tr>
</table>
Note
The classification metric AUC is only valid when labeled_field contains binary data - that is, two classes, such as Yes/No, or True/False.
</td></tr>
<tr><td>SEARCHTIME minutes</td><td>
The total time in minutes to spend training and optimizing a predictive model.

Training and optimizing involves searching across different pipeline configurations (different model, preprocessor, and hyperparameter combinations).

Note
Total runtime of the TRAIN command is SEARCHTIME plus up to twice MAXEVALTIME.

Tip
Specify a SEARCHTIME that is at least 10x the MAXEVALTIME

This time allotment strikes a reasonable balance between processing time and allowing a variety of model types to be evaluated.
</td></tr>
<tr><td>MAXEVALTIME minutes</td><td>
Maximum runtime in minutes per model evaluation.

Tip
Allot 45 minutes for every 100 MB of training data.

This time allotment strikes a reasonable balance between processing time and allowing a variety of model types to be evaluated.
</td></tr>
<tr><td>MODEL model_name</td><td>
The name of the model file output by the training process.

The model file contains the model best fitted to the training data set. You will input the model to the PREDICT command to generate predictions about a new, unseen data set.

Specify model_name as a quoted string. For example: <code>TO "Loan_default_prediction"</code>

You can specify the <code>*.model</code> file extension, or let Analytics automatically specify it.

By default, the model file is saved to the folder containing the Analytics project.

Use either an absolute or relative file path to save the model file to a different, existing
</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	folder: ◦ TO "C:\Loan_default_prediction" ◦ TO "ML Train output\Loan_default_prediction.model"
TO *table_name*	The name of the model evaluation table output by the training process. The model evaluation table contains two distinct types of information: ◦ **Scorer/Metric** - for the classification or regression metrics, quantitative estimates of the predictive performance of the model file output by the training process Different metrics provide different types of estimates. **Scorer** identifies the metric you specified with SCORER. **Metric** identifies the metrics you did not specify. ◦ **Importance/Coefficient** - in descending order, values indicating how much each feature (predictor) contributes to the predictions made by the model Specify *table_name* as a quoted string with a .FIL file extension. For example: TO "Model_evaluation.FIL" By default, the table data file (.FIL) is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the data file to a different, existing folder: ◦ TO "C:\Model_evaluation.FIL" ◦ TO "ML Train output\Model_evaluation.FIL" **Note** Table names are limited to 64 alphanumeric characters, not including the .FIL extension. The name can include the underscore character (_), but no other special characters, or any spaces. The name cannot start with a number.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	If you omit FIRST and NEXT, all records are processed by default.
FOLDS *number_of_folds*	The number of cross-validation folds to use when evaluating and optimizing the model. Folds are subdivisions of the training data set, and are used in a cross-validation process. Typically, using from 5 to 10 folds yields good results when training a model. The minimum number of folds allowed is 2, and the maximum number is 10. **Tip** Increasing the number of folds can produce a better estimate of the predictive performance of a model, but it also increases overall runtime.
SEED *seed_value* optional	The seed value to use to initialize the random number generator in Analytics. If you omit SEED, Analytics randomly selects the seed value. Explicitly specify a seed value, and record it, if you want to replicate the training process with the same data set in the future.
LINEAR optional	Train and score only linear models. If LINEAR is omitted, all model types relevant to classification or regression are evaluated. **Note** With larger data sets, the training process typically completes more quickly if you include only linear models. Including only linear models guarantees coefficients in the output.
NOFP optional	Exclude feature selection and data preprocessing from the training process. Feature selection is the automated selection of the fields in the training data set that are the most useful in optimizing the predictive model. Automated selection can improve predictive performance, and reduce the amount of data involved in model optimization. Data preprocessing performs transformations such as scaling and standardizing on the training data set to make it better suited for the training algorithms. **Caution** You should only exclude feature selection and data preprocessing if you have a reason for doing so.

Examples

Train a classification model

You want to train a classification model that you can use in a subsequent process to predict which loan applicants will default.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You train the model using a set of historical loan data with a known outcome for each loan, including whether the client defaulted.

In the subsequent prediction process, you will use the model produced by the TRAIN command to process current loan applicant data.

```
OPEN "Loan_applicants_historical"
TRAIN CLASSIFIER ON Age Job_Category Salary Account_Balance Loan_Amount Loan_Period
Refinanced Credit_Score TARGET Default SCORER LOGLOSS SEARCHTIME 960
MAXEVALTIME 90 MODEL "Loan_default_prediction.model" TO "Model_evaluation.FIL" FOLDS 5
```

Train a regression model

You want to train a regression model that you can use in a subsequent process to predict the future sale price of houses.

You train the model using a set of recent house sales data, including the sale price.

In the subsequent prediction process, you will use the model produced by the TRAIN command to generate house price evaluations.

```
OPEN "House_sales"
TRAIN REGRESSOR ON Lot_Size Bedrooms Bathrooms Stories Driveway Recroom Full_Basement
Gas_HW Air_conditioning Garage_Places Preferred_Area TARGET Price SCORER MSE
SEARCHTIME 960 MAXEVALTIME 90 MODEL "House_price_prediction.model" TO "Model_eval-
uation.FIL" FOLDS 5
```

Remarks

> **Note**
>
> For more information about how this command works, see the Analytics Help.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

VERIFY command

Checks for data validity errors in one or more fields in an Analytics table by verifying that the data is consistent with the field definitions in the table layout.

Syntax

VERIFY {<FIELDS> *field* <...*n*>|<FIELDS> ALL} <IF *test*> <WHILE *test*> <FIRST *range*|NEXT *range*> <ERRORLIMIT *n*> <TO {SCREEN|*filename*|PRINT}> <APPEND>

Parameters

Name	Description
FIELDS *field* <...*n*> | FIELDS ALL	The fields or expressions to verify. Specify ALL to verify all fields in the table. **Note** By definition, computed fields, ad hoc expressions, and binary fields are always valid.
IF *test* optional	A conditional expression that must be true in order to process each record. The command is executed on only those records that satisfy the condition. **Note** The IF parameter is evaluated against only the records remaining in a table after any scope parameters have been applied (WHILE, FIRST, NEXT).
WHILE *test* optional	A conditional expression that must be true in order to process each record. The command is executed until the condition evaluates as false, or the end of the table is reached. **Note** If you use WHILE in conjunction with FIRST or NEXT, record processing stops as soon as one limit is reached.
FIRST *range* | NEXT *range* optional	The number of records to process: ◦ **FIRST** - start processing from the first record until the specified number of records is reached ◦ **NEXT** - start processing from the currently selected record until the specified number of records is reached Use *range* to specify the number of records to process. If you omit FIRST and NEXT, all records are processed by default.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
ERRORLIMIT *n* optional	The number of errors allowed before the command is terminated. The default value is 10.
TO SCREEN | *filename* | PRINT optional	The location to send the results of the command to: ◦ **SCREEN** - displays the results in the Analytics display area **Tip** You can click any linked result value in the display area to drill down to the associated record or records in the source table. ◦ ***filename*** - saves the results to a file Specify *filename* as a quoted string with the appropriate file extension. For example: `TO "Output.TXT"` By default, the file is saved to the folder containing the Analytics project. Use either an absolute or relative file path to save the file to a different, existing folder: • `TO "C:\Output.TXT"` • `TO "Results\Output.TXT"` ◦ **PRINT** - sends the results to the default printer
APPEND optional	Appends the command output to the end of an existing file instead of overwriting it. **Note** You must ensure that the structure of the command output and the existing file are identical: • the same fields • the same field order • matching fields are the same length • matching fields are the same data type Analytics appends output to an existing file regardless of its structure. If the structure of the output and the existing file do not match, jumbled, missing, or inaccurate data can result.

Analytics output variables

Name	Contains
WRITE*n*	The total number of data validity errors identified by the command.

Examples

Verifying data and specifying an error limit

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You verify all of the columns in a table and set the error limit to 10. The command stops processing if 10 data validity errors are detected:

```
VERIFY ALL ERRORLIMIT 10 TO "ImportErrors.txt"
```

Remarks

How it works

VERIFY compares the values in one or more fields to the data type specified for each of the fields in the table layout and reports any errors. The command ensures the following:

- **character fields** – contain only valid characters, and that no unprintable characters are present
- **numeric fields** – contain only valid numeric data. In addition to numbers, numeric fields can contain one preceding plus sign or minus sign and one decimal point
- **datetime fields** – contain valid dates, datetimes, or times

For each error that is identified, the record number and field name are output, along with the invalid value in hexadecimal format.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Functions

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ABS( ) function

Returns the absolute value of a numeric expression. The absolute value of a number is the number without its sign.

Syntax

```
ABS(number)
```

Parameters

Name	Type	Description
number	numeric	The value to find the absolute value for.

Output

Numeric.

Examples

Basic examples

Returns 7.2:

```
ABS(7.2)
```

Returns 7.2:

```
ABS(-7.2)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

AGE( ) function

Returns the number of elapsed days (the age) between a specified date and a specified cutoff date, or the current operating system date, or the number of elapsed days between any two dates.

Syntax

AGE(*date/datetime/string* <,*cutoff_date*>)

Parameters

Name	Type	Description
date/datetime/string	character datetime	The field, expression, or literal value to age.
cutoff_date optional	character datetime	The field, expression, or literal value to which *date/datetime/string* is compared. If omitted, the current operating system date is used as the cutoff date.

> **Note**
>
> *date/datetime/string* and *cutoff_date* can both accept a datetime value. You cannot use AGE( ) with time values alone.
>
> For more information, see "Using AGE( ) with datetime data" on page 470.

Output

Numeric.

Examples

Basic examples

No cutoff date

Returns the number of days between 31 Dec 2014 and the current date:

- If a positive value is returned, it is equal to the number of days in the past December 31, 2014 occurred

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- If a negative value is returned, it is equal to the number of days in the future December 31, 2014 occurs
- If 0 is returned, December 31, 2014 is the current date

```
AGE(`20141231`)
```

Returns the number of days between each date in the **Due_date** field and the current date:

```
AGE(Due_date)
```

Mixing data types

Returns 518, the number of days between the two specified dates:

```
AGE(`20130731`,`20141231`)
```

```
AGE("20130731","20141231")
```

```
AGE(`20130731`,"20141231")
```

```
AGE(`20130731 235959`,`20141231`)
```

Using cutoff dates and fields

Returns the number of days between each date in the **Due_date** field and the cutoff date of December 31, 2014:

- Dates prior to the cutoff date return a positive value equal to the number of days before the cutoff day they occur
- Dates after the cutoff date return a negative value equal to the number of days after the cutoff day they occur

```
AGE(Due_date, `20141231`)
```

Returns the number of days between December 31, 2014 and each date in the **Due_date** field. Results are the same as the example immediately above, but the sign of the returned values (positive or negative) is reversed:

```
AGE(`20141231`, Due_date)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Comparing dates in fields

Returns the number of days between each date in the **Payment_date** field and a corresponding date in the **Due_date** field:

- Payment dates prior to due dates return a positive value, indicating timely payment
- Payment dates after due dates return a negative value, indicating late payment

```
AGE(Payment_date, Due_date)
```

Returns the number of days between each date in the **Payment_date** field and a corresponding date in the **Due_date** field plus a grace period of 15 days.

- Payment dates prior to due dates, or up to 15 days after due dates, return a positive value
- Payment dates more than 15 days after due dates return a negative value, indicating late payment outside the grace period

```
AGE(Payment_date, Due_date+15)
```

Advanced examples

Extracting overdue payments

Extract the name, amount, and invoice date for each record where the age of the invoice is greater than 180 days, based on a cutoff date of December 31, 2014:

```
EXTRACT FIELDS Name Amount Invoice_Date TO "Overdue" IF AGE(Invoice_Date,`20141231`) > 180
```

Remarks

How it works

The AGE( ) function calculates the number of days between two dates.

When to use AGE( )

Use AGE( ) to compare two dates to determine overdue accounts, to perform aged analyses of balances, or to perform any task that requires the number of elapsed days between two dates.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Negative return values

A negative value is returned if the date specified for *date/datetime/string* is more recent than the date specified as the *cutoff_date*, or the operating system date if no *cutoff_date* is specified.

Returns -518:

```
AGE(`20141231`, `20130731`)
```

If you want the elapsed number of days between two dates to always be a positive number, regardless of which date is more recent, nest the AGE( ) function inside the ABS( ) function.

Returns 518:

```
ABS(AGE(`20141231`, `20130731`))
```

Using AGE( ) with datetime data

The AGE( ) function can accept datetime data in one or both parameters. However, you need to be careful if the time portion of the data includes a UTC offset (timezone indicator).

Datetime data without a UTC offset

The time portion of a datetime value does not affect the date calculation performed by AGE( ) if the time data does not include a UTC offset.

Datetime data with a UTC offset

The time portion of a datetime value can affect the date calculation performed by AGE( ) if the time data in one or both parameters includes a UTC offset. Analytics automatically reconciles the UTC offset before performing the calculation, which can cause the result to change by 1 day if reconciliation moves the time forward or backward through the boundary of midnight.

For more information, see How UTC offsets affect datetime expressions.

Using a field for the cutoff date

Unlike the AGE command, which requires a literal date value for the cutoff date, the AGE( ) function allows you to use a field for the cutoff date.

For example:

```
AGE(Payment_date, Due_date)
```

Using the AGE( ) function in this manner is equivalent to calculating the difference between two date fields by subtracting them in an expression.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

For example:

```
Due_date - Payment_date
```

Parameter details

A datetime field specified for *date/datetime/string* or *cutoff_date* can use any date or datetime format, as long as the field definition correctly defines the format.

Specifying a literal date or datetime value

When specifying a literal date or datetime value for *date/datetime/string* or *cutoff_date*, you are restricted to the formats in the table below, and you must enclose the value in backquotes, or single or double quotation marks - for example, `` `20141231` `` or `"20141231"`

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times. Colons are permitted in character time values.

- **Datetime values** - you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.

Example formats	Example literal values
YYYYMMDD	`` `20141231` `` `"20141231"`
YYMMDD	`` `141231` `` `"141231"`
YYYYMMDD hhmmss	`` `20141231 235959` `` `"20141231 235959"`
YYMMDDthhmm	`` `141231t2359` `` `"141231t2359"`
YYYYMMDDThh	`` `20141231T23` `` `"20141231T23"`
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`` `20141231 235959-0500` `` `"20141231 235959-0500"`
YYMMDD hhmm+/-hh (UTC offset)	`` `141231 2359+01` `` `"141231 2359+01"`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Example formats	Example literal values
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ALLTRIM( ) function

Returns a string with leading and trailing spaces removed from the input string.

Syntax

```
ALLTRIM(string)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to remove leading and trailing spaces from.

Output

Character.

Examples

Basic examples

Returns "Vancouver":

```
ALLTRIM("  Vancouver  ")
```

Returns "New York":

```
ALLTRIM("  New York  ")
```

Advanced examples

Concatenating character fields

Use ALLTRIM( ) to eliminate spaces when you concatenate character fields, such as first name and last

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

name fields, so that the resulting field does not contain multiple blank spaces between the concatenated values.

```
DEFINE FIELD Full_Name COMPUTED ALLTRIM(First_Name) + " " + ALLTRIM(Last_Name)
```

Removing non-breaking spaces

Non-breaking spaces are not removed by the ALLTRIM( ) function.

If you need to remove leading or trailing non-breaking spaces, create a computed field using the following expression:

```
DEFINE FIELD Description_cleaned COMPUTED ALLTRIM(REPLACE(Description, CHR(160), CHR(32)))
```

The REPLACE( ) function replaces any non-breaking spaces with regular spaces, and then ALLTRIM( ) removes any leading or trailing regular spaces.

Remarks

How it works

The ALLTRIM( ) function removes the leading and trailing spaces from a string. Spaces inside the string are not removed.

Related functions

Use the LTRIM( ) function if you want to remove only leading spaces from a string, or the TRIM( ) function if you want to remove only trailing spaces.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ASCII( ) function

Returns the ASCII code for a specified character.

Syntax

```
ASCII(character)
```

Parameters

Name	Type	Description
character	character	The character to identify the ASCII code for. You can specify a quoted character, or a multi-character string, field, or expression. If you specify multiple characters, only the first character is evaluated.

Output

Numeric.

Examples

Basic examples

Returns 65:

```
ASCII("A")
```

Returns 49:

```
ASCII("1")
```

Advanced examples

Extracting a record that starts with a tab character

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Extract records that have a tab character at the beginning of a field called "Description". The ASCII code for a tab character is "9".

```
EXTRACT RECORD TO "Tab_Entries.acl" IF ASCII(Description) = 9
```

Remarks

Testing for non-printable characters

You can use ASCII( ) to test for non-printable characters such as:

- **Nul** - ASCII "0"
- **Tab** - ASCII "9"
- **Line Feed (LF)** - ASCII "10"
- **Carriage Return (CR)** - ASCII "13"

Related functions

ASCII( ) is the inverse of the CHR( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

AT( ) function

Returns a number specifying the starting location of a particular occurrence of a substring within a character value.

Syntax

```
AT(occurrence_num, search_for_string, within_text)
```

Parameters

Name	Type	Description
occurance_num	numeric	The occurrence (instance) of *search_for_string* to return the location of. For example, specify 1 to return the starting location of the first occurrence of *search_for_string*.
search_for_string	character	The substring to search for in *within_text*. This value is case-sensitive. If *search_for_string* includes double quotation marks, you need to enclose the value in single quotation marks: `AT(1,'"test"', Description)`
within_text	character	The value to search in. You can concatenate two or more fields in the *within_text* parameter if you want to search in more than one field in a table: `AT(1,'"test"', Description+Summary)`

Output

Numeric. Returns the starting byte position of the specified occurrence of the *search_for_string* value, or 0 if no matches are found.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Occurrences found

Returns 4:

```
AT(1, "-", "604-669-4225")
```

Returns 8:

```
AT(2, "-", "604-669-4225")
```

Occurrences not found

Returns 0, because there is not a third hyphen in the value:

```
AT(3, "-", "604-669-4225")
```

Returns 0, because there is not a fourth lowercase "a" in the value:

```
AT(4, "a", "Alabama")
```

Groups of characters

Returns 5:

```
AT(2, "iss", "Mississippi")
```

Searching a field

Returns the byte position of the first hyphen in each value in the **Invoice_Number** field:

```
AT(1, "-", Invoice_Number)
```

Advanced examples

Finding invoice numbers in which the second hyphen occurs after the tenth byte position

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You can analyze the consistency of invoice numbers in a table by using the AT( ) function to create a filter like the one below. This filter isolates all records in which the invoice number contains two or more hyphens, and the second hyphen occurs after the tenth byte position:

```
SET FILTER TO AT(2, "-", Invoice_Number) > 10
```

Remarks

When to use AT( )

Use this function to retrieve the following starting positions within a character value:

- the starting position of a substring
- the starting position of a subsequent occurrence of the substring

If you only want to confirm multiple occurrences of the same substring in a field, the OCCURS( ) function is a better alternative. For more information, see "OCCURS( ) function" on page 679.

Return value when *occurrence_num* exceeds the number of occurrences

If *occurrence_num* is greater than the actual number of substring occurrences in *within_text*, the function returns 0 because it cannot find that occurrence of the substring.

Concatenated fields and return values

When you search in more than one field, the value returned for the instance is the starting location of *search_for_string* across all fields that you specify. The concatenated fields are treated like a single field that includes leading and trailing spaces from the individual fields, unless you use the ALLTRIM( ) function to remove spaces.

For example, if you search for the first occurrence of a string in two fields with a width of eight characters each, and the string is found at the beginning of the second field, the return value is 9.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

BETWEEN( ) function

Returns a logical value indicating whether the specified value falls within a range.

Syntax

```
BETWEEN(value, min, max)
```

Parameters

Name	Type	Description
value	character numeric datetime	The field, expression, or literal value to test.
min	character numeric datetime	The minimum value of the range. Can be a field, expression, or literal value.
max	character numeric datetime	The maximum value of the range. Can be a field, expression, or literal value.

> **Note**
>
> The range evaluating to **T** (true) includes the *min* and *max* values.
>
> For information regarding character ranges, see "SET EXACT behavior" on the facing page.

Output

Logical. Returns **T** (true) if *value* is greater than or equal to the *min* value, and less than or equal to the *max* value. Returns **F** (false) otherwise.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Numeric input

Returns T:

```
BETWEEN(500,400,700)
```

Returns F:

```
BETWEEN(100,400,700)
```

Character input

Returns T:

```
BETWEEN("B","A","C")
```

Returns F, because the character comparison is case-sensitive, and lowercase "b" does not fall between uppercase "A" and "C":

```
BETWEEN("b","A","C")
```

Datetime input

Returns T:

```
BETWEEN(`141230`,`141229`,`141231`)
```

Returns T for all values in the **Login_time** field from 07:00:00 AM to 09:00:00 AM, inclusive, and F otherwise:

```
BETWEEN(Login_time,`t070000`,`t090000`)
```

SET EXACT behavior

Returns T for all values in the **Last_Name** field that begin with the letters from "C" to "K", inclusive, and F otherwise (SET EXACT must be OFF):

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
BETWEEN(Last_Name, "C", "K")
```

Returns T for all values in the **Last_Name** field that begin with the letters from "C" to "J", inclusive, and F otherwise (SET EXACT must be ON). Also returns T for the single letter "K":

```
BETWEEN(Last_Name, "C", "K")
```

Field input

Returns T for all values in the **Invoice_Date** field from 30 Sep 2014 to 30 Oct 2014, inclusive, and F otherwise:

```
BETWEEN(Invoice_Date, `20140930`, `20141030`)
```

Returns T for all records in which the invoice date does not fall between the PO date and the paid date, inclusive, and F otherwise:

```
NOT BETWEEN(Invoice_Date, PO_Date, Paid_Date)
```

Returns T for all values in the **Invoice_Amount** field from $1000 to $5000, inclusive, and F otherwise:

```
BETWEEN(Invoice_Amount, 1000, 5000)
```

Advanced examples

Create a filter to view a salary range

The following example opens the **Employee_List** sample table and applies a filter that limits the records displayed to include only employees that earn a salary greater than or equal to $40,000.00, and less than or equal to $50,000.00.

```
OPEN Employee_List
SET FILTER TO BETWEEN(Salary, 40000.00, 50000.00)
```

Create a filter to find dates within a range that changes

You have created a table that joins data from your company's travel and expense system with company credit card data. You want to find any instances where an employee was reimbursed for a hotel room charge that was also charged to the company credit card.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You join the two sets of data on the Amount field and plan to use the dates of the hotel stay and the T&E expense date to confirm that the two amounts refer to the same expense. The problem is that the date in the T&E system can differ by a day or two from the hotel dates in the company credit card data.

The example below opens the **Joined_expense_data** table and applies a filter that finds T&E dates within a range of hotel room dates. By using fields rather than actual date values, the ranges shift with the data.

```
OPEN Joined_expense_data
SET FILTER TO BETWEEN(T_E_date, Arrival_date-2, Arrival_date+2) OR BETWEEN(T_E_date,
Departure_date-2, Departure_date+2)
```

Remarks

Supported data types

Inputs to the BETWEEN( ) function can be numeric, character, or datetime data. You cannot mix data types. All three inputs must belong to the same data category.

Use BETWEEN( ) instead of the AND operator

You can use the BETWEEN( ) function instead of expressions that use the AND operator.

For example:

```
BETWEEN(Invoice_Amount, 1000, 5000)
```

is equivalent to

```
Invoice_Amount >= 1000 AND Invoice_Amount <= 5000
```

The order of *min* and *max*

The order of *min* and *max* in the BETWEEN( ) function does not matter because Analytics automatically identifies which value is the minimum and which value is the maximum.

Both of the examples below return T:

```
BETWEEN(2500, 1000, 5000)
```

```
BETWEEN(2500, 5000, 1000)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Decimal precision of numeric inputs

When the numeric inputs being compared have a different decimal precision, the comparison uses the higher level of precision.

Returns T, because 1.23 is equal to 1.23:

```
BETWEEN(1.23, 1.23, 1.25)
```

Returns F, because 1.23 is less than 1.234 once the third decimal place is considered:

```
BETWEEN(1.23, 1.234, 1.25)
```

Character data

Case sensitivity

The BETWEEN( ) function is case-sensitive when used with character data. When it compares characters, "a" is not equivalent to "A".

Returns F:

```
BETWEEN("B","a","C")
```

If you are working with data that includes inconsistent case, you can use the UPPER( ) function to convert the values to consistent case before using BETWEEN( ).

Returns T:

```
BETWEEN(UPPER("B"), UPPER("a"), UPPER("C"))
```

Partial matching

Partial matching is supported for character comparisons.

value can be contained by *min*.

Returns T, even though *value* "AB" appears to be less than *min* "ABC":

```
BETWEEN("AB", "ABC", "Z")
```

max can be contained by *value*.

Returns T, even though *value* "ZZ" appears to be greater than *max* "Z":

```
BETWEEN("ZZ", "ABC", "Z")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> The shorter value in the character comparison must appear at the start of the longer value to constitute a match.

Partial matching and SET EXACT

Partial matching is enabled when SET EXACT = OFF, which is the Analytics default setting. If SET EXACT = ON, partial matching is disabled and the comparison values must exactly match to constitute a match.

Both examples above are False when SET EXACT is ON.

For more information about SET EXACT (the **Exact Character Comparisons** option), see "SET command" on page 418.

Turning SET EXACT off or on

If you want to ensure that the **Exact Character Comparisons** option is not used with the BETWEEN( ) function, check that the option is deselected in the **Table** tab in the **Options** dialog box (**Tools > Options**).

If you are using a script, you can add the `SET EXACT OFF` command before the BETWEEN( ) function appears. If required, you can restore the previous state with the `SET EXACT ON` command.

Datetime parameters

A date, datetime, or time field specified as a function input can use any date, datetime, or time format, as long as the field definition correctly defines the format.

Mixing date, datetime, and time inputs

You are not prevented from mixing date, datetime, and time values in the BETWEEN( ) function's three inputs, but mixing these Datetime subtypes can give results that are not meaningful.

Analytics uses serial number equivalents to process datetime calculations, so even if you are interested in only the date portion of a datetime value, the time portion still forms part of the calculation.

Consider the following examples:

Returns T, because 31 December 2014 falls within the range specified by *min* and *max*:

```
BETWEEN(`20141231`,`20141229`,`20141231`)
```

Returns F, even though 12:00 PM on 31 December 2014 appears to fall within the range specified by *min* and *max*:

```
BETWEEN(`20141231 120000`,`20141229`,`20141231`)
```

If we look at the serial number equivalent of these two expressions, we can see why the second one evaluates to false.

Returns T, because the serial number *value* is equal to the serial number *max*:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
BETWEEN(42003.000000, 42001.000000, 42003.000000)
```

Returns F, because the serial number *value* is greater than the serial number *max*:

```
BETWEEN(42003.500000, 42001.000000, 42003.000000)
```

The serial number 42003.500000 is greater than 42003.000000 and therefore is out of range, even though the two dates are identical. 0.500000 is the serial number equivalent of 12:00 PM.

Harmonize Datetime subtypes

To avoid the problems that can be caused by mixing Datetime subtypes, you can use functions to harmonize the subtypes.

For example, this expression, which uses the same initial values as the second example above, returns T rather than F:

```
BETWEEN(CTOD(DATE(`20141231
120000`,"YYYYMMDD"),"YYYYMMDD"),`20141229`,`20141231`)
```

Specifying a literal date, datetime, or time value

When specifying a literal date, datetime, or time value for any of the function inputs, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.
- **Time values** – you must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

Example formats	Example literal values
YYYYMMDD	`` `20141231` ``
YYMMDD	`` `141231` ``
YYYYMMDD hhmmss	`` `20141231 235959` ``
YYMMDDthhmm	`` `141231t2359` ``
YYYYMMDDThh	`` `20141231T23` ``

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Example formats	Example literal values
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`20141231 235959-0500`
YYMMDD hhmm+/-hh (UTC offset)	`141231 2359+01`
thhmmss	`t235959`
Thhmm	`T2359`
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

BINTOSTR( ) function

Returns Unicode character data converted from ZONED or EBCDIC character data. Abbreviation for "Binary to String".

> **Note:**
>
> This function is specific to the Unicode edition of Analytics. It is not a supported function in the non-Unicode edition.

Syntax

BINTOSTR(*string*, *string_type*)

Parameters

Name	Type	Description
string	character	The ZONED or EBCDIC value that you want to convert to Unicode character encoding.
string_type	character	The format to convert from. You must specify one of the following values: ◦ "A" - convert from ZONED (ASCII) data ◦ "E" - convert from EBCDIC data

Output

Character.

Examples

Basic examples

The expression ZONED(-6448,4) converts the value -6448 to the character format "644Q", however the Unicode edition of Analytics requires that you convert the output of ZONED( ) to Unicode characters using BINTOSTR( ).

Returns "644Q" in Unicode format:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
BINTOSTR(ZONED(-6448,4), "A")
```

Remarks

When to use BINTOSTR( )

Use this function to convert return values from the ZONED( ) and EBCDIC( ) functions to a Unicode value.

> **Note**
>
> If this function is not applied to the return values of ZONED( ) and EBCDIC( ) in Unicode editions of Analytics, then they are displayed incorrectly because the encoding is not interpreted correctly.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

BIT( ) function

Returns the binary representation for the specified byte position in the current record as an eight character string.

Syntax

```
BIT(byte_location)
```

Parameters

Name	Type	Description
byte_location	numeric	The byte position to return as a binary value.

Output

Character.

Examples

Basic examples

Returns "00110001" if the eighth byte contains "1":

```
BIT(8)
```

Returns "01000001" if the ninth byte contains "A":

```
BIT(9)
```

Returns "01100001" if the seventeenth byte contains "a":

```
BIT(17)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Advanced examples

Using BIT ( ) and SUBSTRING ( ) to extract a value

Assume that byte position 17 contains a set of 8 credit flags.

To extract all customer records that have the third bit set to one (meaning "do not ship"), specify:

```
EXTRACT IF SUBSTRING(BIT(17), 3, 1) = "1"
```

In this example, the SUBSTRING( ) function is used to extract the value of the third bit.

Remarks

How it works

BIT( ) converts the byte at the specified byte position into an eight character string of ones and zeros.

When to use BIT( )

Use BIT( ) to examine the individual bits in a byte.

Related functions

If you want to retrieve the character at the specified byte location, use the BYTE( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

BLANKS( ) function

Returns a string containing a specified number of blank spaces.

Syntax

```
BLANKS(count)
```

Parameters

Name	Type	Description
count	numeric	The number of blank spaces to insert.

Output

Character.

Examples

Basic examples

Returns " ":

```
BLANKS(5)
```

Returns "ABC Corporation":

```
"ABC" + BLANKS(1) + "Corporation"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

When to use BLANKS( )

Use the BLANKS( ) function to harmonize fields, to initialize variables in scripts, or to insert blank spaces when formatting fields or concatenating strings.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

BYTE( ) function

Returns the character stored in the specified byte position in the current record.

Syntax

```
BYTE(byte_location)
```

Parameters

Name	Type	Description
byte_location	numeric	The byte position to return as a character value. The value refers to a position in the record (counting from 1), irrespective of any field definitions.

Output

Character.

Examples

Basic examples

Returns "1" from a record that begins with an ID field containing "1":

```
byte(112)
```

Advanced examples

Identify records in print files or PDFs based on consistent formatting

Use BYTE( ) to identify records in a data file where a particular character is present in a particular byte position. This is typically the case in Print Image (Report) files or Adobe Acrobat (PDF) files where data is formatted in a consistent way throughout the document.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

For example, to locate and extract records that include a period at byte position 113:

```
EXTRACT RECORD IF BYTE(113) = "." TO "Output.fil"
```

Remarks

When to use BYTE( )

Use BYTE( ) to examine the contents of a position in a record, without having to define a field for the purpose.

Using BYTE( ) on EBCDIC data

If you use this function on EBCDIC data, the value returned will also be EBCDIC. You may not be able to compare this to character values.

Related functions

If you want to retrieve the binary representation for specified byte location, use the BIT( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CDOW( ) function

Returns the name of the day of the week for a specified date or datetime. Abbreviation for "Character Day of Week".

Syntax

```
CDOW(date/datetime, length)
```

Parameters

Name	Type	Description
date/datetime	datetime	The field, expression, or literal value to return the day name for.
length	numeric	A value between 1 and 9 that specifies the length of the output string. To display abbreviated day names, specify a smaller value.

Output

Character.

Examples

Basic examples

Returns "Wednesday" because December 31, 2014 falls on a Wednesday, and *length* is 9:

```
CDOW(`20141231`, 9)
```

Returns "Wed" because December 31, 2014 falls on a Wednesday, and *length* is 3:

```
CDOW(`20141231 235959`, 3)
```

Returns the full day name for each value in the **Invoice_date** field:

```
CDOW(Invoice_date, 9)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns the abbreviated day name for each value in the **Receipt_timestamp** field:

```
CDOW(Receipt_timestamp, 3)
```

Advanced examples

Adding a field that identifies the days of the week for dates

Use the CDOW( ) function to create a computed field that identifies the days of the week for all the dates in a date field. Once you have created the computed field, you can add it to the view beside the date column:

```
DEFINE FIELD Name_of_Day COMPUTED CDOW(Trans_Date, 3)
```

Creating a filter to test for transactions that occurred on a weekend

Use the CDOW( ) function to create a filter that isolates transactions that occurred on a weekend:

```
SET FILTER TO CDOW(Trans_Date, 3) = "Sat" OR CDOW(Trans_Date, 3) = "Sun"
```

Remarks

Parameter details

A field specified for *date/datetime* can use any date or datetime format, as long as the field definition correctly defines the format.

If the *length* parameter is shorter than the day name, the day name is truncated to the specified length. If the *length* parameter is longer than the day name, the day name is padded with blank spaces.

Specifying a literal date or datetime value

When specifying a literal date or datetime value for *date/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.
- **Time values** – you must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Example formats	Example literal values
YYYYMMDD	`20141231`
YYMMDD	`141231`
YYYYMMDD hhmmss	`20141231 235959`
YYMMDDthhmm	`141231t2359`
YYYYMMDDThh	`20141231T23`
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`20141231 235959-0500`
YYMMDD hhmm+/-hh (UTC offset)	`141231 2359+01`
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Related functions

If you need to return the day of the week as a number (1 to 7), use DOW( ) instead of CDOW( ).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CHR( ) function

Returns the character associated with the specified ASCII code.

Syntax

```
CHR(number)
```

Parameters

Name	Type	Description
number	numeric	A valid numeric expression between 1 and 255.

Output

Character.

Examples

Basic examples

Returns "A":

```
CHR(65)
```

Returns "1":

```
CHR(49)
```

Advanced examples

Adding the UK pound symbol (£) to each of the values in a currency field

Create a computed field that adds the pound symbol (ASCII code 163) before amounts in the **Invoice_Amount** field. The numeric **Invoice_Amount** field is first converted to a character field, and leading and

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

trailing blank spaces are trimmed.

```
DEFINE FIELD Currency_UK COMPUTED CHR(163)+ALLTRIM(STRING(Invoice_Amount, 12))
```

Remarks

When to use CHR( )

Use the CHR( ) function to return the character associated with any ASCII code, including those characters that cannot be entered directly from a keyboard or displayed on screen. With CHR( ), you can search fields or records for the existence of these specific characters.

Referencing NUL

Referencing the ASCII NUL (null) character, `CHR(0)`, may produce unpredictable results because it is used by Analytics as a text qualifier, and should be avoided if possible.

Related functions

CHR( ) is the inverse of the ASCII( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CLEAN( ) function

Replaces the first invalid character in a string, and all subsequent characters, with blanks.

Syntax

```
CLEAN(string <,extra_invalid_characters>)
```

Parameters

Name	Type	Description
string	character	The value from which to remove default and any extra invalid characters.

extra_invalid_characters
optional
	character	Invalid characters you want to remove from *string* in addition to the default invalid characters. You may specify more than one extra invalid character:
		`" ,;\"`
		Tab characters, null characters, and carriage return and line feed characters are default invalid characters that are automatically removed and do not need to be specified.
		To specify the double quotation mark as an extra invalid character, enclose *extra_invalid_characters* in single quotation marks:
		`""`

Output

Character.

Examples

Basic examples

Returns "ABC " ("ABC" followed by four blank spaces):

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
CLEAN("ABC%DEF","%")
```

Returns "1234.56 " ("1234.56" followed by six blank spaces):

```
CLEAN("1234.56,111,2", ",")
```

Remarks

When to use CLEAN( )

Use this function to ensure that fields containing invalid data are printed correctly. You can also use this function to isolate parts of a field, such as the last name in a customer field that includes both the first and last name of the customer.

Specifying invalid single and double quotation marks

If you need to specify both single and double quotation marks as invalid characters, you must nest the CLEAN( ) function within itself:

```
CLEAN(CLEAN(string, ""), """)
```

Automatic CLEAN( )

In an Analytics script, you can apply the CLEAN( ) function automatically to all character fields by adding `SET CLEAN ON` to the script. You cannot specify extra individual characters using this option.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CMOY( ) function

Returns the name of the month of the year for a specified date or datetime. Abbreviation for "Character Month of Year".

Syntax

```
CMOY(date/datetime, length)
```

Parameters

Name	Type	Description
date/datetime	datetime	The field, expression, or literal value to return the month name for.
length	numeric	A value between 1 and 9 that specifies the length of the output string. To display abbreviated month names, specify a smaller value.

Output

Character.

Examples

Basic examples

Returns "December":

```
CMOY(`20141231`, 9)
```

Returns "Dec":

```
CMOY(`20141231 235959`, 3)
```

Returns the abbreviated month name for each value in the **Receipt_timestamp** field:

```
CMOY(Receipt_timestamp, 3)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns the full month name for each value in the **Invoice_date** field:

```
CMOY(Invoice_date, 9)
```

Returns the full name of the month 15 days after each value in the **Invoice_date** field:

```
CMOY(Invoice_date + 15, 9)
```

Remarks

Parameter details

A field specified for *date/datetime* can use any date or datetime format, as long as the field definition correctly defines the format.

If the *length* parameter is shorter than the month name, the month name is truncated to the specified length. If the *length* parameter is longer than the month name, the month name is padded with blank spaces.

Specifying a literal date or datetime value

When specifying a literal date or datetime value for *date/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.
- **Time values** – you must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

Example formats	Example literal values
YYYYMMDD	`` `20141231` ``
YYMMDD	`` `141231` ``
YYYYMMDD hhmmss	`` `20141231 235959` ``
YYMMDDthhmm	`` `141231t2359` ``
YYYYMMDDThh	`` `20141231T23` ``

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Example formats	Example literal values
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`20141231 235959-0500`
YYMMDD hhmm+/-hh (UTC offset)	`141231 2359+01`
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Related functions

If you need to return the month of the year as a number (1 to 12), use MONTH( ) instead of CMOY( ).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

COS( ) function

Returns the cosine of an angle expressed in radians, with a precision of 15 decimal places.

Syntax

```
COS(radians)
```

Parameters

Name	Type	Description
radians	numeric	The measurement of the angle in radians.

Output

Numeric.

Examples

Basic examples

Returns 0.500000000000000 (the specified number of radians):

```
COS(1.047197551196598)
```

Advanced examples

Using degrees as input

Returns 0.500000000000000 (the cosine of 60 degrees):

```
COS(60 * PI( )/180)
```

Rounding to 3 decimal places

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns 0.500 (the cosine of 60 degrees, rounded to 3 decimal places):

```
DEC(COS(60 * PI( )/180),3)
```

Remarks

Performing the Mantissa Arc Test

The three trigonometric functions in Analytics – SIN( ), COS( ), and TAN( ) – support performing the Mantissa Arc Test associated with Benford's Law.

Converting degrees to radians

If your input is in degrees you can use the PI( ) function to convert the input to radians: (*degrees* * PI( )/180) = *radians*. If required, you can round or truncate the return value using the DEC( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CTOD( ) function

Converts a character or numeric date value to a date. Can also extract the date from a character or numeric datetime value and return it as a date. Abbreviation for "Character to Date".

Syntax

```
CTOD(string/number<,format>)
```

Parameters

Name	Type	Description
string/number	character numeric	The field, expression, or literal value to convert to a date, or from which to extract the date.
format optional	character	The date format of *string/number*. The *format* is required for values that use any date format other than YYYYMMDD or YYMMDD, for example "DD/MM/YYYY". **Note** If you use the CTOD function with a datetime value that requires the *format* parameter, specify only the date portion of the format, and not the time portion. For example: `CTOD("31/12/2014 23:59:59", "DD/MM/YYYY")` Specifying the time portion prevents the results from appearing.

Output

Datetime. The date value is output using the current Analytics date display format.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Character literal input

Returns `20141231` displayed as 31 Dec 2014 assuming a current Analytics date display format of DD MMM YYYY:

```
CTOD("20141231")
```

```
CTOD("31/12/2014", "DD/MM/YYYY")
```

```
CTOD("20141231 235959")
```

Numeric literal input

Returns `20141231` displayed as 31 Dec 2014 assuming a current Analytics date display format of DD MMM YYYY:

```
CTOD(20141231)
```

```
CTOD(31122014, "DDMMYYYY")
```

```
CTOD(20141231.235959)
```

Character field input

Returns each value in the specified character field as a date, using the current Analytics date display format:

```
CTOD(Invoice_date, "DD/MM/YYYY")
```

```
CTOD(Receipt_timestamp)
```

Numeric field input

Returns each value in the specified numeric field as a date, using the current Analytics date display format:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
CTOD(Due_date, "DDMMYYYY")
```

```
CTOD(Payment_timestamp)
```

Advanced examples

Compare a character or numeric field to a date

Use the CTOD( ) function to compare a date against character or numeric fields that contain values representing dates.

The filter below compares two values:

- the numeric **Due_date** field that stores dates as numbers in the format DDMMYYYY
- the literal date value July 1, 2014

```
SET FILTER TO CTOD(Due_date, "DDMMYYYY") < `20140701`
```

Remarks

Required date formats

Character and numeric fields containing date or datetime values must match the formats in the table below. Datetime values can use any combination of the date, separator, and time formats valid for their data type. The date must precede the time, and there must be a separator between the two.

Dates, or the date portion of datetime values, can use any date format supported by Analytics, and valid for the data type, as long as formats other than YYYYMMDD or YYMMDD are correctly defined by *format*.

Date formats	Separator formats	Time formats
Character fields		
YYYYMMDD	single blank space	hhmmss hh:mm:ss
YYMMDD	the letter 't'	hhmm hh:mm
any Analytics-supported date format, valid for the data type, if defined by *format*	the letter 'T'	hh
		+/-hhmm +/-hh:mm (UTC offset)

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Date formats	Separator formats	Time formats
		+/-hh (UTC offset)

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Date formats	Separator formats	Time formats
		Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hh

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Date formats	Separator formats	Time formats
Numeric fields		
YYYYMMDD	decimal point	hhmmss
YYMMDD		hhmm
any Analytics-supported date format, valid for the data type, if defined by *format*		hh

Other datetime conversion functions

Character or Numeric to Datetime conversion

Function	Description
CTODT( )	Converts a character or numeric datetime value to a datetime. Abbreviation for "Character to Datetime".
CTOT( )	Converts a character or numeric time value to a time. Can also extract the time from a character or numeric datetime value and return it as a time. Abbreviation for "Character to Time".

Datetime to Character conversion

Function	Description
DATE( )	Extracts the date from a specified date or datetime and returns it as a character string. Can also return the current operating system date.
DATETIME( )	Converts a datetime to a character string. Can also return the current operating system datetime.
TIME( )	Extracts the time from a specified time or datetime and returns it as a character string. Can also return the current operating system time.

Serial to Datetime conversion

Function	Description
STOD( )	Converts a serial date - that is, a date expressed as an integer - to a date value. Abbreviation for "Serial to Date".
STODT( )	Converts a serial datetime - that is, a datetime expressed as an integer, and a fractional portion of 24 hours - to a datetime value. Abbreviation for "Serial to Datetime".

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Function	Description
STOT( )	Converts a serial time - that is, a time expressed as a fractional portion of 24 hours, with 24 hours equaling 1 - to a time value. Abbreviation for "Serial to Time".

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CTODT( ) function

Converts a character or numeric datetime value to a datetime. Abbreviation for "Character to Datetime".

Syntax

```
CTODT(string/number<,format>)
```

Parameters

Name	Type	Description
string/number	character numeric	The field, expression, or literal value to convert to a datetime.
format optional	character	The date format of *string/number*. The *format* is required for values that use any date format other than YYYYMMDD or YYMMDD for the date portion of the value, for example "DD/MM/YYYY".

Output

Datetime. The datetime value is output using the current Analytics date and time display formats.

Examples

Basic examples

Character literal input

Returns `20141231t235959` displayed as 31 Dec 2014 23:59:59 assuming a current Analytics date and time display formats of DD MMM YYYY and hh:mm:ss:

```
CTODT("20141231 235959")
```

```
CTODT("31/12/2014 23:59:59", "DD/MM/YYYY hh:mm:ss")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Numeric literal input

Returns `20141231t235959` displayed as 31 Dec 2014 23:59:59 assuming a current Analytics date and time display formats of DD MMM YYYY and hh:mm:ss:

```
CTODT(20141231.235959)
```

```
CTODT(31122014.235959, "DDMMYYYY.hhmmss")
```

Character field input

Returns each value in the **Receipt_timestamp** character field as a datetime, using the current Analytics date display format:

```
CTODT(Receipt_timestamp, "DD/MM/YYYY hh:mm:ss")
```

Numeric field input

Returns each value in the **Payment_timestamp** numeric field as a datetime, using the current Analytics date display format:

```
CTODT(Payment_timestamp, "DD/MM/YYYY hh:mm:ss")
```

Advanced examples

Compare a character or numeric field to a datetime

Use the CTODT( ) function to compare a datetime against character or numeric fields that contain values representing datetimes.

The filter below compares two values:

- the character **Receipt_timestamp** field that stores datetimes as character data in the format DD/MM/YYYY hh:mm:ss
- the literal datetime value July 1, 2014 13:30:00

```
SET FILTER TO CTODT(Receipt_timestamp, "DD/MM/YYYY hh:mm:ss") < `20140701t133000`
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Required datetime formats

Character and numeric fields containing datetime values must match the formats in the table below. The datetime values can use any combination of the date, separator, and time formats valid for their data type. The date must precede the time, and there must be a separator between the two.

The date portion of values can use any date format supported by Analytics, and valid for the data type, as long as formats other than YYYYMMDD or YYMMDD are correctly defined by *format*. If you use *format*, you must also specify the time format, which must be one of the time formats that appear in the table below.

Analytics automatically recognizes the separator between the date and time portions of datetime values, so there is no need to specify the separator in *format*. You can specify the separator if you want to.

Date formats	Separator formats	Time formats
Character fields		
YYYYMMDD	single blank space	hhmmss hh:mm:ss
YYMMDD	the letter 't'	hhmm hh:mm
any Analytics-supported date format, valid for the data type, if defined by *format*	the letter 'T'	hh
		+/-hhmm +/-hh:mm (UTC offset)
		+/-hh (UTC offset)

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Date formats	Separator formats	Time formats
		Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Date formats	Separator formats	Time formats
Numeric fields		
YYYYMMDD	decimal point	hhmmss
YYMMDD		hhmm
any Analytics-supported date format, valid for the data type, if defined by *format*		hh

Other datetime conversion functions

Character or Numeric to Datetime conversion

Function	Description
CTOD( )	Converts a character or numeric date value to a date. Can also extract the date from a character or numeric datetime value and return it as a date. Abbreviation for "Character to Date".
CTOT( )	Converts a character or numeric time value to a time. Can also extract the time from a character or numeric datetime value and return it as a time. Abbreviation for "Character to Time".

Datetime to Character conversion

Function	Description
DATE( )	Extracts the date from a specified date or datetime and returns it as a character string. Can also return the current operating system date.
DATETIME( )	Converts a datetime to a character string. Can also return the current operating system datetime.
TIME( )	Extracts the time from a specified time or datetime and returns it as a character string. Can also return the current operating system time.

Serial to Datetime conversion

Function	Description
STOD( )	Converts a serial date - that is, a date expressed as an integer - to a date value. Abbreviation for "Serial to Date".
STODT( )	Converts a serial datetime - that is, a datetime expressed as an integer, and a fractional portion of 24 hours - to a datetime value. Abbreviation for "Serial to Datetime".

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Function	Description
STOT( )	Converts a serial time - that is, a time expressed as a fractional portion of 24 hours, with 24 hours equaling 1 - to a time value. Abbreviation for "Serial to Time".

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CTOT( ) function

Converts a character or numeric time value to a time. Can also extract the time from a character or numeric datetime value and return it as a time. Abbreviation for "Character to Time".

Syntax

```
CTOT(string/number)
```

Parameters

Name	Type	Description
string/number	character numeric	The field, expression, or literal value to convert to a time, or from which to extract the time.

Output

Datetime. The time value is output using the current Analytics time display format.

Examples

Basic examples

Character literal input

Returns `t235959` displayed as 23:59:59 assuming a current Analytics time display format of hh:mm:ss:

```
CTOT("t235959")
```

```
CTOT("23:59:59")
```

```
CTOT("20141231 235959")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Numeric literal input

Returns `t235959` displayed as 23:59:59 assuming a current Analytics time display format of hh:mm:ss:

```
CTOT(.235959)
```

```
CTOT(0.235959)
```

```
CTOT(20141231.235959)
```

Character field input

Returns each value in the **Login_time** character field as a time, using the current Analytics time display format:

```
CTOT(Login_time)
```

Numeric field input

Returns each value in the **Payment_datetime** numeric field as a time, without any date portion, using the current Analytics time display format:

```
CTOT(Payment_datetime)
```

Advanced examples

Compare a character or numeric field to a time

Use the CTOT( ) function to compare a time against character or numeric fields that contain values representing times.

The filter below compares two values:

- the numeric **Login_time** field that stores times as numeric data
- the literal time value 09:30:00

```
SET FILTER TO CTOT(Login_time) > `t093000`
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Required datetime formats

Character and numeric fields containing time or datetime values must match the formats in the table below.

Time values can use any combination of separator and time format. There must be a separator before the time value, or colons between the time components, for the function to operate correctly.

Datetime values can use any combination of the date, separator, and time formats valid for their data type. The date must precede the time, and there must be a separator between the two.

Use the CTOD( ) function if you want to convert a character or numeric date value to a date, or extract the date from a character or numeric datetime value and return it as a date.

Use the CTODT( ) function if you want to convert a character or numeric datetime value to a datetime.

Date formats	Separator formats	Time formats
Character fields		
YYYYMMDD	single blank space	hhmmss hh:mm:ss
YYMMDD	the letter 't'	hhmm hh:mm
	the letter 'T'	hh
		+/-hhmm +/-hh:mm (UTC offset)
		+/-hh (UTC offset)
		Note: Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+h-hmm. Results can be unreliable.)
Numeric fields		
YYYYMMDD	decimal point	hhmmss

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Date formats	Separator formats	Time formats
YYMMDD		hhmm
		hh

Other datetime conversion functions

Character or Numeric to Datetime conversion

Function	Description
CTOD( )	Converts a character or numeric date value to a date. Can also extract the date from a character or numeric datetime value and return it as a date. Abbreviation for "Character to Date".
CTODT( )	Converts a character or numeric datetime value to a datetime. Abbreviation for "Character to Datetime".

Datetime to Character conversion

Function	Description
DATE( )	Extracts the date from a specified date or datetime and returns it as a character string. Can also return the current operating system date.
DATETIME( )	Converts a datetime to a character string. Can also return the current operating system datetime.
TIME( )	Extracts the time from a specified time or datetime and returns it as a character string. Can also return the current operating system time.

Serial to Datetime conversion

Function	Description
STOD( )	Converts a serial date - that is, a date expressed as an integer - to a date value. Abbreviation for "Serial to Date".
STODT( )	Converts a serial datetime - that is, a datetime expressed as an integer, and a fractional portion of 24 hours - to a datetime value. Abbreviation for "Serial to Datetime".
STOT( )	Converts a serial time - that is, a time expressed as a fractional portion of 24 hours, with 24 hours equaling 1 - to a time value. Abbreviation for "Serial to Time".

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CUMIPMT( ) function

Returns the cumulative interest paid on a loan during a range of periods.

Syntax

CUMIPMT(*rate*, *periods*, *amount*, *start_period*, *end_period* <,*type*>)

Parameters

Name	Type	Description
rate	numeric	The interest rate per period.
periods	numeric	The total number of payment periods.
amount	numeric	The principal amount of the loan.
start_period	numeric	The first period in the calculation. *start_period* cannot be 0.
end_period	numeric	The last period in the calculation. *end_period* cannot be greater than the total number of payment periods.
type optional	numeric	The timing of payments: ◦ 0 - payment at the end of a period ◦ 1 - payment at the beginning of a period If omitted, the default value of 0 is used.

> **Note**
>
> You must use consistent time periods when specifying *rate* and *periods* to ensure that you are specifying interest rate **per period**.
>
> For example:
>
> - for monthly payments on a two-year loan or investment with interest of 5% per annum, specify 0.05/12 for *rate* and 2 * 12 for *periods*
> - for annual payments on the same loan or investment, specify 0.05 for *rate* and 2 for *periods*

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Output

Numeric.

Examples

Basic examples

Returns 17437.23, the total amount of interest paid in the second year of a twenty-five year, $275,000 loan at 6.5% per annum, with payments due at the end of the month:

```
CUMIPMT(0.065/12, 12*25, 275000, 13, 24, 0)
```

Returns 17741.31, the total amount of interest paid on the same loan in the first year of the loan:

```
CUMIPMT(0.065/12, 12*25, 275000, 1, 12, 0)
```

Remarks

Related functions

The CUMPRINC( ) function is the complement of the CUMIPMT( ) function.

The IPMT( ) function calculates interest paid for a single period.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

CUMPRINC( ) function

Returns the cumulative principal paid on a loan during a range of periods.

Syntax

CUMPRINC(*rate*, *periods*, *amount*, *start_period*, *end_period* <,*type*>)

Parameters

Name	Type	Description
rate	numeric	The interest rate per period.
periods	numeric	The total number of payment periods.
amount	numeric	The principal amount of the loan.
start_period	numeric	The first period in the calculation. *start_period* cannot be 0.
end_period	numeric	The last period in the calculation. *end_period* cannot be greater than the total number of payment periods.
type optional	numeric	The timing of payments: ◦ 0 - payment at the end of a period ◦ 1 - payment at the beginning of a period If omitted, the default value of 0 is used.

> **Note**
>
> You must use consistent time periods when specifying *rate* and *periods* to ensure that you are specifying interest rate **per period**.
>
> For example:
>
> - for monthly payments on a two-year loan or investment with interest of 5% per annum, specify 0.05/12 for *rate* and 2 * 12 for *periods*
> - for annual payments on the same loan or investment, specify 0.05 for *rate* and 2 for *periods*

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Output

Numeric.

Examples

Basic examples

Returns 4844.61, the total amount of principal paid in the second year of a twenty-five year, $275,000 loan at 6.5% per annum, with payments due at the end of the month:

```
CUMPRINC(0.065/12, 12*25, 275000, 13, 24, 0)
```

Returns 367.24, the amount of principal paid on the same loan in the first month of the loan:

```
CUMPRINC(0.065/12, 12*25, 275000, 1, 1, 0)
```

Remarks

Related functions

The CUMIPMT( ) function is the complement of the CUMPRINC( ) function.

The PPMT( ) function calculates principal paid for a single period.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DATE( ) function

Extracts the date from a specified date or datetime and returns it as a character string. Can also return the current operating system date.

Syntax

```
DATE(<date/datetime> <,format>)
```

Parameters

Name	Type	Description
date/datetime optional	datetime	The field, expression, or literal value to extract the date from. If omitted, the current operating system date is returned.
format optional	character	The format to apply to the output string, for example "DD/MM/YYYY". If omitted, the current Analytics date display format is used. You cannot specify a *format* if you have omitted *date/datetime*.

Output

Character.

Examples

Basic examples

Returns "20141231" in the current Analytics date display format:

```
DATE(`20141231 235959`)
```

Returns "31-Dec-2014":

```
DATE(`20141231 235959`, "DD-MMM-YYYY")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns the current operating system date as a character string, using the current Analytics date display format:

```
DATE()
```

Returns each value in the **Receipt_timestamp** field as a character string using the current Analytics date display format:

```
DATE(Receipt_timestamp)
```

Returns each value in the **Receipt_timestamp** field as a character string using the specified date display format:

```
DATE(Receipt_timestamp, "DD/MM/YYYY")
```

Remarks

Output string length

The length of the output string is always 12 characters. If the specified output *format*, or the Analytics date display format, is less than 12 characters, the output string is padded with trailing blank spaces.

Parameter details

A field specified for *date/datetime* can use any date or datetime format, as long as the field definition correctly defines the format.

If you use *format* to control how the output string is displayed, you can use any supported Analytics date display format. For example:

- DD/MM/YYYY
- MM-DD-YY
- DD MMM YYYY

format must be specified using single or double quotation marks – for example, `"DD MMM YYYY"`.

Specifying a literal date or datetime value

When specifying a literal date or datetime value for *date/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- **Datetime values** - you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.
- **Time values** - you must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

Example formats	Example literal values
YYYYMMDD	`20141231`
YYMMDD	`141231`
YYYYMMDD hhmmss	`20141231 235959`
YYMMDDthhmm	`141231t2359`
YYYYMMDDThh	`20141231T23`
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`20141231 235959-0500`
YYMMDD hhmm+/-hh (UTC offset)	`141231 2359+01`
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Related functions

If you need to return the current operating system date as a datetime value, use TODAY( ) instead of DATE( ).

Other datetime conversion functions

Datetime to Character conversion

Function	Description
DATETIME( )	Converts a datetime to a character string. Can also return the current operating system datetime.
TIME( )	Extracts the time from a specified time or datetime and returns it as a character string. Can also

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Function	Description
	return the current operating system time.

Character or Numeric to Datetime conversion

Function	Description
CTOD( )	Converts a character or numeric date value to a date. Can also extract the date from a character or numeric datetime value and return it as a date. Abbreviation for "Character to Date".
CTODT( )	Converts a character or numeric datetime value to a datetime. Abbreviation for "Character to Datetime".
CTOT( )	Converts a character or numeric time value to a time. Can also extract the time from a character or numeric datetime value and return it as a time. Abbreviation for "Character to Time".

Serial to Datetime conversion

Function	Description
STOD( )	Converts a serial date - that is, a date expressed as an integer - to a date value. Abbreviation for "Serial to Date".
STODT( )	Converts a serial datetime - that is, a datetime expressed as an integer, and a fractional portion of 24 hours - to a datetime value. Abbreviation for "Serial to Datetime".
STOT( )	Converts a serial time - that is, a time expressed as a fractional portion of 24 hours, with 24 hours equaling 1 - to a time value. Abbreviation for "Serial to Time".

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DATETIME( ) function

Converts a datetime to a character string. Can also return the current operating system datetime.

Syntax

```
DATETIME(<datetime> <,format>)
```

Parameters

Name	Type	Description
datetime optional	datetime	The field, expression, or literal value to convert. If omitted, the current operating system date is returned.
format optional	character	The format to apply to the output string, for example "DD/MM/YYYY". If omitted, the current Analytics date display format is used. You cannot specify a *format* if you have omitted *date/datetime*.

Output

Character.

Examples

Basic examples

Literal datetime input

Returns "20141231 235959" in the current Analytics date and time display formats:

```
DATETIME(`20141231 235959`)
```

Returns "31-Dec-2014 11:59 P":

```
DATETIME(`20141231 235959`, "DD-MMM-YYYY hh:mm A")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns the current operating system date and time as a character string, using the current Analytics date and time display formats:

```
DATETIME()
```

Field input

Returns each value in the **Receipt_timestamp** field as a character string using the current Analytics date and time display formats:

```
DATETIME(Receipt_timestamp)
```

Returns each value in the **Receipt_timestamp** field as a character string using the specified date and time display formats:

```
DATETIME(Receipt_timestamp, "DD/MM/YYYY hh:mm:ss")
```

Remarks

Output string length

The length of the output string is always 27 characters. If the specified output *format*, or the Analytics date and time display formats, are less than 27 characters, the output string is padded with trailing blank spaces.

Parameter details

A field specified for *datetime* can use any datetime format, as long as the field definition correctly defines the format.

If you use *format* to control how the output string is displayed, you are restricted to the formats in the table below.

- You can use any combination of date, time, and AM/PM formats.
- The date must precede the time. Placing a separator between the two is not required as Analytics automatically uses a single space as a separator in the output string.
- The AM/PM format is optional, and is placed last.
- *format* must be specified using single or double quotation marks.

  For example: `"DD-MMM-YYYY hh:mm:ss AM"`

Date formats	Time formats	AM/PM formats	Examples
all supported Analytics date display formats	hh:mm:ss	none 24-hour clock	"DD/MM/YYYY hh:mm:ss"

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Date formats	Time formats	AM/PM formats	Examples
	hhmmss	AM, or PM 12-hour clock	"MMDDYY hhmmss PM"
	hh:mm	A, or P 12-hour clock	"DD-MMM-YYYY hh:mm A"
	hhmm		
	hh		

Specifying a literal datetime value

When specifying a literal datetime value for *datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231 235959` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.
- **Time values** – you must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

Example formats	Example literal values
YYYYMMDD hhmmss	`` `20141231 235959` ``
YYMMDDthhmm	`` `141231t2359` ``
YYYYMMDDThh	`` `20141231T23` ``
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`` `20141231 235959-0500` ``
YYMMDD hhmm+/-hh (UTC offset)	`` `141231 2359+01` ``
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Other datetime conversion functions

Datetime to Character conversion

Function	Description
DATE( )	Extracts the date from a specified date or datetime and returns it as a character string. Can also return the current operating system date.
TIME( )	Extracts the time from a specified time or datetime and returns it as a character string. Can also return the current operating system time.

Character or Numeric to Datetime conversion

Function	Description
CTOD( )	Converts a character or numeric date value to a date. Can also extract the date from a character or numeric datetime value and return it as a date. Abbreviation for "Character to Date".
CTODT( )	Converts a character or numeric datetime value to a datetime. Abbreviation for "Character to Datetime".
CTOT( )	Converts a character or numeric time value to a time. Can also extract the time from a character or numeric datetime value and return it as a time. Abbreviation for "Character to Time".

Serial to Datetime conversion

Function	Description
STOD( )	Converts a serial date - that is, a date expressed as an integer - to a date value. Abbreviation for "Serial to Date".
STODT( )	Converts a serial datetime - that is, a datetime expressed as an integer, and a fractional portion of 24 hours - to a datetime value. Abbreviation for "Serial to Datetime".
STOT( )	Converts a serial time - that is, a time expressed as a fractional portion of 24 hours, with 24 hours equaling 1 - to a time value. Abbreviation for "Serial to Time".

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DAY( ) function

Extracts the day of the month from a specified date or datetime and returns it as a numeric value (1 to 31).

Syntax

```
DAY(date/datetime)
```

Parameters

Name	Type	Description
date/datetime	datetime	The field, expression, or literal value to extract the day from.

Output

Numeric.

Examples

Basic examples

Returns 31:

```
DAY(`20141231`)
```

```
DAY(`20141231 235959`)
```

Returns the day of the month for each value in the **Invoice_date** field:

```
DAY(Invoice_date)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Parameter details

A field specified for *date/datetime* can use any date or datetime format, as long as the field definition correctly defines the format.

Specifying a literal date or datetime value

When specifying a literal date or datetime value for *date/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.
- **Time values** – you must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

Example formats	Example literal values
YYYYMMDD	`` `20141231` ``
YYMMDD	`` `141231` ``
YYYYMMDD hhmmss	`` `20141231 235959` ``
YYMMDDthhmm	`` `141231t2359` ``
YYYYMMDDThh	`` `20141231T23` ``
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`` `20141231 235959-0500` ``
YYMMDD hhmm+/-hh (UTC offset)	`` `141231 2359+01` ``
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Related functions

If you need to return:

- the day of the week as a number (1 to 7), use DOW( ) instead of DAY( )
- the name of the day of the week, use CDOW( ) instead of DAY( )

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DBYTE( ) function

Returns the Unicode character located at the specified byte position in a record.

> **Note**
>
> This function is specific to the Unicode edition of Analytics. It is not a supported function in the non-Unicode edition.

Syntax

```
DBYTE(byte_location)
```

Parameters

Name	Type	Description
byte_location	numeric	The byte position to return as a character value. To return a meaningful value, you must specify the starting point of the double-byte character, which means that you should only specify odd numbers in the *byte_location* parameter.

Output

Character.

Examples

Basic examples

The examples illustrate the behavior of the function when applied to the following Unicode value, which contains 11 characters (22 bytes) 美丽 **10072DOE**:

Returns "丽":

```
DBYTE(3)
```

Returns "D":

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DBYTE(17)
```

Returns "E":

```
DBYTE(21)
```

Remarks

When to use DBYTE( )

Use DBYTE( ) to examine the contents of a position in a record, without having to define a field for this purpose.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DEC( ) function

Returns a value, or the result of a numeric expression, with the specified number of decimal places.

Syntax

```
DEC(number, decimals)
```

Parameters

Name	Type	Description
number	numeric	The value or result to adjust the number of decimal places for. ◦ **integers** - decimal places are added to the end of *number* as trailing zeros. ◦ **fractional numbers** - If the number of decimal places is reduced, *number* is rounded, not truncated. If the number of decimal places is increased, trailing zeros are added to the end of *number*.
decimals	numeric	The number of decimal places to use in the return value. **Note** You cannot use DEC( ) to increase the decimal precision of results. For information about how to increase decimal precision, see Controlling rounding and decimal precision in numeric expressions.

Output

Numeric.

Examples

Basic examples

Returns 7.00:

```
DEC(7, 2)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns 7.565:

```
DEC(7.5647, 3)
```

Returns 7.56470:

```
DEC(7.5647, 5)
```

Advanced examples

Calculating daily interest

Calculates the daily interest to six decimal places for a field called **Annual_rate**:

```
DEC(Annual_rate, 6) / 365
```

Remarks

When to use DEC( )

Use this function to adjust the number of decimal places in a field, or when you want to round a value or a result to a specified number of decimal places.

DEC( ) cannot reverse fixed-point rounding

You cannot use the DEC( ) function to reverse the standard rounding that fixed-point arithmetic performs in numeric expressions.

Example

Consider the following series of expressions in Analytics:

```
1.1 * 1.1 = 1.2
1.1 * 1.10 = 1.21
DEC(1.1 * 1.1, 2) = 1.20
```

Fixed-point rounding means that the result of `1.1 * 1.1` is 1.2, not 1.21, which is the unrounded result. Using DEC( ) to specify a two-decimal-place result does not create two-decimal-place precision. Instead, it adds a trailing zero to create the specified number of decimal places, without increasing precision.

For information about how to increase decimal precision, see Controlling rounding and decimal precision in numeric expressions.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Related functions

If you want to round a value to the nearest whole number, use the "ROUND( ) function" on page 769.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DHEX( ) function

Converts a Unicode string to a hexadecimal string.

> **Note:**
>
> This function is specific to the Unicode edition of Analytics. It is not a supported function in the non-Unicode edition.

Syntax

```
DHEX(field)
```

Parameters

Name	Type	Description
field	character	The Unicode string to convert to a hexadecimal string.

Output

Character.

Examples

Basic examples

Returns "004100420043003100320033":

```
DHEX("ABC123")
```

Remarks

How it works

DHEX( ) displays each double-byte character using big-endian format, with the most significant double-byte stored first.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Each character is represented by a four-character code. The output string is four times as long as the *field* value, and includes the digits between 0 and 9 and letters between A and F that make up the hexadecimal values.

Related functions

DHEX( ) is the inverse of the HTOU( ) function, which converts a hexadecimal string to a Unicode string.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DICECOEFFICIENT( ) function

Returns the Dice's coefficient of two specified strings, which is a measurement of how similar the two strings are.

Syntax

DICECOEFFICIENT(*string1*, *string2* <,*ngram*>)

Parameters

Name	Type	Description
string1	character	The first string in the comparison.
string2	character	The second string in the comparison.
ngram optional	numeric	The *n*-gram length to use. Specify a whole number, 1 or greater. Increasing the *ngram* length makes the criterion for similarity between two strings stricter. If you do not specify a length, the default length of 2 is used. *N*-grams are overlapping substrings (character blocks) into which the comparison strings are divided as part of the Dice's coefficient calculation. For detailed information, see "Remarks" on page 549.

Output

Numeric. The value is the Dice's coefficient of the two strings, which represents the percentage of the total number of *n*-grams in the two strings that are identical. The range is 0.0000 to 1.0000, inclusive.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

How the *n*-gram length affects the result

The three examples below compare the same two strings. The degree of similarity returned varies depending on the specified *n*-gram length.

Returns 0.9167 (using the default *n*-gram length (2), the *n*-grams in the two strings are 92% identical):

```
DICECOEFFICIENT("125 SW 39TH ST, Suite 100","Suite 100, 125 SW 39TH ST")
```

Returns 1.0000 (using an *n*-gram length of 1, the *n*-grams in the two strings are 100% identical):

```
DICECOEFFICIENT("125 SW 39TH ST, Suite 100","Suite 100, 125 SW 39TH ST", 1)
```

Returns 0.8261 (using an *n*-gram length of 3, the *n*-grams in the two strings are 83% identical):

```
DICECOEFFICIENT("125 SW 39TH ST, Suite 100","Suite 100, 125 SW 39TH ST", 3)
```

Field input

Returns the Dice's coefficient of each value in the **Address** field when compared to the string "125 SW 39TH ST, Suite 100" (based on the default *n*-gram length of 2):

```
DICECOEFFICIENT(Address,"125 SW 39TH ST, Suite 100")
```

Advanced examples

Working with transposed elements

By reducing the *n*-gram length, and removing non-essential characters, you can optimize DICECOEFFICIENT( ) when searching for transposed elements.

Returns 0.7368 (using the default *n*-gram length (2), the *n*-grams in the two strings are 74% identical):

```
DICECOEFFICIENT("John Smith","Smith, John")
```

Returns 1.0000 (by excluding the comma between last name and first name, and by using an *n*-gram length of 1, the *n*-grams in the two strings are 100% identical):

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DICECOEFFICIENT("John Smith", EXCLUDE("Smith, John", ","), 1)
```

Ranking values against "125 SW 39TH ST, Suite 100"

Create the computed field **Dice_Co** to display the Dice's coefficient between "125 SW 39TH ST, Suite 100" and each value in the **Address** field:

```
DEFINE FIELD Dice_Co COMPUTED DICECOEFFICIENT(Address,"125 SW 39TH ST, Suite 100")
```

Add the computed field **Dice_Co** to the view, and then quick sort it in descending order, to rank all values in the **Address** field based on their similarity to "125 SW 39TH ST, Suite 100".

Isolating fuzzy duplicates for "125 SW 39TH ST, Suite 100"

Create a filter that isolates all values in the **Address** field that are within a specified degree of similarity to "125 SW 39TH ST, Suite 100":

```
SET FILTER TO DICECOEFFICIENT(Address,"125 SW 39TH ST, Suite 100") > 0.5
```

Changing the number in the expression allows you to adjust the degree of similarity in the filtered values.

Remarks

When to use DICECOEFFICIENT( )

Use the DICECOEFFICIENT( ) function to find nearly identical values (fuzzy duplicates). You can also use DICECOEFFICIENT( ) to find values with identical or near-identical content, but transposed elements. For example:

- telephone numbers, or social security numbers, with transposed digits
- versions of the same address, formatted differently

How it works

DICECOEFFICIENT( ) returns the Dice's coefficient of the two evaluated strings, which is a measurement of the degree of similarity between the strings, on a scale from 0.0000 to 1.0000. The greater the returned value the more similar the two strings:

- **1.0000** – means that each string is composed of an identical set of characters, although the characters may be in a different order, and may use different case.
- **0.7500** – means the *n*-grams in the two strings are 75% identical.
- **0.0000** – means the two strings have no shared *n*-grams (explained below), or the specified length of the *n*-gram used in the calculation is longer than the shorter of the two strings being compared.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Usage tips

- **Filtering or sorting** – Filtering or sorting the values in a field based on their Dice's coefficient identifies those values that are most similar to the comparison string.
- **Case-sensitivity** – The function is not case-sensitive, so "SMITH" is equivalent to "smith."
- **Leading and trailing blanks** – The function automatically trims leading and trailing blanks in fields, so there is no need to use the TRIM( ) or ALLTRIM( ) functions when specifying a field as a parameter.
- **Removing generic elements** – The OMIT( ) and EXCLUDE( ) functions can improve the effectiveness of the DICECOEFFICIENT( ) function by removing generic elements such as "Corporation" or "Inc.", or characters such as commas, periods, and ampersands (&), from field values.

  Removal of generic elements and punctuation focuses the DICECOEFFICIENT( ) string comparison on just the portion of the strings where a meaningful difference may occur.

How Dice's coefficient is calculated

Dice's coefficient represents the percentage of the total number of *n*-grams in two strings that are identical.

Dice's coefficient is calculated by first dividing the strings being compared into *n*-grams. *N*-grams (also referred to as *q*-grams) are overlapping substrings, or overlapping character blocks, with a length of *n*. You can specify the length of *n* using the *ngram* parameter, or accept the default length of 2.

Two names divided into *n*-grams

Here are the names "John Smith" and "Smith, John D." divided into *n*-grams with a length of 2, and *n*-grams with a length of 3. Underscores indicate spaces. Internal spaces and punctuation are counted as characters.

n-gram length	"John Smith" *n*-grams	"Smith, John D." *n*-grams
2	Jo | oh | hn | n_ | _S | Sm | mi | it | th	Sm | mi | it | th | h, | ,_ | _J | Jo | oh | hn | n_ | _D | D.
3	Joh | ohn | hn_ | n_S | _Sm | Smi | mit | ith	Smi | mit | ith | th, | h,_ | ,_J | _Jo | Joh | ohn | hn_ | n_D | _D.

The Dice's coefficient formula

Once the *n*-grams have been established for two strings being compared, the calculation is completed using the following formula:

- *2 x the number of shared n-grams / the total number of n-grams in both strings*

Shared *n*-grams are *n*-grams that appear in both strings. For example, "ABC" and "BCD" share the *n*-gram "BC", assuming an *n*-gram length of 2 (AB \| **BC** and **BC** \| CD).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples of calculating the Dice's coefficient

The table below illustrates calculating the Dice's coefficient for the two strings, "John Smith" and "Smith, John D.", using different *n*-gram lengths.

Note that as the *n*-gram length increases for the same pair of strings, the Dice's coefficient value decreases, indicating less similarity. Although the strings remain the same, the criterion for similarity becomes stricter because dividing the strings into longer *n*-grams means that longer sequences of characters must match for an *n*-gram to qualify as shared.

Another way of thinking about this point is that the relative position of characters is weighted more heavily as you increase the *n*-gram length. By contrast, the relative position of characters is not considered when using an *n*-gram length of 1. Relative position refers to the position of characters in relation to one another, rather than to their absolute position within a string.

> **Tip**
>
> If you are specifically looking for transposition, use an *n*-gram length of 1.

n-gram length	"John Smith" *n*-grams	"Smith, John D." *n*-grams	Shared *n*-grams	Dice's coef-ficient
1	J | o | h | n | _ | S | m | i | t | h (10 *n*-grams)	S | m | i | t | h | , | _ | J | o | h | n | _ | D | . (14 *n*-grams)	10	2x10 / (10+14) = **0.8333**
2 (default)	Jo | oh | hn | n_ | _S | Sm | mi | it | th (9 *n*-grams)	Sm | mi | it | th | h, | ,_ | _J | Jo | oh | hn | n_ | _D | D. (13 *n*-grams)	8	2x8 / (9+13) = **0.7273**
3	Joh | ohn | hn_ | n_S | _Sm | Smi | mit | ith (8 *n*-grams)	Smi | mit | ith | th, | h,_ | ,_J | _Jo | Joh | ohn | hn_ | n_D | _D. (12 *n*-grams)	6	2x6 / (8+12) = **0.6000**
4	John | ohn_ | hn_S | n_Sm | _ Smi | Smit | mith (7 *n*-grams)	Smit | mith | ith, | th,_ | h,_J | ,_Jo | _Joh | John | ohn_ | hn_D | n_D. (11 *n*-grams)	4	2x4 / (7+11) = **0.4444**

DICECOEFFICIENT( ) compared to ISFUZZYDUP( ) and LEVDIST( )

One of the key differences between the DICECOEFFICIENT( ) function and the ISFUZZYDUP( ) and LEVDIST( ) functions, which use Levenshtein distance, is that DICECOEFFICIENT( ) de-emphasizes or completely ignores the relative position of characters or character blocks in the two strings being compared. Relative position is significant in the functions based on Levenshtein distance.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Comparison values with transposition

If you are comparing strings such as addresses, in which entire elements might be transposed, DICECOEFFICIENT( ) might be a better choice. For example, the same address with the "Suite" element transposed is identified as highly similar by DICECOEFFICIENT( ), but highly different by LEVDIST( ):

Address pair	Dice's coefficient (default *n*-gram of 2)	Levenshtein distance
◦ 125 SW 39TH ST, Suite 100 ◦ Suite 100, 125 SW 39TH ST	0.9167	22 (the greater the Levenshtein distance, the more two strings differ)

Comparison values without transposition

If transposition is not an issue, LEVDIST( ) may give more useful results. For example, the same corporation name with different punctuation is identified as highly different by DICECOEFFICIENT( ), but highly similar by LEVDIST( ):

Corporation name pair	Dice's coefficient (default *n*-gram of 2)	Levenshtein distance
◦ AVS, Inc ◦ A.V.S. Inc	0.3750	3

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DIGIT( ) function

Returns the upper or lower digit of a specified Packed data type byte.

Syntax

```
DIGIT(byte_location, position)
```

Parameters

Name	Type	Description
byte_location	numeric	The byte position in the record.
position	numeric	The digit to return: ◦ specify 1 to return the upper half of the byte ◦ specify 2 to return the lower half of the byte

Output

Numeric.

Examples

Basic examples

A packed field with the value 123.45 (00 12 34 5C), containing two decimals, and starting in byte position 10, appears in the data record in the following format:

	Byte 10	Byte 11	Byte 12	Byte 13
UPPER(1)	0	1	3	5
LOWER(2)	0	2	4	C

Returns 3 (finds the digit that appears in the 12th position in the upper half of the byte):

```
DIGIT(12, 1)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns 4 (finds digit that appears in the 12th position in the lower half of the byte):

```
DIGIT(12, 2)
```

Remarks

How it works

DIGIT( ) separates individual halves of a byte, and returns the value of the byte specified in the position parameter as a digit between 0 and 15.

When to use DIGIT( )

Use DIGIT( ) to access individual half-bytes. This is required if you work with applications that use half-byte-aligned packed fields, such as Unisys applications.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DOW( ) function

Returns a numeric value (1 to 7) representing the day of the week for a specified date or datetime. Abbreviation for "Day of Week".

Syntax

```
DOW(date/datetime)
```

Parameters

Name	Type	Description
date/datetime	datetime	The field, expression, or literal value to extract the numeric day of the week from.

Output

Numeric.

Examples

Basic examples

Returns 4, because December 31, 2014 falls on a Wednesday, the 4th day of the week:

```
DOW(`20141231`)
```

```
DOW(`20141231 235959`)
```

Returns the numeric day of the week for each value in the **Invoice_date** field:

```
DOW(Invoice_date)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Advanced examples

Identifying transactions occurring on a weekend

Use the DOW( ) function to identify transactions that occur on a weekend. The filter below isolates dates in the **Trans_Date** field that occur on a Saturday or a Sunday:

```
SET FILTER TO DOW(Trans_Date) = 7 OR DOW(Trans_Date) = 1
```

Remarks

Parameter details

A field specified for *date/datetime* can use any date or datetime format, as long as the field definition correctly defines the format.

Specifying a literal date or datetime value

When specifying a literal date or datetime value for *date/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.
- **Time values** – you must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

Example formats	Example literal values
YYYYMMDD	`` `20141231` ``
YYMMDD	`` `141231` ``
YYYYMMDD hhmmss	`` `20141231 235959` ``
YYMMDDthhmm	`` `141231t2359` ``
YYYYMMDDThh	`` `20141231T23` ``
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`` `20141231 235959-0500` ``

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Example formats	Example literal values
YYMMDD hhmm+/-hh (UTC offset)	`141231 2359+01`
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Related functions

If you need to return:

- the name of the day of the week, use CDOW( ) instead of DOW( )
- the day of the month as a number (1 to 31), use DAY( ) instead of DOW( )

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DTOU( ) function

Converts an Analytics date value to a Unicode string in the specified language and locale format. Abbreviation for "Date to Unicode".

> **Note**
>
> This function is specific to the Unicode edition of Analytics. It is not a supported function in the non-Unicode edition.

Syntax

```
DTOU(<date> <,locale> <,style>)
```

Parameters

Name	Type	Description
date optional	datetime	The field, expression, or literal value to convert to a Unicode string. If omitted, the current operating system date is used. The *date* can contain a datetime value, but the time portion of the value is ignored. Standalone time values are not supported. You can specify a field or a literal date value: ◦ **Field** - can use any date format, as long as the field definition correctly defines the format ◦ **Literal** - must use one of the YYYYMMDD or YYMMDD formats, for example `` `20141231` `` The minimum supported *date* value is 31 December 1969.
locale optional	character	The locale code that specifies the language of the output string, and optionally the version of the language associated with a particular country or region. For example, `"zh"` specifies Chinese, and `"pt_BR"` specifies Brazilian Portuguese. If omitted, the default locale for your computer is used. If a language is specified, but no country is specified, the default country for the language is used. You cannot specify *locale* if you have not specified *date*. For information about locale codes, see www.unicode.org.
style optional	numeric	The date format style to use for the Unicode string. The format style matches the standard for the locale you specify:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Type	Description
		◦ 0 - full specification format, such as "Sunday, September 18, 2016" ◦ 1 - long format, such as "September 18, 2016" ◦ 2 - medium format, such as "Sep 18, 2016" ◦ 3 - short, numeric format such as "9/18/16" If omitted, the default value of 2 is used. You cannot specify *style* if you have not specified *date* and *locale*.

Output

Character.

Examples

Basic examples

Literal input values

Returns "31 de dezembro de 2014":

```
DTOU(`20141231`, "pt_BR", 1)
```

Returns "31 grudnia 2014":

```
DTOU(`20141231`, "pl", 1)
```

Field input values

Returns each numeric date in the **Invoice_date** field as a Unicode string:

```
DTOU(Invoice_date, "zh", 1)
```

Output uses full date style

Returns "星期三, 2014 十二月 31" (no region identifier specified):

```
DTOU(`20141231`, "zh", 0)
```

Returns "2014年12月31日 星期三" (region identifier specified):

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DTOU(`20141231`, "zh_CN", 0)
```

Output uses long date style

Returns "2014 十二月 31" (no region identifier specified):

```
DTOU(`20141231`, "zh", 1)
```

Returns "2014年12月31日" (region identifier specified):

```
DTOU(`20141231`, "zh_CN", 1)
```

Remarks

Related functions

DTOU( ) is the inverse of the UTOD( ) function, which converts a Unicode string to a date.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EBCDIC( ) function

Returns a string that has been converted to EBCDIC character encoding.

Syntax

```
EBCDIC(string)
```

Parameters

Name	Type	Description
string	character	The value to convert to EBCDIC.

Output

Character.

Examples

Basic examples

Returns "ñòó@Æ'...@â£K":

```
EBCDIC("123 Fake St.")
```

Advanced examples

Creating an EBCDIC-encoded field to export

To create a field containing the EBCDIC encoded value of a **Name** field for export to an application that requires EBCDIC encoding, specify the following:

```
DEFINE FIELD Name_Exp COMPUTED EBCDIC(Name)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

When to use EBCDIC( )

Use this function to convert data to the Extended Binary Coded Decimal Interchange Code (EBCDIC) character encoding. EBCDIC character encoding is used primarily on IBM mainframe operating systems, such as z/OS.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EFFECTIVE( ) function

Returns the effective annual interest rate on a loan.

Syntax

```
EFFECTIVE(nominal_rate, periods)
```

Parameters

Name	Type	Description
nominal_rate	numeric	The nominal annual interest rate.
periods	numeric	The number of compounding periods per year. **Note** Specify an integer. If you specified a decimal portion, it is truncated.

Output

Numeric. The rate is calculated to eight decimals places.

Examples

Basic examples

Returns 0.19561817 (19.56%), the effective annual rate of interest on the unpaid balance of a credit card that charges 18% per annum, compounded monthly:

```
EFFECTIVE(0.18, 12)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

What is the effective annual interest rate?

The effective annual interest rate on a loan is the actual annual rate of interest paid, taking into account interest on the remaining balance, compounded monthly or daily.

Related functions

The NOMINAL( ) function is the inverse of the EFFECTIVE( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EOMONTH( ) function

Returns the date of the last day of the month that is the specified number of months before or after a specified date.

Syntax

```
EOMONTH(<date/datetime> <,months>)
```

Parameters

Name	Type	Description
date/datetime optional	datetime	The field, expression, or literal value from which to calculate the end-of-month date. If omitted, the end-of-month date is calculated from the current operating system date. **Note** You can specify a datetime value for *date/datetime* but the time portion of the value is ignored.
months optional	numeric	The number of months before or after *date/datetime*. If omitted, the default of 0 (zero) is used. You cannot specify *months* if you have omitted *date/datetime*.

Output

Datetime. The date value is output using the current Analytics date display format.

Examples

Basic examples

No input

Returns the last day of the month for the current operating system date:

```
EOMONTH()
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Literal input values

Returns `20140131` displayed as 31 Jan 2014 assuming a current Analytics date display format of DD MMM YYYY:

```
EOMONTH(`20140115`)
```

Returns `20140430` displayed as 30 Apr 2014 assuming a current Analytics date display format of DD MMM YYYY:

```
EOMONTH(`20140115`, 3)
```

Returns `20131031` displayed as 31 Oct 2013 assuming a current Analytics date display format of DD MMM YYYY:

```
EOMONTH(`20140115`, -3)
```

Field input values

Returns the last day of the month that falls three months after each date in the **Invoice_date** field:

```
EOMONTH(Invoice_date, 3)
```

Returns the last day of the month that falls three months after each date in the **Invoice_date** field plus a grace period of 15 days:

```
EOMONTH(Invoice_date + 15, 3)
```

Returns the first day of the month in which the invoice date falls:

```
EOMONTH(Invoice_date, -1) + 1
```

Remarks

Datetime formats

A field specified for *date/datetime* can use any date or datetime format, as long as the field definition correctly defines the format.

A literal date value must use one of the following formats:

- YYYYMMDD
- YYMMDD

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You must enclose literal date values in backquotes. For example: `20141231`

How the *months* value works

- **Positive value** – the output date is more recent than the specified *date/datetime*
- **Negative value** – the output date is prior to the specified *date/datetime*
- **Value omitted, or '0' (zero)** – the output date is the last day of the month in which the *date/datetime* occurs

Return the date of the first day of a month

Add 1 day to the result of the EOMONTH( ) function to return the date of the first day of a month.

Returns `20140201` displayed as 01 Feb 2014 assuming a current Analytics date display format of DD MMM YYYY:

```
EOMONTH(`20140115`) + 1
```

Related functions

Use the GOMONTH( ) function if you want to return the exact date, rather than the date of the last day of the month, that is the specified number of months before or after a specified date.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EXCLUDE( ) function

Returns a string that excludes the specified characters.

Syntax

```
EXCLUDE(string, characters_to_exclude)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value from which to exclude characters.
characters_to_exclude	character	The list of characters to exclude. If you specify double quotation marks in *characters_to_exclude*, you must enclose the list of characters in single quotation marks. For example: `"'-/'`

Output

Character.

Examples

Basic examples

Returns " Alberni Street", which is the input string with all numbers excluded:

```
EXCLUDE("1550 Alberni Street", "0123456789")
```

Returns all the values in the **Product_Number** field with the forward slash and number sign excluded:

```
EXCLUDE(Product_Number, "/#")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

How it works

The EXCLUDE( ) function compares each character in *string* with the characters listed in *characters_to_exclude*. If a match occurs, the character is excluded from the output string.

For example, the output for `EXCLUDE("123-45-4536", "-")` is "123454536".

No matching characters

If there are no matches between *string* and *characters_to_exclude*, then *string* and the output of the function are the same.

For example, the output for `EXCLUDE("ABC", "D")` is "ABC".

Case sensitivity

The EXCLUDE( ) function is case-sensitive. If you specify "ID" in *characters_to_exclude*, these characters are not excluded from "id#94022". If there is a chance the case may be mixed, use the UPPER( ) function to convert *string* to uppercase.

For example:

```
EXCLUDE(UPPER("id#94022"), "ID#")
```

Usage tip

Use EXCLUDE( ) if the set of characters you want to exclude is small, and the set you want to include is large.

Excluding both single and double quotation marks

Quotation marks are used as string delimiters, therefore to exclude both single and double quotation marks you must nest EXCLUDE( ) so that there is a single function for each type of quotation mark:

```
EXCLUDE(EXCLUDE(field_to_process, """), "")
```

Related functions

The EXCLUDE( ) function is the opposite of the INCLUDE( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

EXP( ) function

Returns the exponential value (base 10) of a numeric expression with a specified number of decimal places.

Syntax

```
EXP(number, decimals)
```

Parameters

Name	Type	Description
number	numeric	The numeric field, expression, or value to return the exponential value of.
decimals	numeric	The number of decimals to include in the return value.

Output

Numeric.

Examples

Basic examples

Returns 1000.00:

```
EXP(3, 2)
```

Returns 72443.596007:

```
EXP(4.86, 6)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Advanced examples

Finding the cube root

Creates a field that is the cube root of the field X to two decimal places:

```
DEFINE FIELD cube_root COMPUTED EXP(LOG(X, 6) / 3, 2)
```

> Tip
> You can determine the *n*th root by dividing the log of the value by *n* and taking the exponential of the result.

Remarks

How it works

This function returns the exponential value (base 10) of a numeric expression, which is defined as 10 raised to the *n*th power. For example, the exponential value of 3 is 10^3, or 1000.

When to use EXP( )

Use EXP( ) for financial applications requiring complex mathematical calculations. EXP( ) performs the same operation as the exponentiation operator (^), but can be useful in applications that also use the LOG( ) function.

Related functions

The inverse of an exponent is its logarithm, so EXP( ) is the opposite of the LOG( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FILESIZE( ) function

Returns the size of the specified file in bytes or -1 if the file does not exist.

Syntax

```
FILESIZE(filename)
```

Parameters

Name	Type	Description
filename	character	The name of the file. If the file is in the same folder as the Analytics project, you do not need to specify the file path. For files in other folders, specify either a relative path or an absolute path. For example: ◦ "results\test_output.fil" ◦ "c:\results\test_output.fil" **Note** You need to specify the physical data file name (.fil) for Analytics tables, not the table name.

Output

Numeric.

Examples

Basic examples

Returns 14744:

```
FILESIZE("Inventory.fil")
```

If the file you are checking is not in the same folder as the Analytics project, you must specify either the relative path or absolute path to the file.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns 6018:

```
FILESIZE("C:\ACL Data\Sample Data Files\Backup\Ap_Trans.fil")
```

Advanced examples

Executing a script if a file does not exist

Only executes the script **import_data** if the file `Metaphor_Inventory_2002.fil` does not exist:

```
DO SCRIPT import_data IF FILESIZE("Metaphor_Inventory_2002.fil") = -1
```

Recording a file's size in the Analytics command log

Use the CALCULATE command to record the size of `Metaphor_Inventory_2002.fil` in the Analytics command log:

```
CALCULATE FILESIZE("Metaphor_Inventory_2002.fil")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FIND( ) function

Returns a logical value indicating whether the specified string is present in a particular field, or anywhere in an entire record.

> **Note**
>
> The FIND( ) function and the "FIND command" on page 210 are two separate Analytics features with significant differences.

Syntax

```
FIND(string <,field_to_search_in>)
```

Parameters

Name	Type	Description
string	character	The character string to search for. This search is not case-sensitive.
field_to_search_in optional	character	The field, or variable, to search in. If omitted, the entire record is searched, including any undefined portion of the record.

Output

Logical. Returns **T** (true) if the specified *string* value is found, and **F** (false) otherwise.

Examples

Basic examples

Searching an entire record

Returns T for all records that contain the string "New York" in any field, across any field boundaries, and in any undefined portion of the record. Returns F otherwise:

```
FIND("New York")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Searching a single field

Returns T for all records that contain the string "New York" in the **City** field. Returns F otherwise.

```
FIND("New York", City)
```

Returns T for all records that contain the string "Ne" in the **City** field. Returns F otherwise:

```
FIND("Ne", City)
```

Returns T for all records that contain the string "New York" preceded by one or more spaces in the **City** field. Returns F otherwise:

```
FIND(" New York", City)
```

Returns T for all records that have a value in the **Description** field that matches, or contains, the value in the *v_search_term* variable. Returns F otherwise:

```
FIND(v_search_term, Description)
```

Searching multiple fields

Returns T for all records that contain the string "New York" in either the **City** or the **City_2** fields. Returns F otherwise:

```
FIND("New York", City+City_2)
```

Returns T for all records that contain the string "New York" in either the **City** or the **City_2** fields. Returns F otherwise:

```
FIND("New York", City) OR FIND("New York", City_2)
```

Combining with other functions

Returns T for all records that have a value in the **Last_Name_2** field that matches, or contains, the trimmed value from the **Last_Name** field. Returns F otherwise:

```
FIND(ALLTRIM(Last_Name), Last_Name_2)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

When to use FIND( )

Use the FIND( ) function to test for the presence of the specified *string* in a field, two or more fields, or an entire record.

How matching works

The *string* value can be exactly matched or it can be contained within a longer string. Leading spaces in fields do not affect the search unless you include one or more leading spaces in the *string* value.

Search an entire record

If the optional *field_to_search_in* is not specified, the entire record is searched, including any undefined portion of the record. Field boundaries are ignored when the entire record is searched, and trailing spaces in fields are treated as characters.

> **Note**
>
> When you search an entire record, the physical record is searched. Any computed fields or related fields are not searched.

Search a subset of fields

You can concatenate two or more fields in the *field_to_search_in* if you want to search in a subset of the fields in a table. For example, to search both the **City** and **City_2** fields for the string "New York":

```
FIND("New York", City+City_2)
```

The concatenated fields are treated like a single field that includes leading and trailing spaces from the individual fields, unless you use the ALLTRIM( ) function to remove spaces.

You can also build an expression that searches each field individually:

```
FIND("New York", City) OR FIND("New York", City_2)
```

If *string* includes a leading space, search results from the two approaches can differ.

Case sensitivity and Exact Character Comparisons

The FIND( ) function is not case-sensitive, and finds both ASCII and EBCDIC characters. The function is not affected by the **Exact Character Comparisons** option (SET EXACT ON/OFF).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Search in a computed field

To search in a computed field you must specify the name of the field in *field_to_search_in*. For example, if **Vendor_City** is a computed field that isolates the city in an address:

```
FIND("New York", Vendor_City)
```

Search in a related field

To search in a related field you must specify the fully qualified name of the field (that is, *table.field name*) in the *field_to_search_in* value:

```
FIND("New York", Vendor.Vendor_City)
```

Search datetime or numeric data

It is possible to use the FIND( ) function to search datetime or numeric data at the record level. Specifying the *field_to_search_in* is not supported for datetime or numeric searching.

The numeric or datetime *string* must be enclosed in quotation marks, and needs to exactly match the source data formatting rather than the formatting in the view.

Using the FIND( ) function to search datetime or numeric data in computed or related fields is not supported.

> **Note**
>
> Using the FIND( ) function to search datetime or numeric data is not recommended because it can be difficult to do successfully.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FINDMULTI( ) function

Returns a logical value indicating whether any string in a set of one or more specified strings is present in a particular field, or anywhere in an entire record.

Syntax

```
FINDMULTI({search_in|RECORD}, string_1 <,...n>)
```

Parameters

Name	Type	Description
search_in | RECORD	character	The field, or variable, to search in. Specify the keyword RECORD to search the entire record, including any undefined portion of the record. You can also specify a list of fields by concatenating field names: `Field_1+Field_2+Field_3`
string_1 <,...n>	character	One or more character strings to search for. Separate multiple search strings with commas: `FINDMULTI(RECORD, "Joa", "Jim", "Joh")` The search is not case-sensitive.

Output

Logical. Returns **T** (true) if any of the specified *string* values are found, and **F** (false) otherwise.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Searching an entire record

Returns T for all records that contain "New York" or "Chicago" in any field, across any field boundaries, and in any undefined portion of the record. Returns F otherwise:

```
FINDMULTI(RECORD, "New York", "Chicago")
```

Searching a single field

Returns T for all records that contain "New York" or "Chicago" in the **City** field. Returns F otherwise:

```
FINDMULTI(City, "New York", "Chicago")
```

Returns T for all records that contain the string "Ne" or "Chi" in the **City** field. Returns F otherwise:

```
FINDMULTI(City, "Ne", "Chi")
```

Returns T for all records that contain "New York" or "Chicago" preceded by one or more spaces in the **City** field. Returns F otherwise:

```
FINDMULTI(City, " New York", " Chicago")
```

Returns T for all records that have a value in the **Description** field that matches, or contains, any of the values in the *v_search_term* variables . Returns F otherwise:

```
FINDMULTI(Description, v_search_term_1, v_search_term_2, v_search_term_3)
```

Searching multiple fields

Returns T for all records that contain the string "New York" or "Chicago" in either the **City** or the **City_2** fields. Returns F otherwise:

```
FINDMULTI(City+City_2, "New York", "Chicago")
```

Returns T for all records that contain the string "New York" or "Chicago" in either the **City** or the **City_2** fields. Returns F otherwise:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
FINDMULTI(City, "New York", "Chicago") OR FINDMULTI(City_2, "New York", "Chicago")
```

Combining with other functions

Returns T for all records that have a value in the **Last_Name_1** field that matches, or contains, the trimmed value from the **Last_Name_2** or **Last_Name_3** fields. Returns F otherwise:

```
FINDMULTI(Last_Name_1, ALLTRIM(Last_Name_2), ALLTRIM(Last_Name_3))
```

Remarks

When to use FINDMULTI( )

Use the FINDMULTI( ) function to test for the presence of any of the specified strings in a field, two or more fields, or an entire record.

How matching works

The *string* value can be exactly matched or it can be contained within a longer string. Leading spaces in fields do not affect the search unless you include one or more leading spaces in the *string* value.

Search an entire record

If you specify RECORD instead of a *search_in* field, the entire record is searched, including any undefined portion of the record. Field boundaries are ignored when the entire record is searched, and trailing spaces in fields are treated as characters.

> **Note**
>
> When you search an entire record, the physical record is searched. Any computed fields or related fields are not searched.

Search a subset of fields

You can concatenate two or more fields in the *search_in* parameter if you want to search in a subset of the fields in a table. For example, to search both the **City** and the **City_2** fields for the strings "New York" or "Chicago":

```
FINDMULTI(City+City_2, "New York", "Chicago")
```

The concatenated fields are treated like a single field that includes leading and trailing spaces from the individual fields, unless you use the ALLTRIM( ) function to remove spaces.

You can also build an expression that searches each field individually:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
FINDMULTI(City, "New York", "Chicago") OR FINDMULTI(City_2, "New York", "Chicago")
```

If a *string* value includes a leading space, search results from the two approaches can differ.

Case sensitivity and Exact Character Comparisons

The FINDMULTI( ) function is not case-sensitive, and finds both ASCII and EBCDIC characters. The function is not affected by the **Exact Character Comparisons** option (SET EXACT ON/OFF).

Search in a computed field

To search in a computed field you must specify the name of the field in *search_in*. For example, if **Vendor_City** is a computed field that isolates the city in an address:

```
FINDMULTI(Vendor_City, "New York", "Chicago")
```

Search in a related field

To search in a related field you must specify the fully qualified name of the field (that is, *table.field name*) in the *search_in* value:

```
FINDMULTI(Vendor.Vendor_City, "New York", "Chicago")
```

Search datetime or numeric data

It is possible to use the FINDMULTI( ) function to search datetime or numeric data at the record level, when specifying RECORD. Specifying a *search_in* field is not supported for datetime or numeric searching.

The numeric or datetime *string* values must be enclosed in quotation marks, and need to exactly match the source data formatting rather than the formatting in the view.

Using the FINDMULTI( ) function to search datetime or numeric data in computed or related fields is not supported.

> **Note**
>
> Using the FINDMULTI( ) function to search datetime or numeric data is not recommended because it can be difficult to do successfully.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FREQUENCY( ) function

Returns the expected Benford frequency for sequential leading positive numeric digits to a precision of eight decimal places.

Syntax

```
FREQUENCY(digit_string)
```

Parameters

Name	Type	Description
digit_string	character	A character string containing digits (0-9) to identify the frequency for. *digit_string* must be a positive number, and leading zeros are ignored.

Output

Numeric.

Examples

Basic examples

Returns 0.00998422:

```
FREQUENCY("43")
```

Returns 0.00000000:

```
FREQUENCY("87654321")
```

> **Note**
>
> The result is 0.00000000495, but because Analytics computes to a precision of eight decimal places, a zero value is returned.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

How it works

FREQUENCY( ) returns the expected Benford frequency for sequential leading positive numeric digits to a precision of eight digits. It lets you perform limited Benford tests for specific situations.

Using this function for specific digit combinations

You can use this function instead of the BENFORD command if you want to focus on specific digit combinations. For example, when auditing insurance claims that have approval limits at specified claim amounts, you could use the FREQUENCY( ) function to investigate amounts just under an approval threshold.

To investigate claims valued close to an approval limit of $5,000, you could select the range from $4,900 through $4,999. First, count the total number of records, then use a filter to count the records for which LEADING( ) returns 49, and compare the ratio of the two counts to the value you get for FREQUENCY ("49").

This is faster than running a complete analysis on a table of a million records, and it does not generate a large table or lengthy entries in the command log.

Specifying strings longer than six digits

Specifying strings longer than six digits can result in zero values. Calculations for strings longer than six digits may require greater precision than Analytics's limit of eight decimal places.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FTYPE( ) function

Returns a character identifying the data category of a field or variable, or the type of an Analytics project item.

Syntax

```
FTYPE(field_name_string)
```

Parameters

Name	Type	Description
field_name_string	character	A field name, variable name, or Analytics project item name. Enclose *field_name_string* in quotation marks: `FTYPE("Amount")`

Output

Character. This function returns one of the following characters, which indicates the field, variable, or Analytics project item type:

- **"C"** - Character field
- **"N"** - Numeric field
- **"D"** - Datetime field
- **"L"** - Logical field
- **"c"** - Character variable
- **"n"** - Numeric variable
- **"d"** - Datetime variable
- **"l"** - Logical variable
- **"b"** - Analytics script
- **"y"** - Analytics table layout
- **"w"** - Analytics workspace
- **"i"** - Analytics index
- **"r"** - Analytics report
- **"a"** - Analytics log file
- **"U"** - Undefined

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

The following example assigns a value of 4 to the *num* variable and then checks the type.

Returns "n":

```
ASSIGN num = 4
FTYPE("num")
```

Advanced examples

Testing for the data type of a field

You have a script or analytic that requires a numeric **Amount** field, and you need to test that the field is the correct type before running the script.

The following command only runs Script_1 if **Amount** is a numeric field:

```
OPEN Invoices
DO Script_1 IF FTYPE("Amount") = "N"
```

Testing if a table or Analytics project item exists

The following command only runs Script_1 if a table named Invoices is present in the project:

```
DO Script_1 IF FTYPE("Invoices") <> "U"
```

Testing the runtime environment

You can use FTYPE to determine if an analytic is running in Analytics, or on Analytics Exchange or in the Analysis App window.

If an analytic is running on Analytics Exchange, or in the Analysis App window, 'ax_main' is equal to 'b':

```
IF FTYPE('ax_main') = 'b' v_running_in_ax_or_analysis_app = T
```

If an analytic is running in Analytics, 'ax_main' is not equal to 'b':

```
IF FTYPE('ax_main') <> 'b' v_running_in_ax_or_analysis_app = F
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

The ability to detect the runtime environment allows you to design a single script that executes different blocks of codes depending on which application it is running in.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FVANNUITY( ) function

Returns the future value of a series of payments calculated using a constant interest rate. Future value is the sum of the payments plus the accumulated compound interest.

Syntax

FVANNUITY(*rate*, *periods*, *payment* <,*type*>)

Parameters

Name	Type	Description
rate	numeric	The interest rate per period.
periods	numeric	The total number of payment periods.
payment	numeric	The payment per period. The payment amount must remain constant over the term of the annuity.
type optional	numeric	The timing of payments: ◦ 0 - payment at the end of a period ◦ 1 - payment at the beginning of a period If omitted, the default value of 0 is used.

> **Note**
>
> You must use consistent time periods when specifying *rate*, *periods*, and *payment* to ensure that you are specifying interest rate **per period**.
>
> For example:
>
> - for a monthly *payment* on a two-year loan or investment with interest of 5% per annum, specify 0.05/12 for *rate* and 2 * 12 for *periods*
> - for an annual *payment* on the same loan or investment, specify 0.05 for *rate* and 2 for *periods*

Output

Numeric. The result is calculated to two decimal places.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Monthly payments

Returns 27243.20, the future value of $1,000 paid at the beginning of each month for 2 years at 1% per month, compounded monthly:

```
FVANNUITY(0.01, 2*12, 1000, 1)
```

Returns 12809.33, the future value of the same annuity after the first year:

```
FVANNUITY(0.01, 12, 1000, 1)
```

Annual payments

Returns 25440.00, the future value of $12,000 paid at the end of each year for 2 years at 12% per annum, compounded annually:

```
FVANNUITY(0.12, 2, 12000, 0)
```

Advanced examples

Annuity calculations

Annuity calculations involve four variables:

- **present value, or future value** – $21,243.39 and $ 26,973.46 in the examples below
- **payment amount per period** – $1,000.00 in the examples below
- **interest rate per period** – 1% per month in the examples below
- **number of periods** – 24 months in the examples below

If you know the value of three of the variables, you can use an Analytics function to calculate the fourth.

I want to find:	Analytics function to use:
Present value	PVANNUITY( ) Returns 21243.39: `PVANNUITY(0.01, 24, 1000)`
Future value	FVANNUITY( )

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

I want to find:	Analytics function to use:
	Returns 26973.46: FVANNUITY(0.01, 24, 1000)
Payment amount per period	PMT( ) Returns 1000: PMT(0.01, 24, 21243.39)
Interest rate per period	RATE( ) Returns 0.00999999 (1%): RATE(24, 1000, 21243.39)
Number of periods	NPER( ) Returns 24.00: NPER(0.01, 1000, 21243.39)

Annuity formulas

The formula for calculating the **present value** of an ordinary annuity (payment at the end of a period):

$$PV_A = Pmt\left[\frac{1-\frac{1}{(1+i)^N}}{i}\right]$$

$$21243.39 = 1000\left[\frac{1-\frac{1}{(1+0.01)^{24}}}{0.01}\right]$$

The formula for calculating the **future value** of an ordinary annuity (payment at the end of a period):

$$FV_A = Pmt\left[\frac{(1+i)^N-1}{i}\right]$$

$$26973.46 = 1000\left[\frac{(1+0.01)^{24}-1}{0.01}\right]$$

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Related functions

The PVANNUITY( ) function is the inverse of the FVANNUITY( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FVLUMPSUM( ) function

Returns the future value of a current lump sum calculated using a constant interest rate.

Syntax

FVLUMPSUM(*rate*, *periods*, *amount*)

Parameters

Name	Type	Description
rate	numeric	The interest rate per period.
periods	numeric	The total number of periods.
amount	numeric	The investment made at the start of the first period.

> **Note**
>
> You must use consistent time periods when specifying *rate* and *periods* to ensure that you are specifying interest rate **per period**.
>
> For example:
>
> - for monthly payments on a two-year loan or investment with interest of 5% per annum, specify 0.05/12 for *rate* and 2 * 12 for *periods*
> - for annual payments on the same loan or investment, specify 0.05 for *rate* and 2 for *periods*

Output

Numeric. The result is calculated to two decimal places.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Interest compounded monthly

Returns 1269.73, the future value of a lump sum of $1,000 invested for 2 years at 1% per month, compounded monthly:

```
FVLUMPSUM(0.01, 2*12, 1000)
```

Returns 1126.83, the future value of the same investment after the first year:

```
FVLUMPSUM(0.01, 12, 1000)
```

Returns 27243.20, the future value of $21,455.82 invested for 2 years at 1% per month, compounded monthly:

```
FVLUMPSUM(0.01, 2*12, 21455.82)
```

Interest compounded semi-annually

Returns 1262.48, the future value of a lump sum of $1,000 invested for 2 years at 12% per annum, compounded semi-annually:

```
FVLUMPSUM(0.12/2, 2*2, 1000)
```

Interest compounded annually

Returns 1254.40, the future value of a lump sum of $1,000 invested for 2 years at 12% per annum, compounded annually:

```
FVLUMPSUM(0.12, 2, 1000)
```

Remarks

What is future value?

The future value of an invested lump sum is the initial investment principal plus the accumulated compound interest.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Related functions

The PVLUMPSUM( ) function is the inverse of the FVLUMPSUM( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FVSCHEDULE( ) function

Returns the future value of a current lump sum calculated using a series of interest rates.

Syntax

```
FVSCHEDULE(principal, rate1 <,rate2...>)
```

Parameters

Name	Type	Description
principal	numeric	The amount of the initial investment.
rate1, rate2...	numeric	A series of interest rates for equal-length periods. **Note** The periods can represent months or years, or some other time period, as long as the type of time period is consistent. You must specify the interest rates **per period**. So, if one of the interest rates is 5% per annum and the periods are months, specify 0.05/12.

Output

Numeric. The result is calculated to two decimal places.

Examples

Basic examples

Returns 1282.93, the future value of a lump sum of $1000 invested for 3 years at 10% for the first year, 9% for the second year, and 7% for the third year, compounded annually:

```
FVSCHEDULE(1000, 0.1, 0.09, 0.07)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

The future value of an invested lump sum is the initial investment principal plus the accumulated compound interest.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

GETOPTIONS( ) function

Returns the current setting for the specified Analytics option (**Options** dialog box setting).

Syntax

```
GETOPTIONS(option)
```

Parameters

Name	Type	Description
option	character	The Analytics option to return a setting for. The name of the option must be specified exactly as it appears in the list below, and it must be enclosed in quotation marks: ◦ **separators** - returns the current settings for the three Analytics separator characters, in the following order: • decimal place symbol • thousands separator • list separator **Note** Currently, "separators" is the only *option* that can be specified for the GETOPTIONS( ) function.

Output

Character.

Examples

Basic examples

Returns the current settings for the three Analytics separator characters. For example, ".,,":

```
GETOPTIONS("separators")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Advanced examples

Using GETOPTIONS( ) in a script

If a script needs to change one or more of the Analytics separator characters, the GETOPTIONS( ) function provides a method for discovering the current settings. The current settings can be stored in a variable and then reinstated at the end of the script.

```
ASSIGN v_SeparatorsSetting = GETOPTIONS("separators")
SET SEPARATORS ",.;"
<script content>
SET SEPARATORS "%v_SeparatorsSetting%"
```

Remarks

The three Analytics separator characters are specified in the following options in the **Options** dialog box:

- **Decimal Place Symbol**
- **Thousands Separator**
- **List Separator**

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

GOMONTH( ) function

Returns the date that is the specified number of months before or after a specified date.

Syntax

```
GOMONTH(date/datetime, months)
```

Parameters

Name	Type	Description
date/datetime	datetime	The field, expression, or literal value from which to calculate the output date.
months	numeric	The number of months before or after *date/datetime*. **Note** You can specify a datetime value for *date/datetime* but the time portion of the value is ignored.

Output

Datetime. The date value is output using the current Analytics date display format.

Examples

Basic examples

Literal input values

Returns `20140415` displayed as 15 Apr 2014 assuming a current Analytics date display format of DD MMM YYYY:

```
GOMONTH(`20140115`, 3)
```

Returns `20131015` displayed as 15 Oct 2013 assuming a current Analytics date display format of DD MMM YYYY:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
GOMONTH(`20140115`, -3)
```

Returns `20140430` displayed as 30 Apr 2014 assuming a current Analytics date display format of DD MMM YYYY (date rounding prevents returning 31 Apr 2014, which is an invalid date):

```
GOMONTH(`20140330`, 1)
```

```
GOMONTH(`20140331`, 1)
```

Returns `20140501` displayed as 01 May 2014 assuming a current Analytics date display format of DD MMM YYYY:

```
GOMONTH(`20140401`, 1)
```

Field input values

Returns the date three months after each date in the **Invoice_date** field:

```
GOMONTH(Invoice_date, 3)
```

Returns the date three months after each date in the **Invoice_date** field plus a grace period of 15 days:

```
GOMONTH(Invoice_date + 15, 3)
```

Remarks

Datetime formats

A field specified for *date/datetime* can use any date or datetime format, as long as the field definition correctly defines the format.

A literal date value must use one of the following formats:

- YYYYMMDD
- YYMMDD

You must enclose literal date values in backquotes. For example: `` `20141231` ``

How the *months* value works

- **Positive value** - the output date is more recent than the specified *date/datetime*
- **Negative value** - the output date is prior to the specified *date/datetime*
- **Value omitted, or '0' (zero)** - the output date is the same as the *date/datetime*

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Date rounding to avoid non-existent dates

If the combination of *date/datetime* and *months* would produce a non-existent date, the GOMONTH( ) function uses 'date rounding' to return the closest valid date within the same month.

Returns `20140430` (30 Apr 2014) because 31 Apr 2014 is an invalid date:

```
GOMONTH(`20140331`,1)
```

Related functions

Use the EOMONTH( ) function if you want to return the date of the last day of the month, rather than the exact date, that is the specified number of months before or after a specified date.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

HASH( ) function

Returns a salted cryptographic hash value based on the input value.

Syntax

```
HASH(field <,salt_value>)
```

Parameters

Name	Type	Description
field	character numeric datetime logical	The value to hash.
salt_value optional	character numeric	The salt value to use. You can specify a PASSWORD identifier number from 1 to 10, or a character string. If omitted, the Analytics default salt value is used. The salt value is limited to 128 characters, and is automatically truncated to 128 characters if you specify a longer salt value. For more information, see "The salt value" on page 604.

Output

Character.

Examples

Basic examples

With the Analytics default salt value

Returns "819A974BB91215D58E7753FD5A42226150100A0763087CA7DECD93F3C3090405":

```
HASH("555-44-3322")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns the hash value for each number in the **Credit_card_num** field:

```
HASH(Credit_card_num)
```

With a user-specified salt value

Returns "AD1E7D9B97B6F6B5345AB13471A74C31EBE6630CA2622BB7E8C280E9FBEE1F17":

```
HASH("555-44-3322", "my salt value 123")
```

Advanced examples

Ensuring hash values are identical

Use other functions in conjunction with HASH( ) to standardize clear text values that should produce identical hash values.

Consider the following set of examples. Note how the case of the clear text values completely alters the output hash value in the first two examples.

Returns "DF6789E1EC65055CD9CA17DD5B0BEA5892504DFE7661D258737AF7CB9DC46462":

```
HASH("John Smith")
```

Returns "3E12EABB5940B7A2AD90A6B0710237B935FAB68E629907927A65B3AA7BE6781D":

```
HASH("JOHN SMITH")
```

By using the UPPER( ) function to standardize case, an identical hash value results.

Returns "3E12EABB5940B7A2AD90A6B0710237B935FAB68E629907927A65B3AA7BE6781D":

```
HASH(UPPER("John Smith"))
```

Using HASH( ) to compare large blocks of text

Use HASH( ) to test if blocks of text in two comment fields are identical.

To perform this test, create two computed fields similar to the ones shown below, and then create a filter to find any text blocks that are not identical.

```
DEFINE FIELD Hash_1 COMPUTED HASH(Comment_Field_1)
DEFINE FIELD Hash_2 COMPUTED HASH(Comment_Field_2)
SET FILTER TO Hash_1 <> Hash_2
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

If the comment fields are in separate tables, create a computed HASH( ) field in each table and then use the computed fields as a common key field to do an unmatched join of the two tables. The records in the joined output table represent text blocks that are not identical.

Remarks

When to use HASH( )

Use the HASH( ) function to protect sensitive data, such as credit card numbers, salary information, or social security numbers.

How it works

HASH( ) provides one-way encoding. Data in clear text can be used to produce a hash value, however the hash value cannot subsequently be unencoded or decrypted.

A specific clear text value always produces the same hash value, so you can search a field of hashed credit card numbers for duplicates, or join two fields of hashed credit card numbers, and the results are the same as if you had performed the operation on the equivalent clear text fields.

Protecting sensitive data

To avoid storing sensitive data on a server, you can create a computed field locally using the HASH( ) function, and then create a new table by extracting the hashed field and any other required fields, while excluding the clear text field. You can use the new table on the server for your analysis, and once you have the results, refer back to the original table if you need to see the clear text version of any of the hashed data.

If storing sensitive data locally, beyond initial use, is prohibited, you can delete the original table after you have created the new table with the hashed values, and refer to the original source system for the clear text values.

Clear text values must be exactly identical

In order to produce identical hash values, two clear text values must be exactly identical. For example, different hash values result from the same credit card number with or without hyphens, or the same name in title case or all upper case.

You may need to incorporate functions such as INCLUDE( ), EXCLUDE( ), or UPPER( ) in the HASH( ) function to standardize clear text values.

Leading and trailing blanks are automatically trimmed by the HASH( ) function, so there is no need to use the TRIM( ) or ALLTRIM( ) functions.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

What if leading or trailing blanks are meaningful?

If you have data in which leading or trailing blanks represent meaningful differences between values you need to replace the blanks with another character before hashing the values.

Replaces blanks in the field values with the underscore character (_) before hashing:

```
HASH(REPLACE(field_name, " ", "_"))
```

The cryptographic algorithm used by HASH( )

HASH( ) uses an SHA-2 cryptographic hash algorithm that produces a fixed-length hashed output of 64 bytes, regardless of the length of the input value. The clear text input value can be longer than 64 bytes.

The salt value

How it works

The protection offered by the HASH( ) function is strengthened by the automatic addition of a salt value prior to hashing. The salt value is an alphanumeric string that is concatenated with the source data value. The entire concatenated string is then used to produce the salted, hashed value. This approach makes the hashed values more resistant to decoding techniques.

Optionally specify your own salt value

A fixed, default salt value is automatically used unless you specify a salt value. You can use either of the following methods to specify a salt value:

- **Salt value as clear text string**

  Specify an alphanumeric string. For example:

  ```
  HASH(Credit_card_num, "my salt value")
  ```

- **Salt value as password**

  Use the PASSWORD command in conjunction with the HASH( ) function and specify a PASSWORD identifier number from 1 to 10. For example:

  ```
  PASSWORD 3 "Enter a salt value"
  EXTRACT FIELDS HASH(Credit_card_num, 3) TO "Protected_table"
  ```

  > **Note**
  >
  > The PASSWORD salt value must be entered before the field in the HASH ( ) function can be extracted.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

The benefit of using a PASSWORD identifier number with HASH( ) is that you do not have to expose a clear text salt value.

For more information, see "PASSWORD command" on page 360.

Password method guidelines

The password method is intended for use in scripts that prompt for the password at the beginning of the script, or prior to the HASH( ) function appearing in the script.

The password method is not suitable for use in computed fields because PASSWORD assignments are deleted when you close Analytics.

In addition, computed fields that use a password-based salt value are automatically removed from views when you reopen Analytics. This removal is necessary to avoid the recalculation of hash values using the default salt value. The recalculated values would differ from the original hash values calculated with a user-supplied salt value.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

HEX( ) function

Converts an ASCII string to a hexadecimal string.

Syntax

```
HEX(field)
```

Parameters

Name	Type	Description
field	character	The ASCII string to convert to a hexadecimal string.

Output

Character.

Examples

Basic examples

Returns "3132333435":

```
HEX("12345")
```

Returns the values in the **Count** field as hexadecimal strings:

```
HEX(Count)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

How it works

This function returns the hexadecimal string that is equivalent to the field value or expression you specify. You can use the function when you need to identify the exact contents of a field, including characters that cannot be displayed on screen, such as CR (carriage return), LF (line feed), and NUL (null).

Return value length

The return value is a string that is double the length of the *field* value. The digits 0 to 9 and the letters A to F (for the digits 10 to 15) represent the 16 hexadecimal values.

Use fields as input rather than expressions

In general, you should apply this function to fields rather than expressions because it displays a representation of the internal storage format of expressions, which is not meaningful in most instances.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

HOUR( ) function

Extracts the hour from a specified time or datetime and returns it as a numeric value using the 24-hour clock.

Syntax

```
HOUR(time/datetime)
```

Parameters

Name	Type	Description
time/datetime	datetime	The field, expression, or literal value to extract the hour from.

Output

Numeric.

Examples

Basic examples

Returns 23:

```
HOUR(`t235959`)
```

```
HOUR(`20141231 235959`)
```

Returns the hour for each value in the **Call_start_time** field:

```
HOUR(Call_start_time)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Parameter details

A field specified for *time/datetime* can use any time or datetime format, as long as the field definition correctly defines the format.

Specifying a literal time or datetime value

When specifying a literal time or datetime value for *time/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231 235959` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Time values** – you can use any of the time formats listed in the table below. You must use a separator before a standalone time value for the function to operate correctly. Valid separators are the letter 't', or the letter 'T'. You must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).
- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.

Example formats	Example literal values
thhmmss	`` `t235959` ``
Thhmm	`` `T2359` ``
YYYYMMDD hhmmss	`` `20141231 235959` ``
YYMMDDthhmm	`` `141231t2359` ``
YYYYMMDDThh	`` `20141231T23` ``
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`` `20141231 235959-0500` ``
YYMMDD hhmm+/-hh (UTC offset)	`` `141231 2359+01` ``
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

HTOU( ) function

Converts a hexadecimal string to a Unicode string. Abbreviation for "Hexadecimal to Unicode".

> **Note**
>
> This function is specific to the Unicode edition of Analytics. It is not a supported function in the non-Unicode edition.

Syntax

```
HTOU(hex_string)
```

Parameters

Name	Type	Description
hex_string	character	The hexadecimal string to convert to a Unicode string. The string can only contain hexadecimal values, for example "20AC".

Output

Character.

Examples

Basic examples

Returns "ABC123":

```
HTOU("004100420043003100320033")
```

Advanced examples

Adding a currency symbol to a value

You need to extract a monetary field to a new table. The field should display the numeric **Amount** field's

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

value and prepend it with a Euro currency symbol (€):

```
EXTRACT HTOU("20AC") + STRING(Amount, 10) AS "Currency_Amount" TO Display_Table
```

When the EXTRACT command runs, HTOU( ) returns the Euro symbol "€" and concatenates it with the **Amount** value that STRING( ) converts to a character. If the original value of **Amount** was 2000, then the value in **Currency_Amount** is "€2000".

Remarks

Related functions

HTOU( ) is the inverse of the DHEX( ) function, which converts a Unicode string to a hexadecimal string.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

INCLUDE( ) function

Returns a string that includes only the specified characters.

Syntax

```
INCLUDE(string, characters_to_include)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to restrict to included characters.
characters_to_include	character	The list of characters to include. If you specify double quotation marks in *characters_to_include*, you must enclose the list of characters in single quotation marks. For example: `'"-/'` **Note** If a character you specify to include does not appear in *string*, it is not included in the return value.

Output

Character.

Examples

Basic examples

Returns "123", which is the input string with only numbers included:

```
INCLUDE("123 Main St.", "0123456789")
```

Returns "1231234", which is the input string with only numbers included:

```
INCLUDE("123-123-4", "1243")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns "" (nothing), because the input string does not contain "D":

```
INCLUDE("ABC", "D")
```

Remarks

How it works

The INCLUDE( ) function compares each character in *string* with the characters listed in *characters_to_include*. If a match occurs, the character is included in the output string.

No matching characters

If there are no matches between *string* and *characters_to_include* the output of the function is blank.

Case sensitivity

The INCLUDE( ) function is case-sensitive. If you specify "ID" in *characters_to_include*, these characters are not included in "id#94022". If there is a chance the case may be mixed, use the UPPER( ) function to convert *string* to uppercase.

For example:

```
INCLUDE(UPPER("id#94022"), "ID0123456789")
```

Usage tip

Use INCLUDE( ) if the set of characters you want to include is small, and the set you want to exclude is large.

Related functions

The INCLUDE( ) function is the opposite of the EXCLUDE( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

INSERT( ) function

Returns the original string with specified text inserted at a specific byte location.

Syntax

```
INSERT(string, insert_text, location)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to insert the text into.
insert_text	character	The text to insert.
location	numeric	The character position at which to insert *insert_text* into the *string*.

Output

Character.

Examples

Basic examples

Returns "aXXXbcde":

```
INSERT("abcde", "XXX", 2)
```

Returns "XXXabcde":

```
INSERT("abcde", "XXX", 0)
```

Returns "abcdeXXX", with `"XXX"` inserted at byte position 6 instead of 8, because `"abcde"` is only 5 bytes long::

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
INSERT("abcde", "XXX", 8)
```

Remarks

How it works

The INSERT( ) function inserts specified characters or spaces into a character string, beginning at a specified position in the string.

When to use INSERT( )

Use INSERT( ) to normalize data for formatting, for duplicate matching, and for the JOIN and DEFINE RELATION commands, which require identical fields.

For example, part numbers in one file may be in the format "12345", and in another file, "12-345." In the first file, you can use INSERT( ) to insert a hyphen (-) in position 3.

Location guidelines

- If the *location* value is greater than the length of *string*, the *insert_text* value is inserted at the end of the string.
- If *location* is 0 or 1, *insert_text* is inserted at the beginning of the string.

Inserting double quotation marks

If you specify double quotation marks in *insert_text*, you must enclose them in single quotation marks.

For example: `""`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

INT( ) function

Returns the integer value of a numeric expression or field value.

Syntax

```
INT(number)
```

Parameters

Name	Type	Description
number	numeric	The field or numeric expression to convert to an integer. If the value specified includes decimals, the decimals are truncated without rounding.

Output

Numeric.

Examples

Basic examples

Returns 7:

```
INT(7.9)
```

Returns -7:

```
INT(-7.9)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

IPMT( ) function

Returns the interest paid on a loan for a single period.

Syntax

IPMT(*rate*, *specified_period*, *periods*, *amount* <,*type*>)

Parameters

Name	Type	Description
rate	numeric	The interest rate per period.
specified_period	numeric	The period for which you want to find the interest payment.
periods	numeric	The total number of payment periods.
amount	numeric	The principal amount of the loan.
type optional	numeric	The timing of payments: ◦ 0 - payment at the end of a period ◦ 1 - payment at the beginning of a period If omitted, the default value of 0 is used.

> **Note**
>
> You must use consistent time periods when specifying *rate* and *periods* to ensure that you are specifying interest rate **per period**.
>
> For example:
>
> - for monthly payments on a two-year loan or investment with interest of 5% per annum, specify 0.05/12 for *rate* and 2 * 12 for *periods*
> - for annual payments on the same loan or investment, specify 0.05 for *rate* and 2 for *periods*

Output

Numeric.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Returns 1489.58, the interest paid in the first month of a twenty-five year, $275,000 loan at 6.5% per annum, with payments due at the end of the month:

```
IPMT(0.065/12, 1, 12*25, 275000, 0)
```

Returns 10.00, the interest paid on the same loan in the last month of the loan:

```
IPMT(0.065/12, 300, 12*25, 275000, 0)
```

Remarks

Related functions

The PPMT( ) function is the complement of the IPMT( ) function.

The CUMIPMT( ) function calculates interest paid during a range of periods.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ISBLANK( ) function

Returns a logical value indicating whether the input value is blank.

Syntax

```
ISBLANK(string)
```

Parameters

Name	Type	Description
string	character	The value to test for blank data.

Output

Logical. Returns **T** (true) if the *string* parameter value is blank, and **F** (false) otherwise.

Examples

Basic examples

Returns F:

```
ISBLANK(" A")
```

Returns T:

```
ISBLANK(" ")
```

```
ISBLANK("")
```

Returns T for all values in the **Address** field that are blank, and F otherwise:

```
ISBLANK(Address)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

When to use ISBLANK( )

Use ISBLANK( ) during the data integrity phase of an analysis project to identify fields with missing data, which may indicate issues with the source data.

What is blank input?

For the function to evaluate to true, the input value must be one of the following:

- entirely blank (that is, contain only spaces)
- a zero-length string

The function only identifies true blanks in source data, not invalid characters that appear as blanks in a view.

Null characters

ISBLANK( ) may not return useful results when used with character fields that contain null characters. Analytics uses the null character to indicate the end of a string, and for this reason the ISBLANK( ) function will not read any character data that follows a null character, including blanks.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ISDEFINED( ) function

Returns **T** (true) if the specified field or variable is defined, and **F** (false) otherwise.

Syntax

```
ISDEFINED(string)
```

Parameters

Name	Type	Description
string	character	The name of the field or variable to check for the existence of. The value must be entered as a quoted string: `ISDEFINED("v_numeric_limit")`

Output

Logical.

Examples

Basic examples

Returns **T** if `v_numeric_limit` is defined as a variable or field, otherwise returns **F**:

```
ISDEFINED("v_numeric_limit")
```

Advanced examples

Using ISDEFINED( ) to test a field

The following example uses the ISDEFINED( ) function to test if the **Limit** field is defined in the table before extracting records based on the value in the field:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
OPEN Metaphor_Employees_US
IF ISDEFINED("Limit") EXTRACT RECORD IF Limit > 50000 TO "HighLimit.fil"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ISFUZZYDUP( ) function

Returns a logical value indicating whether a string is a fuzzy duplicate of a comparison string.

Syntax

ISFUZZYDUP(*string1*, *string2*, *levdist* <,*diffpct*>)

Parameters

Name	Type	Description
string1	character	The first string in the comparison.
string2	character	The second string in the comparison.
levdist	numeric	The maximum allowable Levenshtein distance between the two strings for them to be identified as fuzzy duplicates. The *levdist* value cannot be less than 1 or greater than 10. Increasing the *levdist* value increases the number of results by including values with a greater degree of fuzziness - that is, values that are more different from each another.
diffpct optional	numeric	The upper threshold for the 'difference percentage'. Difference percentage is explained in "How it works" on page 625. The *diffpct* value cannot be less than 1 or greater than 99. Increasing the *diffpct* value increases the number of results by including values with a greater proportion of difference relative to their length. If omitted, difference percentage is not considered during processing of the ISFUZZYDUP( ) function.

Output

Logical. Returns **T** (true) if *string* values are fuzzy duplicates, and **F** (false) otherwise.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Returns F, because two edits are required to transform "Smith" into "Smythe", but the *levdist* value is only 1:

```
ISFUZZYDUP("Smith","Smythe", 1, 99)
```

Returns T, because two edits are required to transform "Smith" into "Smythe", and the *levdist* value is 2:

```
ISFUZZYDUP("Smith","Smythe", 2, 99)
```

Returns T, because zero edits are required to transform "SMITH" into "smith", and the *levdist* value is 1 (the ISFUZZYDUP( ) function is not case-sensitive):

```
ISFUZZYDUP("SMITH","smith", 1, 99)
```

Returns a logical value (T or F) indicating whether individual values in the **Last_Name** field are fuzzy duplicates for the string "Smith":

```
ISFUZZYDUP(Last_Name,"Smith", 3, 99)
```

Advanced examples

Working with difference percentage

The difference percentage gives you a tool for reducing the number of false positives returned by ISFUZZYDUP( ).

No *diffpct* specified

Returns T, because five edits are required to transform "abc" into "Smith", and the *levdist* value is 5:

```
ISFUZZYDUP("abc", "Smith", 5)
```

***diffpct* specified**

Returns F, even though "abc" is within the specified Levenshtein distance of "Smith", because *5 edits/a string length of 3* results in a difference percentage of 167%, which exceeds the specified *diffpct* of 99%:

```
ISFUZZYDUP("abc", "Smith", 5, 99)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Difference percentage is fully explained in "How it works" below.

Isolating fuzzy duplicates for "Smith"

Create a filter that isolates all values in the **Last_Name** field that are fuzzy duplicates for "Smith":

```
SET FILTER TO ISFUZZYDUP(Last_Name, "Smith", 3, 99)
```

Changing the *levdist* or *diffpct* values allows you to adjust the amount of difference in the filtered values.

Remarks

When to use ISFUZZYDUP( )

Use the ISFUZZYDUP( ) function to find nearly identical values (fuzzy duplicates) or locate inconsistent spelling in manually entered data.

How it works

The ISFUZZYDUP( ) function calculates the Levenshtein distance between two strings, and calculates the difference percentage.

ISFUZZYDUP( ) evaluates to T (true) if:

- The Levenshtein distance is less than or equal to the *levdist* value.
- The difference percentage is less than or equal to the *diffpct* value (if specified).

Levenshtein distance

The Levenshtein distance is a value representing the minimum number of single character edits required to make one string identical to the other string.

For more information, see "LEVDIST( ) function" on page 633.

Difference percentage

The difference percentage is the percentage of the shorter of the two evaluated strings that is different.

The difference percentage is the result of the following internal Analytics calculation, which uses the Levenshtein distance between the two strings:

Levenshtein distance / number of characters in the shorter string × 100 = difference percentage

Using the optional difference percentage helps reduce the number of false positives returned by ISFUZZYDUP( ):

- The upper threshold for *diffpct* is 99%, which prevents the entire replacement of a string in order to make it identical.
- Strings that require a large number of edits in relation to their length are excluded.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Usage tips

- **Case-sensitivity** – The function is not case-sensitive, so "SMITH" is equivalent to "smith."
- **Trailing blanks** – The function automatically trims trailing blanks in fields, so there is no need to use the TRIM( ) function when specifying a field as a parameter.
- **Removing generic elements** – The OMIT( ) function can improve the effectiveness of the ISFUZZYDUP( ) function by removing generic elements such as "Corporation" or "Inc." from field values.

  Removal of generic elements focuses the ISFUZZYDUP( ) string comparison on just the portion of the strings where a meaningful difference may occur.

How the FUZZYDUP command and the ISFUZZYDUP( ) function differ

The FUZZYDUP command identifies all fuzzy duplicates in a field, organizes them in groups, and outputs non-exhaustive results.

The ISFUZZYDUP( ) function identifies an exhaustive list of fuzzy duplicates for a single character value.

The command and the function both identify exact duplicates. Unlike the command, you cannot exclude exact duplicates when using the function.

What exhaustive means

Exhaustive means that all values within the specified degree of difference of the test value are returned, regardless of their position in the test field relative to the test value.

The ISFUZZYDUP( ) function is useful if the non-exhaustive results produced by the FUZZYDUP command are not sufficient for the purposes of your analysis, and you need to directly scrutinize every fuzzy duplicate for a specific character value.

Related functions

- **LEVDIST( )** – provides an alternate method for comparing strings based on Levenshtein distance.

  Unlike ISFUZZYDUP( ), LEVDIST( ) is case-sensitive by default.
- **DICECOEFFICIENT( )** – de-emphasizes or completely ignores the relative position of characters or character blocks when comparing strings.
- **SOUNDSLIKE( ) and SOUNDEX( )** – compare strings based on a phonetic comparison (sound) rather than on an orthographic comparison (spelling).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

LAST( ) function

Returns a specified number of characters from the end of a string.

Syntax

```
LAST(string, length)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to return the characters from.
length	numeric	The number of characters to return.

Output

Character.

Examples

Basic examples

Returns "Savings":

```
LAST("Account Type: Savings", 7)
```

Returns "efghi":

```
LAST("abcdefghi", 5)
```

Returns "fghi ":

```
LAST("abcdefghi ", 5)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns " abc", because the *string* value is shorter than the specified length of 6, so leading spaces are added to the output:

```
LAST("abc", 6)
```

Remarks

Blank results caused by trailing spaces

Trailing spaces in *string* can cause the results produced by the LAST( ) function to be blank.

For example, the output for `LAST("6483-30384  ", 3)` is " ".

You can use the ALLTRIM( ) function in conjunction with LAST( ) to remove any trailing spaces in *string*.

For example, `LAST(ALLTRIM("6483-30384  "), 3)` returns "384".

Returning characters from the start of a string

If you want to return a specified number of characters from the start of a string, use the SUBSTR( ) function. For more information, see "SUBSTR( ) function" on page 810.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

LEADING( ) function

Returns a string containing a specified number of leading digits.

Syntax

```
LEADING(number, leading_digits)
```

Parameters

Name	Type	Description
number	numeric	The value to return the leading digits from.
leading_digits	numeric	The number of leading digits to be returned.

Output

Character.

Examples

Basic examples

Literal numeric input

Returns 623:

```
LEADING(6234.56, 3)
```

Returns 62345:

```
LEADING(-6234.56, 5)
```

Padding with trailing zeros

Returns 000:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
LEADING(0.00, 3)
```

Returns 00000:

```
LEADING(0.00, 5)
```

Returns 35500:

```
LEADING(3.55, 5)
```

Remarks

Use LEADING( ) to extract digits from a numeric field as a string, and filter out non-digit elements such as decimals or dollar signs.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

LENGTH( ) function

Returns the number of characters in a string.

Syntax

```
LENGTH(string)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to find the length of.

Output

Numeric.

Examples

Basic examples

Returns 15:

```
LENGTH("ABC Corporation")
```

Returns the length in characters of the **Description** field in the table layout:

```
LENGTH(Description)
```

Advanced examples

Displaying the length of each address in an address field

Create a computed field that displays the length in characters of each address in the **Vendor_Street** field. Leading and trailing blank spaces are first trimmed from the address values so they are not counted in the

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

length.

```
DEFINE FIELD Address_Length COMPUTED LENGTH(ALLTRIM(Vendor_Street))
```

Remarks

How it works

The LENGTH( ) function counts the number of characters in *string*, including any spaces, and returns the number.

Trailing spaces

Trailing spaces are counted as characters. If you do not want trailing spaces to be counted, use the TRIM ( ) or ALLTRIM( ) functions to remove them. For example:

```
LENGTH(TRIM(Vendor_Street))
```

If you create a computed field to display the length of the values in a field, and you do not remove trailing spaces, the maximum length of the field is displayed for each value.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

LEVDIST( ) function

Returns the Levenshtein distance between two specified strings, which is a measurement of how much the two strings differ.

Syntax

```
LEVDIST(string1, string2 <,case_sensitive>)
```

Parameters

Name	Type	Description
string1	character	The first string in the comparison.
string2	character	The second string in the comparison.
case_sensitive optional	logical	Specify T for a case-sensitive comparison of strings, or F to ignore case. If omitted, the default value of T is used.

Output

Numeric. The value is the Levenshtein distance between two strings.

Examples

Basic examples

Returns 3, because two substitutions and one insertion are required to transform "smith" into "Smythe":

```
LEVDIST("smith","Smythe")
```

Returns 2, because case is ignored, so only two substitutions are required to transform "smith's" into "Smythes":

```
LEVDIST("smith's","Smythes",F)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns the Levenshtein distance between each value in the **Last_Name** field and the string "Smith":

```
LEVDIST(TRIM(Last_Name),"Smith")
```

Advanced examples

Ranking values against "Smith"

Create the computed field **Lev_Dist** to display the Levenshtein distance between "Smith" and each value in the **Last_Name** field:

```
DEFINE FIELD Lev_Dist COMPUTED LEVDIST(TRIM(Last_Name),"Smith", F)
```

Add the computed field **Lev_Dist** to the view, and then quick sort it in ascending order, to rank all values in the **Last_Name** field by their amount of difference from "Smith".

Isolating fuzzy duplicates for "Smith"

Create a filter that isolates all values in the **Last_Name** field that are within a specified Levenshtein distance of "Smith":

```
SET FILTER TO LEVDIST(TRIM(Last_Name),"Smith", F) < 3
```

Changing the number in the expression allows you to adjust the amount of Levenshtein distance in the filtered values.

Remarks

When to use LEVDIST( )

Use the LEVDIST( ) function to find nearly identical values (fuzzy duplicates) or locate inconsistent spelling in manually entered data. LEVDIST( ) also identifies exact duplicates.

How it works

The LEVDIST( ) function returns the Levenshtein distance between the two evaluated strings, which is a value representing the minimum number of single character edits required to make one string identical to the other string.

Each required edit increments the value of the Levenshtein distance by 1. The greater the Levenshtein distance, the greater the difference between the two strings. A distance of zero (0) means the strings are identical.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Types of edits

The edits can be of three types:

- insertion
- deletion
- substitution

Transpositions (two adjacent letters reversed) are not recognized by the Levenshtein algorithm, and count as two edits – specifically, two substitutions.

Non-alphanumeric characters

Punctuation marks, special characters, and blanks are treated as single characters, just like letters and numbers.

The case of characters

Changing the case of a character counts as one substitution, unless you turn off case sensitivity using the *case_sensitive* setting.

The position of characters

Levenshtein distance takes the position of characters into account. The same characters ordered differently may result in a different Levenshtein distance.

Returns 2:

```
LEVDIST("abc", "dec")
```

Returns 3:

```
LEVDIST("abc", "cde")
```

Using TRIM( ) with LEVDIST( )

To ensure accurate results when using LEVDIST( ) to compare a literal string such as "Smith" with a character field, you must use the TRIM( ) function to remove trailing blanks from the field.

If you are comparing two fields, you must use the TRIM( ) function with each field.

The Levenshtein algorithm counts blanks as characters, so any trailing blanks are included in the calculation of the number of edits required to make two strings identical.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Using OMIT( ) with LEVDIST( )

The OMIT( ) function can improve the effectiveness of the LEVDIST( ) function by removing generic elements such as "Corporation" or "Inc." from field values. Removal of generic elements focuses the LEVDIST( ) string comparison on just the portion of the strings where a meaningful difference may occur.

Related functions

- **ISFUZZYDUP( )** – provides an alternate method for comparing strings based on Levenshtein distance.

  Unlike the default behavior of LEVDIST( ), ISFUZZYDUP( ) is not case-sensitive.
- **DICECOEFFICIENT( )** – de-emphasizes or completely ignores the relative position of characters or character blocks when comparing strings.
- **SOUNDSLIKE( ) and SOUNDEX( )** – compare strings based on a phonetic comparison (sound) rather than on an orthographic comparison (spelling).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

LOG( ) function

Returns the logarithm (base 10) of a numeric expression or field value with a specified number of decimal places.

Syntax

```
LOG(number, decimals)
```

Parameters

Name	Type	Description
number	numeric	The value to find the logarithm for.
decimals	numeric	The number of decimal places for the return value.

Output

Numeric.

Examples

Basic examples

Returns 3.0000:

```
LOG(1000, 4)
```

Returns 4.86:

```
LOG(72443, 2)
```

Advanced examples

Finding the cube root

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Creates a field that is the cube root of the field X to two decimal places:

```
DEFINE FIELD Cube_root COMPUTED EXP(LOG(X, 6) / 3, 2)
```

> **Note**
>
> You determine the *n*th root by dividing the log of the value by *n* and taking the exponential value of the result.

Remarks

How it works

The logarithm of a number is the exponent (or power) of 10 needed to generate that number. Therefore, the logarithm of 1000 is 3.

Related functions

The LOG( ) function is the inverse of the EXP( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

LOWER( ) function

Returns a string with alphabetic characters converted to lowercase.

Syntax

```
LOWER(string)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to convert to lowercase.

Output

Character.

Examples

Basic examples

Returns "abc":

```
LOWER("ABC")
```

Returns "abc 123 def":

```
LOWER("abc 123 DEF")
```

Returns "abcd 12":

```
LOWER("AbCd 12")
```

Returns all the values in the **Last_Name** field converted to lowercase:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
LOWER(Last_Name)
```

Remarks

How it works

The LOWER( ) function converts all alphabetic characters in *string* to lowercase. All non-alphabetic characters are left unchanged.

When to use LOWER( )

Use LOWER( ) when you search for data that is in mixed or unknown case, or when you want data formatted as lowercase text.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

LTRIM( ) function

Returns a string with leading spaces removed from the input string.

Syntax

```
LTRIM(string)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to remove leading spaces from.

Output

Character.

Examples

Basic examples

Note that in both examples the trailing spaces are not removed by the LTRIM( ) function.

Returns "Vancouver ":

```
LTRIM("  Vancouver  ")
```

Returns "New York ":

```
LTRIM("  New York  ")
```

Advanced examples

Removing non-breaking spaces

Non-breaking spaces are not removed by the LTRIM( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

If you need to remove leading non-breaking spaces, create a computed field using the following expression:

```
DEFINE FIELD Description_cleaned COMPUTED LTRIM(REPLACE(Description, CHR(160), CHR(32)))
```

The REPLACE( ) function replaces any non-breaking spaces with regular spaces, and then LTRIM( ) removes any leading regular spaces.

Remarks

How it works

The LTRIM( ) function removes leading spaces only. Spaces inside the string, and trailing spaces, are not removed.

Related functions

LTRIM( ) is related to the TRIM( ) function, which removes any trailing spaces from a string, and to the ALLTRIM( ) function, which removes both leading and trailing spaces.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

MAP( ) function

Returns a logical value indicating if a character string matches a specified format string containing wildcard characters, literal characters, or both.

Syntax

```
MAP(string, format)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to test for matches.
format	character	The data pattern, or character string, you want to compare with *string*. *format* can contain wildcard characters, literal characters, or a combination of the two: `"\9\9\9-999-9999"` The following wildcard characters are supported: ◦ `"X"` - matches any alphabetic character (a-z, A-Z, European characters). This wildcard character is not case-sensitive. You can use "X" or "x" ◦ `"9"` - matches any number (0-9) ◦ `"!"` - matches any non-blank character ◦ `"?"` - matches any character, including blanks ◦ `"\"` - escape character that specifies that the character immediately following is a literal. Use the escape character if you want to literally match any of the wildcard characters (X, x, 9, !, ?) ◦ `"\\"` - specifies a literal backslash

Output

Logical. Returns **T** (true) if a match is found, and **F** (false) otherwise.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Simple search patterns

Returns T:

```
MAP("ABC Plumbing", "xxx")
```

Returns F (*string* has only 3 numbers where a minimum of 4 are required):

```
MAP("045", "9999")
```

Escaping a wildcard

If the goal is to return T for only those values that start with the literal character "X", followed by any second letter, the *format* parameter "\XX" ensures that the first "X" in the parameter is interpreted literally and not as a wildcard.

Returns T:

```
MAP("XA-123", "XX")
```

```
MAP("GC-123", "XX")
```

```
MAP("XA-123", "\XX")
```

Returns F:

```
MAP("GC-123", "\XX")
```

Fields and patterns

Returns T for all records with invoice numbers that consist of, or that start with, two letters followed by five numbers. Returns F otherwise:

```
MAP(Invoice_Number, "XX99999")
```

Returns T for all records with invoice numbers that are exactly "AB12345", or that start with "AB12345". Returns F otherwise:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
MAP(Invoice_Number, "AB12345")
```

Returns T for all records with invoice numbers that consist of, or that start with, "AB" followed by five numbers. Returns F otherwise:

```
MAP(Invoice_Number, "AB99999")
```

Returns T for all records that do not match the standard format of social security numbers in the SSN field. Returns F otherwise:

```
NOT MAP(SSN, "999-99-9999")
```

Advanced examples

Extracting records with 10-character product codes and with the leading characters "859-"

Use an IF statement and the MAP( ) function to extract only those records that have product codes at least 10 characters long, and the leading characters "859-":

```
EXTRACT RECORD IF MAP(Product_Code, "85\9-999999") TO "Long_Codes_859"
```

Remarks

When to use MAP( )

Use the MAP( ) function to search for patterns or particular formats in alpha-numeric data. The patterns or formats can be made up of wildcard characters, literal characters, or a combination of both.

Case sensitivity

The MAP( ) function is case-sensitive when comparing two literal characters. For example, "a" is not equivalent to "A".

If *string* includes character data with inconsistent case, you can use the UPPER( ) function to convert the values to consistent case before using MAP( ).

For example:

```
MAP(UPPER(Invoice_Number), "AB99999")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Partial matching

MAP( ) supports partial matching in one situation but not in the other.

Partial matching in MAP( ) is not affected by the **Exact Character Comparisons** option (SET EXACT ON/OFF).

Partial matching supported

Partial matching is supported if the *format* value is shorter than the *string* value.

Returns T, because *format* is 7 characters and *string* is 9 characters:

```
MAP("AB1234567", "AB99999")
```

> **Note**
>
> To return True, the *format* value must appear at the start of the *string* value.

Partial matching not supported

Partial matching is not supported if the *format* value is longer than the *string* value.

Returns F, because *format* is 7 characters and *string* is 6 characters:

```
MAP("AB1234", "AB99999")
```

If *format* is longer than *string*, the result is always False.

Accounting for blank spaces

Blank spaces are treated as characters, and can be accounted for in either of two ways:

- match blanks literally, by including blanks in the *format* value at the appropriate position
- use the wildcard "?" , which matches any character, including blanks

If required, you can use the TRIM( ), LTRIM( ), or ALLTRIM( ) functions to remove leading or trailing blanks from *string*, which ensures that only text characters and any internal blanks are compared.

Concatenating fields

You can concatenate two or more fields in *string* if you want to search in more than one field in a table. The concatenated fields are treated like a single field that includes leading and trailing blanks from the individual fields, unless you use the ALLTRIM( ) function to remove them.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

MASK( ) function

Performs a bitwise AND operation on the first bytes of two character strings.

Syntax

```
MASK(character_value, character_mask)
```

Parameters

Name	Type	Description
character_value	character	The string with the byte to test.
character_mask	character	The string with the byte to test against (the mask value).

Output

Character. The output is the character representation of the binary result of the bitwise AND operation.

Examples

Basic examples

Returns "2" (00110010), the result of a bitwise AND of 3 (00110011) and 6 (00110110):

```
MASK("3", "6")
```

Remarks

When to use MASK( )

Use MASK( ) to identify specific bit patterns in a byte of data, including whether or not a particular bit is set to 1.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

How it works

The MASK( ) function performs a bitwise AND operation on the binary representations of the first characters of *character_value* and *character_mask*. The two comparison bytes are compared one bit at a time, resulting in a third binary value.

The result of each comparison of corresponding bits is either 1 or 0:

character_value bit	*character_mask* bit	Result
0	0	0
0	1	0
1	0	0
1	1	1

Comparison strings longer than one byte

If either comparison string is longer than one byte, subsequent characters are ignored.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

MATCH( ) function

Returns a logical value indicating whether the specified value matches any of the values it is compared against.

Syntax

```
MATCH(comparison_value, test <,...n>)
```

Parameters

Name	Type	Description
comparison_value	character numeric datetime	The field, expression, or literal value to test for matches.
test <,...n>	character numeric datetime	Any field, expression, or literal value you want to compare with *comparison_value*. You can specify as many test values as necessary, but all specified values must be of the same data type: `MATCH(comparison_value, `20140930`, `20141030`)`

> **Note**
>
> Inputs to the MATCH( ) function can be character, numeric, or datetime data. You cannot mix data types. All inputs must belong to the same data category.

Output

Logical. Returns **T** (true) if at least one match is found, and **F** (false) otherwise.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

> **Note**
>
> Return values for character comparisons assume that SET EXACT is OFF (the default setting), except where noted.

Testing literal values

Returns T:

```
MATCH("ABC", "BCD", "CDE", "AB")
```

Returns F:

```
MATCH(98, 99, 100, 101)
```

Testing a field

Returns T for all records that contain "Phoenix", "Austin", or "Los Angeles" in the **Vendor_City** field. Returns F otherwise:

```
MATCH(Vendor_City, "Phoenix", "Austin", "Los Angeles")
```

Returns T for all records that do not contain "Phoenix", "Austin", or "Los Angeles" in the **Vendor_City** field. Returns F otherwise:

```
NOT MATCH(Vendor_City, "Phoenix", "Austin", "Los Angeles")
```

Returns T for all records that contain "PHOENIX", "AUSTIN", or "LOS ANGELES" in the **Vendor_City** field, regardless of the case of any of the characters in the field. Returns F otherwise:

Values in the **Vendor_City** field are converted to uppercase before being compared with the uppercase city names.

```
MATCH(UPPER(Vendor_City), "PHOENIX", "AUSTIN", "LOS ANGELES")
```

Testing multiple fields

Returns T for all records that contain "Phoenix" in the **Vendor_City**, **City**, or **City_2** fields. Returns F otherwise:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
MATCH("Phoenix", Vendor_City, City, City_2)
```

SET EXACT behavior

Returns T for all records that have product codes "A", "D", or "F", or product codes beginning with "A", "D", or "F", in the **Product_Code** field. Returns F otherwise:

```
MATCH(Product_Code, "A", "D", "F")
```

Returns T for all records that have one-character product codes "A", "D", or "F" in the **Product_Code** field. Returns F otherwise (SET EXACT must be ON):

```
MATCH(Product_Code, "A", "D", "F")
```

Comparing two fields

Returns T for all records that contain identical vendor and employee addresses. Returns F otherwise:

You may need to use additional functions to standardize the format of vendor and employee addresses.

```
MATCH(Vendor_Address, Employee_Address)
```

Comparing dates

Returns T for all records with an invoice date of 30 Sep 2014 or 30 Oct 2014. Returns F otherwise:

```
MATCH(Invoice_Date, `20140930`, `20141030`)
```

Advanced examples

Extracting anomalous inventory records

Use an IF statement and the MATCH( ) function to extract records that contain different amounts in the **Inventory_Value_at_Cost** field and the computed **Cost_x_Quantity** field:

```
EXTRACT RECORD IF NOT MATCH(Inventory_Value_at_Cost, Cost_x_Quantity) TO "Non_match-
ing_amounts"
```

Extracting records for departments 101, 103, and 107

Use an IF statement and the MATCH( ) function to extract only records associated with departments 101, 103, or 107:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
EXTRACT RECORD IF MATCH(Dept, "101", "103", "107") TO "Three_Departments"
```

Remarks

Use MATCH( ) instead of the OR operator

You can use the MATCH( ) function instead of expressions that use the OR operator.

For example:

```
MATCH(City, "Phoenix", "Austin", "Los Angeles")
```

is equivalent to

```
City="Phoenix" OR City="Austin" OR City="Los Angeles"
```

Decimal precision of numeric inputs

When the numeric inputs being compared have a different decimal precision, the comparison uses the higher level of precision.

Returns T, because 1.23 is equal to 1.23:

```
MATCH(1.23, 1.23, 1.25)
```

Returns F, because 1.23 does not equal 1.234 once the third decimal place is considered:

```
MATCH(1.23, 1.234, 1.25)
```

Character parameters

Case sensitivity

The MATCH( ) function is case-sensitive when used with character data. When it compares characters, "a" is not equivalent to "A".

Returns F:

```
MATCH("a","A","B","C")
```

If you are working with data that includes inconsistent case, you can use the UPPER( ) function to convert the values to consistent case before using MATCH( ).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns T:

```
MATCH(UPPER("a"), UPPER("A"), UPPER("B"),  UPPER("C"))
```

Partial matching

Partial matching is supported for character comparisons. Either value being compared can be contained by the other value and be considered a match.

Both of these examples return T:

```
MATCH("AB", "ABC")
```

```
MATCH("ABC", "AB")
```

> **Note**
>
> The shorter value must appear at the start of the longer value to constitute a match.

Partial matching and SET EXACT

Partial matching is enabled when SET EXACT = OFF, which is the Analytics default setting. If SET EXACT = ON, partial matching is disabled and the comparison values must exactly match to constitute a match.

Both examples above are False when SET EXACT is ON.

For more information about SET EXACT (the **Exact Character Comparisons** option), see "SET command" on page 418.

Turning SET EXACT off or on

If you want to ensure that the **Exact Character Comparisons** option is not used with the MATCH( ) function, check that the option is deselected in the **Table** tab in the **Options** dialog box (**Tools > Options**).

If you are using a script, you can add the `SET EXACT OFF` command before the MATCH( ) function appears. If required, you can restore the previous state with the `SET EXACT ON` command.

Datetime parameters

A date, datetime, or time field specified as a function input can use any date, datetime, or time format, as long as the field definition correctly defines the format.

Mixing date, datetime, and time inputs

You are not prevented from mixing date, datetime, and time values in the MATCH( ) function's inputs, but mixing these Datetime subtypes can give results that are not meaningful.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics uses serial number equivalents to process datetime calculations, so even if you are interested in only the date portion of a datetime value, the time portion still forms part of the calculation.

Consider the following examples:

Returns T, because 31 December 2014 matches the second *test* value:

```
MATCH(`20141231`,`20141229`,`20141231`)
```

Returns F, even though the *comparison_value* and the second *test* value have an identical date of 31 December 2014:

```
MATCH(`20141231 120000`,`20141229`,`20141231`)
```

If we look at the serial number equivalent of these two expressions, we can see why the second one evaluates to false.

Returns T, because the serial number *comparison_value* is equal to the second serial number *test*:

```
MATCH(42003.000000, 42001.000000, 42003.000000)
```

Returns F, because the serial number *comparison_value* does not equal any of the *test* values:

```
MATCH(42003.500000, 42001.000000, 42003.000000)
```

The date portion of the serial numbers `42003.500000` and `42003.000000` match, but the time portions do not. `0.500000` is the serial number equivalent of 12:00 PM.

Harmonize Datetime subtypes

To avoid the problems that can be caused by mixing Datetime subtypes, you can use functions to harmonize the subtypes.

For example, this expression, which uses the same initial values as the second example above, returns T rather than F:

```
MATCH(CTOD(DATE(`20141231 120000`,"YYYYMMDD"),"YYYYMMDD"),`20141229`,
`20141231`)
```

Specifying a literal date, datetime, or time value

When specifying a literal date, datetime, or time value for any of the function inputs, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- **Datetime values** - you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.
- **Time values** - you must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

Example formats	Example literal values
YYYYMMDD	`20141231`
YYMMDD	`141231`
YYYYMMDD hhmmss	`20141231 235959`
YYMMDDthhmm	`141231t2359`
YYYYMMDDThh	`20141231T23`
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`20141231 235959-0500`
YYMMDD hhmm+/-hh (UTC offset)	`141231 2359+01`
thhmmss	`t235959`
Thhmm	`T2359`
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

MAXIMUM( ) function

Returns the maximum value in a set of numeric values, or the most recent value in a set of datetime values.

Syntax

```
MAXIMUM(value_1, value_2 <,...n>)
```

Parameters

Name	Type	Description
value_1, *value_2 <, ...n>*	numeric datetime	The values to compare, separated by commas. All values must be of the same data type. Additionally, datetime values must be of the same subtype. You cannot mix date, datetime, or time values in a single execution of the function.

Output

Numeric or Datetime.

Examples

Basic examples

Literal numeric input

Returns 7:

```
MAXIMUM(4, 7)
```

Returns 8:

```
MAXIMUM(4, 7, 3, 8)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns 8.00:

```
MAXIMUM(4, 7.25, 3, 8)
```

Literal datetime input

Returns `20161231`:

```
MAXIMUM(`20161231`, `20161229`, `20161230`)
```

Returns `20161231 23:59:59`:

```
MAXIMUM(`20161231 235959`, `20161229 235959`)
```

Returns `23:59:59`:

```
MAXIMUM(`.235957`, `.235959`, `.235958`)
```

Field input

Returns the most recent date among the three fields for each record:

```
MAXIMUM(PO_Date, Invoice_Date, Payment_Date)
```

Advanced examples

Creating a computed field with a minimum default value

If you have a table of overdue accounts, create an **Interest_Due** computed field with a minimum default value of $1.00:

```
DEFINE FIELD Interest_Due COMPUTED MAXIMUM(BALANCE * ANNUAL_RATE, 1)
```

If the balance multiplied by the interest rate is less than one dollar, MAXIMUM( ) returns 1. Otherwise, MAXIMUM( ) returns the calculated interest amount.

Discovering dates past the end of a quarter

To discover if any dates across multiple fields are past the end of a quarter, create a computed field with an expression like the one below:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DEFINE FIELD Past_Qtr COMPUTED MAXIMUM(PO_Date, Invoice_Date, Payment_Date,
`20160331`)
```

- Records with all dates on or before 31 Mar 2016 return `20160331`.
- Records with one or more dates after 31 Mar 2016 return the most recent date among the three fields.

Remarks

How decimal places work in sets of numeric values

If the numeric values being compared do not have the same number of decimal places, the result is adjusted to the largest number of decimal places.

Returns 20.400:

```
MAXIMUM(3.682, 10.88, 20.4)
```

You can use the DECIMALS( ) function to adjust the number of decimals for *value* parameters.

Returns 20.40:

```
MAXIMUM(DECIMALS(3.682, 2), 10.88, 20.4)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

MINIMUM( ) function

Returns the minimum value in a set of numeric values, or the oldest value in a set of datetime values.

Syntax

```
MINIMUM(value_1, value_2<,...n>)
```

Parameters

Name	Type	Description
value_1, *value_2<, ...n>*	numeric datetime	The values to compare, separated by commas. All values must be of the same data type. Additionally, datetime values must be of the same subtype. You cannot mix date, datetime, or time values in a single execution of the function.

Output

Numeric or Datetime.

Examples

Basic examples

Literal numeric input

Returns 4:

```
MINIMUM(4, 7)
```

Returns 3:

```
MINIMUM(4, 7, 3, 8)
```

Returns 3.00:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
MINIMUM(4, 7.25, 3, 8)
```

Literal datetime input

Returns `20161229`:

```
MINIMUM(`20161231`, `20161229`, `20161230`)
```

Returns `20161229 23:59:59`:

```
MINIMUM(`20161231 235959`, `20161229 235959`)
```

Returns `23:59:57`:

```
MINIMUM(`.235957`, `.235959`, `.235958`)
```

Field input

Returns the oldest date among the three fields for each record:

```
MINIMUM(PO_Date, Invoice_Date, Payment_Date)
```

Advanced examples

Identifying the lowest value among multiple fields

Create a computed field to identify the lowest value among the **Cost**, **Sale_Price**, and **Discount_Price** fields:

```
DEFINE FIELD Low_Value COMPUTED MINIMUM(Cost, Sale_Price, Discount_Price)
```

Discovering dates before the beginning of a quarter

To discover if any dates across multiple fields are before the beginning of a quarter, create a computed field with an expression like the one below:

```
DEFINE FIELD Pre_Qtr COMPUTED MINIMUM(PO_Date, Invoice_Date, Payment_Date,
`20160101`)
```

- Records with all dates on or after 01 Jan 2016 return `20160101`.
- Records with one or more dates before 01 Jan 2016 return the oldest date among the three fields.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

How decimal places work in sets of numeric values

If the numeric values being compared do not have the same number of decimal places, the result is adjusted to the largest number of decimal places.

Returns 3.600:

```
MINIMUM(3.6,10.88, 20.482)
```

You can use the DECIMALS( ) function to adjust the number of decimals for *value* parameters.

Returns 3.60:

```
MINIMUM(3.6,10.88, DECIMALS(20.482, 2))
```

The MIN( ) abbreviation

In ACLScript, you can use the abbreviation MIN( ) for the MINIMUM( ) function even though it does not uniquely identify the function, which is the normal requirement when abbreviating function names.

MIN( ) could also be an abbreviation for MINUTE( ), however Analytics reserves the abbreviation MIN( ) for the MINIMUM( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

MINUTE( ) function

Extracts the minutes from a specified time or datetime and returns it as a numeric value.

Syntax

```
MINUTE(time/datetime)
```

Parameters

Name	Type	Description
time/datetime	datetime	The field, expression, or literal value to extract the minutes from.

Output

Numeric.

Examples

Basic examples

Returns 59:

```
MINUTE(`t235930`)
```

```
MINUTE(`20141231 235930`)
```

Returns the minutes for each value in the **Call_start_time** field:

```
MINUTE(Call_start_time)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Abbreviating MINUTE( ) in scripts

In ACLScript, if you abbreviate the MINUTE( ) function, you must use at least the first four letters (MINU). Analytics reserves the abbreviation MIN for the MINIMUM( ) function.

Parameter details

A field specified for *time/datetime* can use any time or datetime format, as long as the field definition correctly defines the format.

Specifying a literal time or datetime value

When specifying a literal time or datetime value for *time/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231 235959` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Time values** – you can use any of the time formats listed in the table below. You must use a separator before a standalone time value for the function to operate correctly. Valid separators are the letter 't', or the letter 'T'. You must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).
- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.

Example formats	Example literal values
thhmmss	`` `t235959` ``
Thhmm	`` `T2359` ``
YYYYMMDD hhmmss	`` `20141231 235959` ``
YYMMDDthhmm	`` `141231t2359` ``
YYYYMMDDThh	`` `20141231T23` ``
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`` `20141231 235959-0500` ``
YYMMDD hhmm+/-hh (UTC offset)	`` `141231 2359+01` ``

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Example formats	Example literal values
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

MOD( ) function

Returns the remainder from dividing two numbers.

Syntax

```
MOD(number, divisor_number)
```

Parameters

Name	Type	Description
number	numeric	The number to divide.
divisor_number	numeric	The number to divide *number* by. If *number* or *divisor_number*, or both, include decimals, the output has the same decimal precision as the input value with the higher number of decimal places. For example, the output for `MOD(45.35, 5.3)` is "2.95".

Output

Numeric.

Examples

Basic examples

Returns 3:

```
MOD(93, 10)
```

Returns 2.0:

```
MOD(66, 16.00)
```

Returns 3.45:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
MOD(53.45, 10)
```

Advanced examples

Calculating an anniversary date

Defines a field that shows the number of months since the last anniversary:

```
DEFINE FIELD Months_since_last_anniversary COMPUTED MOD(Months_of_service, 12)
```

Remarks

When to use MOD( )

Use the MOD( ) function to test whether two numbers divide evenly, or to isolate the remainder of a division calculation. This function divides one number by another and returns the remainder.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

MONTH( ) function

Extracts the month from a specified date or datetime and returns it as a numeric value (1 to 12).

Syntax

```
MONTH(date/datetime)
```

Parameters

Name	Type	Description
date/datetime	datetime	The field, expression, or literal value to extract the month from.

Output

Numeric.

Examples

Basic examples

Returns 12:

```
MONTH(`20141231`)
```

```
MONTH(`20141231 235959`)
```

Returns the month for each value in the **Invoice_date** field:

```
MONTH(Invoice_date)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Parameter details

A field specified for *date/datetime* can use any date or datetime format, as long as the field definition correctly defines the format.

Specifying a literal date or datetime value

When specifying a literal date or datetime value for *date/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.
- **Time values** – you must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

Example formats	Example literal values
YYYYMMDD	`` `20141231` ``
YYMMDD	`` `141231` ``
YYYYMMDD hhmmss	`` `20141231 235959` ``
YYMMDDthhmm	`` `141231t2359` ``
YYYYMMDDThh	`` `20141231T23` ``
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`` `20141231 235959-0500` ``
YYMMDD hhmm+/-hh (UTC offset)	`` `141231 2359+01` ``
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Related functions

If you need to return the name of the month of the year, use CMOY( ) instead of MONTH( ).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

NOMINAL( ) function

Returns the nominal annual interest rate on a loan.

Syntax

```
NOMINAL(effective_rate, periods)
```

Parameters

Name	Type	Description
effective_rate	numeric	The effective annual interest rate.
periods	numeric	The number of compounding periods per year. **Note** Specify an integer. If you specified a decimal portion, it is truncated.

Output

Numeric. The rate is calculated to eight decimals places.

Examples

Basic examples

Returns 0.17998457 (18%), the nominal annual rate of interest on the unpaid balance of a credit card that charges an effective annual rate of 19.56%:

```
NOMINAL(0.1956, 12)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

What is the nominal interest rate?

The nominal annual interest rate on a loan is the stated or posted rate of interest, without taking into account interest on the remaining balance, compounded monthly or daily.

Related functions

The EFFECTIVE( ) function is the inverse of the NOMINAL( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

NORMDIST( ) function

Returns the probability that a random variable from a normally distributed data set is less than or equal to a specified value, or exactly equal to a specified value.

Syntax

```
NORMDIST(x, mean, standard_deviation, cumulative)
```

Parameters

Name	Type	Description
x	numeric	The value for which you want to calculate the probability.
mean	numeric	The mean value of the data set.
standard_deviation	numeric	The standard deviation of the data set. The *standard_deviation* value must be greater than 0.
cumulative	logical	Specify T to calculate the probability that a random variable is less than or equal to *x* (cumulative probability), or F to calculate the probability that a random variable is exactly equal to *x* (simple probability).

Output

Numeric.

Examples

Basic examples

Returns 0.908788780274132:

```
NORMDIST(42, 40, 1.5, T)
```

Returns 0.109340049783996:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
NORMDIST(42, 40, 1.5, F)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

NORMSINV( ) function

Returns the z-score associated with a specified probability in a standard normal distribution. The z-score is the number of standard deviations a value is from the mean of a standard normal distribution.

Syntax

```
NORMSINV(probability)
```

Parameters

Name	Type	Description
probability	numeric	The probability for which you want to calculate the z-score.

Output

Numeric.

Examples

Basic examples

Returns 1.333401745213610:

```
NORMSINV(0.9088)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

NOW( ) function

Returns the current operating system time as a Datetime data type.

Syntax

```
NOW()
```

Parameters

This function does not have any parameters.

Output

Datetime.

Examples

Basic examples

Returns the current operating system time as a datetime value, for example, `t235959`, displayed using the current Analytics time display format:

```
NOW()
```

Remarks

Related functions

If you need to return the current operating system time as a character string, use TIME( ) instead of NOW( ).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

NPER( ) function

Returns the number of periods required to pay off a loan.

Syntax

```
NPER(rate, payment, amount <,type>)
```

Parameters

Name	Type	Description
rate	numeric	The interest rate per period.
payment	numeric	The payment per period.
amount	numeric	The principal amount of the loan.
type optional	numeric	The timing of payments: ◦ 0 - payment at the end of a period ◦ 1 - payment at the beginning of a period If omitted, the default value of 0 is used.

Output

Numeric.

Examples

Basic examples

Returns 300.00, the number of months required to pay off a $275,000 loan at 6.5% per annum, with payments of $1,856.82 due at the end of each month:

```
NPER(0.065/12, 1856.82, 275000, 0)
```

Returns 252.81, the number of months required to pay off the same loan, with payments of $2,000 due at the end of each month:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
NPER(0.065/12, 2000, 275000, 0)
```

Returns 249.92, the number of months required to pay off the same loan, with payments of $2,000 due at the beginning of each month:

```
NPER(0.065/12, 2000, 275000, 1)
```

Advanced examples

Annuity calculations

Annuity calculations involve four variables:

- **present value, or future value** – $21,243.39 and $ 26,973.46 in the examples below
- **payment amount per period** – $1,000.00 in the examples below
- **interest rate per period** – 1% per month in the examples below
- **number of periods** – 24 months in the examples below

If you know the value of three of the variables, you can use an Analytics function to calculate the fourth.

I want to find:	Analytics function to use:
Present value	PVANNUITY( ) Returns 21243.39: `PVANNUITY(0.01, 24, 1000)`
Future value	FVANNUITY( ) Returns 26973.46: `FVANNUITY(0.01, 24, 1000)`
Payment amount per period	PMT( ) Returns 1000: `PMT(0.01, 24, 21243.39)`
Interest rate per period	RATE( ) Returns 0.00999999 (1%): `RATE(24, 1000, 21243.39)`
Number of periods	NPER( )

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

I want to find:	Analytics function to use:
	Returns 24.00: NPER(0.01, 1000, 21243.39)

Annuity formulas

The formula for calculating the **present value** of an ordinary annuity (payment at the end of a period):

$$PV_A = Pmt\left[\frac{1-\frac{1}{(1+i)^N}}{i}\right]$$

$$21243.39 = 1000\left[\frac{1-\frac{1}{(1+0.01)^{24}}}{0.01}\right]$$

The formula for calculating the **future value** of an ordinary annuity (payment at the end of a period):

$$FV_A = Pmt\left[\frac{(1+i)^N-1}{i}\right]$$

$$26973.46 = 1000\left[\frac{(1+0.01)^{24}-1}{0.01}\right]$$

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

OCCURS( ) function

Returns a count of the number of times a substring occurs in a specified character value.

Syntax

```
OCCURS(string, search_for)
```

Parameters

Name	Type	Description
string	character	The value to search in. You can concatenate two or more fields if you want to search in more than one field in a table: `OCCURS(First_Name+Last_Name,"John")`
search_for	character	The value to search for. The search is case-sensitive.

Output

Numeric.

Examples

Basic examples

Returns 2:

```
OCCURS("abc/abc/a","ab")
```

Returns 3:

```
OCCURS("abc/abc/a","a")
```

Returns the number of times a hyphen occurs in each value in the **Invoice_Number** field:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
OCCURS(Invoice_Number, "-")
```

Advanced examples

Finding invoice numbers with more than one hyphen

If invoice numbers in a table should have only one hyphen, use the OCCURS( ) function to create a filter that isolates invoice numbers that have two or more hyphens:

```
SET FILTER TO OCCURS(Invoice_Number, "-") > 1
```

Finding occurrences of one field's value in another field

Use OCCURS( ) to find occurrences of one field's value in another field. For example, you could create a filter that isolates records in which **Last_Name** values occur in the **Full_Name** field:

```
SET FILTER TO OCCURS(Full_Name, ALLTRIM(Last_Name)) = 1
```

Including the ALLTRIM( ) function in the expression removes any leading or trailing spaces from the **Last_Name** field, ensuring that only text values are compared.

Performing case-sensitive searches

Unlike the FIND( ) function, the OCCURS( ) function is case sensitive, which allows you to perform case-sensitive searches.

The following expression isolates all records that contain the name "UNITED EQUIPMENT", in uppercase, in the **Vendor_Name** field, while ignoring occurrences of "United Equipment".

```
SET FILTER TO OCCURS(Vendor_Name, "UNITED EQUIPMENT") > 0
```

If you want to find all occurrences of "United Equipment" regardless of casing, use the UPPER( ) function to convert the search field values to uppercase:

```
SET FILTER TO OCCURS(UPPER(Vendor_Name), "UNITED EQUIPMENT") > 0
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

OFFSET( ) function

Returns the value of a field with the starting position offset by a specified number of bytes.

Syntax

```
OFFSET(field, number_of_bytes)
```

Parameters

Name	Type	Description
field	character numeric datetime	A field name.
number_of_bytes	numeric	Any positive numeric expression.

Output

The return value is the same data type as the input *field* parameter.

Examples

Basic examples

If you have a field called "Number" that contains the value "1234567890" and you define an overlapping field called "Offset_Number" that has a starting position of 1, a length of 3, and no decimals places, you can use the OFFSET( ) function to shift the numbers in the field.

Returns 123:

```
OFFSET(Offset_Number,0)
```

Returns 234:

```
OFFSET(Offset_Number,1)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns 789:

```
OFFSET(Offset_Number,6)
```

Remarks

You can use this function to temporarily offset the starting position of a field. This is useful when you are processing data where the field starting position is variable.

If you use the OFFSET( ) function with conditional computed fields, any fields referenced in the IF test will also be offset.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

OMIT( ) function

Returns a string with one or more specified substrings removed.

Syntax

```
OMIT(string1, string2 <,case_sensitive>)
```

Parameters

Name	Type	Description
string1	character	The field, expression, or literal value to remove one or more substrings from.
string2	character	One or more substrings to remove. ◦ Use commas to separate multiple substrings. ◦ Use a space after a comma only if it is part of the substring you want to remove. ◦ If the double quotation mark character occurs in any of the sub-strings, enclose the entire *string2* parameter in single quotation marks (' ') rather than double quotation marks. ◦ To omit a comma, place a single comma last in the list of sub-strings, followed immediately by the closing quotation mark (see the final example below).
case_sensitive optional	logical	Specify T to make the substrings case-sensitive, or F to ignore case. If *case_sensitive* is omitted, the default value of T is used.

Output

Character.

Examples

Basic examples

Literal character input

Returns "Intercity Couriers":

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
OMIT("Intercity Couriers Corporation", " Corporation, Corp.")
```

Returns "Inter-city Couriers":

```
OMIT("Inter-city Couriers Corp.", " Corporation, Corp.")
```

> **Note**
>
> The Levenshtein distance between the returned values in the first two examples is 1. If the generic elements are not removed, the distance between the two examples is 8, which could allow the values to escape detection as fuzzy duplicates.

Field input

Returns all the values in the **Vendor_Name** field with generic elements such as "Corporation" and "Inc." removed:

```
OMIT(Vendor_Name," Corporation, Corp., Corp, Inc., Inc, Ltd., Ltd")
```

Returns all the values in the **Vendor_Name** field with generic elements such as "Corporation" and "Inc." removed:

```
OMIT(Vendor_Name," ,.,Corporation,Corp,Inc,Ltd")
```

> **Note**
>
> The two preceding examples both return the same results but the syntax for the second example is more efficient.

Returns all the values in the **Vendor_Name** field with "Corporation" and "Corp" removed, and all commas removed:

```
OMIT(Vendor_Name," Corporation, Corp,")
```

Remarks

OMIT( ) can remove substrings as units

The OMIT( ) function removes one or more substrings from a string. It differs from functions such as CLEAN( ), EXCLUDE( ), INCLUDE( ), and REMOVE( ) because it matches and removes characters on a substring basis rather than on a character-by-character basis. Substring removal allows you to remove specific words, abbreviations, or repeated sequences of characters from a string without affecting the remainder of the string.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

A helper function for fuzzy comparisons

OMIT( ) can improve the effectiveness of the LEVDIST( ) or ISFUZZYDUP( ) functions, or the FUZZYDUP or FUZZYJOIN commands, by removing generic elements such as "Corporation" or "Inc." from field values. Removal of generic elements focuses the string comparisons performed by these functions and commands on just the portion of the strings where a meaningful difference may occur.

How the order of substrings affects results

If you specify multiple substrings for removal, the order in which you list them in *string2* can affect the output results.

When the OMIT( ) function is processed, the first substring is removed from all values that contain it, the second substring is then removed from all values that contain it, and so on. If one substring forms part of another substring – for example, "Corp" and "Corporation" – removing the shorter substring first also alters values containing the longer substring ("Corporation" becomes "oration") and prevents the longer substring from being found.

To avoid this situation, specify longer substrings before any shorter substrings they contain. For example:

```
OMIT(Vendor_Name," Corporation, Corp., Corp")
```

Try removing special characters first

You can specify single-character substrings, such as punctuation marks, special characters, and a blank space, which further reduces the generic material in strings. It may be more efficient to remove a single character first, such as a period or a blank, which reduces the number of substring variations you subsequently need to specify. Compare the third and fourth examples above. They both return the same results, but the fourth example is more efficient.

Dealing with blanks or spaces

Blanks or spaces in substrings are treated like any other character. You must explicitly specify each blank you want removed as part of a substring. For example, if you specify an ampersand without any blanks ("&"), "Ricoh Sales & Service" becomes "Ricoh Sales Service". If you include blanks (" & "), "Ricoh Sales & Service" becomes "Ricoh SalesService".

If you specify a blank that is not part of the substring, the substring will not be found. For example, if you specify an ampersand with blanks (" & "), "Ricoh Sales&Service" remains unchanged.

When using commas to separate multiple substrings, follow the comma with a space only if it corresponds with the actual substring you want to remove.

One approach to dealing with blanks is to remove all blanks from a field first, by specifying a blank as a single-character substring prior to specifying any other substrings.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Review the results of using OMIT( )

After using OMIT( ) to created a computed field, review the contents of the field to confirm that you have not inadvertently omitted meaningful portions of strings. For example, omitting "Co" gets rid of a common abbreviation for "Company", but it also removes the letters "Co" from two locations in "Coca-Cola".

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PACKED( ) function

Returns numeric data converted to the Packed data type.

Syntax

```
PACKED(number, length_of_result)
```

Parameters

Name	Type	Description
number	numeric	The numeric value or field to convert.
length_of_result	numeric	The number of bytes to use in the output string.

Output

Numeric.

Examples

Basic examples

Integer and decimal input

Returns 00075C:

```
PACKED(75, 3)
```

```
PACKED(7.5, 3)
```

Truncated digits in output

Returns 00000012456D:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
PACKED(-12.456, 6)
```

Returns 456D:

```
PACKED(-12.456, 2)
```

Advanced examples

Creating an 8-byte field to update a mainframe

You need to create an 8-byte field containing each employee's salary as a PACKED number for uploading to a mainframe:

```
EXTRACT PACKED(SALARY, 8) AS "Salary_Export" TO "export"
```

Remarks

What is Packed data?

The Packed data type is used by mainframe operating systems to store numeric values in a format that uses minimal storage space. The Packed data type stores two digits in each byte, and the last byte indicates whether the value is positive or negative.

When to use PACKED( )

Use the PACKED( ) function to convert numeric data to the Packed format for export to mainframe systems.

Truncated return values

If the *length_of_result* value is shorter than the length of the *number* value, the additional digits are truncated.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PI( ) function

Returns the value of pi to 15 decimal places.

Syntax

```
PI( )
```

Parameters

This function does not have any parameters.

Output

Numeric.

Examples

Basic examples

Returns 3.141592653589793 (the value of pi to 15 decimal places):

```
PI( )
```

Returns 1.047197551196598 (the equivalent in radians of 60 degrees):

```
60 * PI( )/180
```

Advanced examples

Using degrees as input

Returns 0.866025403784439 (the sine of 60 degrees):

```
SIN(60 * PI( )/180)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

When to use PI( )

Use PI( ) to convert degrees to radians: (*degrees* * PI( )/180) = *radians*. Radians are the required input for three of Analytics's math functions: SIN( ), COS( ), and TAN( ).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PMT( ) function

Returns the amount of the periodic payment (principal + interest) required to pay off a loan.

Syntax

PMT(*rate*, *periods*, *amount* <,*type*>)

Parameters

Name	Type	Description
rate	numeric	The interest rate per period.
periods	numeric	The total number of payment periods.
amount	numeric	The principal amount of the loan.
type optional	numeric	The timing of payments: ◦ 0 - payment at the end of a period ◦ 1 - payment at the beginning of a period If omitted, the default value of 0 is used.

> **Note**
>
> You must use consistent time periods when specifying *rate* and *periods* to ensure that you are specifying interest rate **per period**.
>
> For example:
>
> - for monthly payments on a two-year loan or investment with interest of 5% per annum, specify 0.05/12 for *rate* and 2 * 12 for *periods*
> - for annual payments on the same loan or investment, specify 0.05 for *rate* and 2 for *periods*

Output

Numeric.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Returns 1856.82, the monthly payment (principal + interest) required to pay off a twenty-five year, $275,000 loan at 6.5% per annum, with payments due at the end of the month:

```
PMT(0.065/12, 12*25, 275000, 0)
```

Returns 1846.82, the monthly payment (principal + interest) required to pay off the same loan, with payments due at the beginning of the month:

```
PMT(0.065/12, 12*25, 275000, 1)
```

Advanced examples

Annuity calculations

Annuity calculations involve four variables:

- **present value, or future value** – $21,243.39 and $ 26,973.46 in the examples below
- **payment amount per period** – $1,000.00 in the examples below
- **interest rate per period** – 1% per month in the examples below
- **number of periods** – 24 months in the examples below

If you know the value of three of the variables, you can use an Analytics function to calculate the fourth.

I want to find:	Analytics function to use:
Present value	PVANNUITY( ) Returns 21243.39: `PVANNUITY(0.01, 24, 1000)`
Future value	FVANNUITY( ) Returns 26973.46: `FVANNUITY(0.01, 24, 1000)`
Payment amount per period	PMT( ) Returns 1000:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

I want to find:	Analytics function to use:
	PMT(0.01, 24, 21243.39)
Interest rate per period	RATE( ) Returns 0.00999999 (1%): RATE(24, 1000, 21243.39)
Number of periods	NPER( ) Returns 24.00: NPER(0.01, 1000, 21243.39)

Annuity formulas

The formula for calculating the **present value** of an ordinary annuity (payment at the end of a period):

$$PV_A = Pmt\left[\frac{1-\frac{1}{(1+i)^N}}{i}\right]$$

$$21243.39 = 1000\left[\frac{1-\frac{1}{(1+0.01)^{24}}}{0.01}\right]$$

The formula for calculating the **future value** of an ordinary annuity (payment at the end of a period):

$$FV_A = Pmt\left[\frac{(1+i)^N-1}{i}\right]$$

$$26973.46 = 1000\left[\frac{(1+0.01)^{24}-1}{0.01}\right]$$

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PPMT( ) function

Returns the principal paid on a loan for a single period.

Syntax

PPMT(*rate*, *specified_period*, *periods*, *amount* <,*type*>)

Parameters

Name	Type	Description
rate	numeric	The interest rate per period.
specified_period	numeric	The period for which you want to find the principal payment.
periods	numeric	The total number of payment periods.
amount	numeric	The principal amount of the loan.
type optional	numeric	The timing of payments: ◦ 0 - payment at the end of a period ◦ 1 - payment at the beginning of a period If omitted, the default value of 0 is used.

> **Note**
>
> You must use consistent time periods when specifying *rate* and *periods* to ensure that you are specifying interest rate **per period**.
>
> For example:
>
> - for monthly payments on a two-year loan or investment with interest of 5% per annum, specify 0.05/12 for *rate* and 2 * 12 for *periods*
> - for annual payments on the same loan or investment, specify 0.05 for *rate* and 2 for *periods*

Output

Numeric.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Returns 367.24, the principal paid in the first month of a twenty-five year, $275,000 loan at 6.5% per annum, with payments due at the end of the month:

```
PPMT(0.065/12, 1, 12*25, 275000, 0)
```

Returns 1846.82, the principal paid on the same loan in the last month of the loan:

```
PPMT(0.065/12, 300, 12*25, 275000, 0)
```

Remarks

Related functions

The IPMT( ) function is the complement of the PPMT( ) function.

The CUMPRINC( ) function calculates principal paid during a range of periods.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PROPER( ) function

Returns a string with the first character of each word set to uppercase and the remaining characters set to lowercase.

Syntax

```
PROPER(string)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to convert to proper case.

Output

Character.

Examples

Basic examples

Returns "John Doe":

```
PROPER("JOHN DOE")
```

Returns "John Doe":

```
PROPER("john doe")
```

Returns "1550 Alberni St.":

```
PROPER("1550 ALBERNI st.")
```

Returns "Bill O'Hara":

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
PROPER("BILL O'HARA")
```

Returns all the values in the **Company_Name** field converted to proper case:

```
PROPER(Company_Name)
```

Remarks

How it works

The PROPER( ) function converts the first character in *string*, and any subsequent character preceded by a blank, to uppercase.

Subsequent characters preceded by a hyphen, an apostrophe, an ampersand (&), and several other punctuation marks and special characters are also converted to uppercase. All other alphabetic characters are converted to lowercase.

When to use PROPER( )

The most common use for PROPER( ) is to convert names stored as all uppercase or all lowercase in source data into proper case format, so the name is displayed correctly in a form letter or report.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PROPERTIES( ) function

Returns properties information for the specified Analytics project item.

Syntax

```
PROPERTIES(name, obj_type, info_type)
```

Parameters

Name	Type	Description
name	character	The name of the Analytics project item you want information about. *name* is not case-sensitive. If the project item is an Analytics table, specify the table layout name, not the data file name. For example: "Invoices", not "january_invoices.fil" If you are using the PROPERTIES( ) function to return the name of the active table, specify the *name* "activetable"
obj_type	character	The type of Analytics project item referred to by *name*. **Note** Currently, "table" is the only type of project item supported.
info_type	character	The type of information you want about the Analytics project item. For more information, see "Types of properties information" on page 700.

Output

Character. The maximum length of the output string is 260 characters. If properties information cannot be found, an empty string is returned.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Information about the Analytics data file (.fil)

Returns "Ap_Trans.fil":

```
PROPERTIES("Ap_Trans", "table", "filename")
```

Returns "C:\ACL DATA\Sample Data Files":

```
PROPERTIES("Ap_Trans", "table", "filepath")
```

Information about the open Analytics table

Returns "Ap_Trans":

```
PROPERTIES("activetable", "table", "open")
```

Information about an external data source

Returns "Trans_May.xls":

```
PROPERTIES("Trans_May", "table", "sourcename")
```

Returns "C:\Project Data\Monthly Invoices_Excel":

```
PROPERTIES("Trans_May", "table", "sourcepath")
```

Returns "EXCEL":

```
PROPERTIES("Trans_May", "table", "sourcetype")
```

Remarks

File information

Information types starting with "file" provide information about the Analytics data file (.fil) associated with an Analytics table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Source information

Information types starting with "source" provide information about external data sources that can be associated with an Analytics table. Only those external data sources that support refreshing an Analytics table can be reported on using the PROPERTIES( ) function:

- Microsoft Excel
- Microsoft Access
- Delimited text
- Adobe Acrobat (PDF)
- Print Image (Report)
- SAP private file format/DART
- XML
- XBRL
- ODBC data sources

Types of properties information

The table below lists the types of properties information that can be returned by the PROPERTIES( ) function. Analytics tables are the only Analytics project items that can currently be used with the PROPERTIES ( ) function:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

obj_type	info_type	Returns:
"table"	"filename"	The name of the data file associated with the Analytics table.
	"filepath"	The path of the data file associated with the Analytics table.
	"filesize"	The size, in KB, of the data file associated with the Analytics table.
	"filemodifiedat"	The time and date that the data file associated with the Analytics table was last modified.
	"sourcename"	The name of the data source associated with the Analytics table. Data sources can be external files such as Excel, Access, PDF, XML, or delimited text files, or ODBC data sources.
	"sourcepath"	The path of the data source associated with the Analytics table. Not supported for ODBC data sources.
	"sourcetype"	The type of the data source associated with the Analytics table.
	"sourcesize"	The size, in KB, of the data source associated with the Analytics table. Not supported for ODBC data sources.
	"sourcemodifiedat"	The time and date that the data source associated with the Analytics table was last modified. Not supported for ODBC data sources.
	"open"	The name of the currently active Analytics table. **Note** Multiple Analytics tables can be open at the same time, but in the user interface only one table at a time can be active.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PVANNUITY( ) function

Returns the present value of a series of future payments calculated using a constant interest rate. Present value is the current, lump-sum value.

Syntax

PVANNUITY(*rate*, *periods*, *payment* <,*type*>)

Parameters

Name	Type	Description
rate	numeric	The interest rate per period.
periods	numeric	The total number of payment periods.
payment	numeric	The payment per period. The payment amount must remain constant over the term of the annuity.
type optional	numeric	The timing of payments: ◦ 0 - payment at the end of a period ◦ 1 - payment at the beginning of a period If omitted, the default value of 0 is used.

> **Note**
>
> You must use consistent time periods when specifying *rate*, *periods*, and *payment* to ensure that you are specifying interest rate **per period**.
>
> For example:
>
> - for a monthly *payment* on a two-year loan or investment with interest of 5% per annum, specify 0.05/12 for *rate* and 2 * 12 for *periods*
> - for an annual *payment* on the same loan or investment, specify 0.05 for *rate* and 2 for *periods*

Output

Numeric. The result is calculated to two decimal places.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Monthly payments

Returns 21455.82, the present value of $1,000 paid at the beginning of each month for 2 years at 1% per month, compounded monthly:

```
PVANNUITY(0.01, 2*12, 1000, 1)
```

Annual payments

Returns 20280.61, the present value of $12,000 paid at the end of each year for 2 years at 12% per annum, compounded annually:

```
PVANNUITY(0.12, 2, 12000, 0)
```

Advanced examples

Annuity calculations

Annuity calculations involve four variables:

- **present value, or future value** – $21,243.39 and $ 26,973.46 in the examples below
- **payment amount per period** – $1,000.00 in the examples below
- **interest rate per period** – 1% per month in the examples below
- **number of periods** – 24 months in the examples below

If you know the value of three of the variables, you can use an Analytics function to calculate the fourth.

I want to find:	Analytics function to use:
Present value	PVANNUITY( ) Returns 21243.39: `PVANNUITY(0.01, 24, 1000)`
Future value	FVANNUITY( ) Returns 26973.46: `FVANNUITY(0.01, 24, 1000)`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

I want to find:	Analytics function to use:
Payment amount per period	PMT( ) Returns 1000: PMT(0.01, 24, 21243.39)
Interest rate per period	RATE( ) Returns 0.00999999 (1%): RATE(24, 1000, 21243.39)
Number of periods	NPER( ) Returns 24.00: NPER(0.01, 1000, 21243.39)

Annuity formulas

The formula for calculating the **present value** of an ordinary annuity (payment at the end of a period):

$$PV_A = Pmt\left[\frac{1-\frac{1}{(1+i)^N}}{i}\right]$$

$$21243.39 = 1000\left[\frac{1-\frac{1}{(1+0.01)^{24}}}{0.01}\right]$$

The formula for calculating the **future value** of an ordinary annuity (payment at the end of a period):

$$FV_A = Pmt\left[\frac{(1+i)^N-1}{i}\right]$$

$$26973.46 = 1000\left[\frac{(1+0.01)^{24}-1}{0.01}\right]$$

Remarks

Related functions

The FVANNUITY( ) function is the inverse of the PVANNUITY( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PVLUMPSUM( ) function

Returns the present value required to generate a specific future lump sum calculated using a constant interest rate. Present value is the current, lump-sum value.

Syntax

PVLUMPSUM(*rate*, *periods*, *amount*)

Parameters

Name	Type	Description
rate	numeric	The interest rate per period.
periods	numeric	The total number of periods.
amount	numeric	The value of the future lump sum at the end of the last period.

> **Note**
>
> You must use consistent time periods when specifying *rate* and *periods* to ensure that you are specifying interest rate **per period**.
>
> For example:
>
> - for monthly payments on a two-year loan or investment with interest of 5% per annum, specify 0.05/12 for *rate* and 2 * 12 for *periods*
> - for annual payments on the same loan or investment, specify 0.05 for *rate* and 2 for *periods*

Output

Numeric. The result is calculated to two decimal places.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Interest compounded monthly

Returns 1000.00, the initial investment principal required to generate a future lump sum of $1,269.73, when invested for 2 years at 1% per month, compounded monthly:

```
PVLUMPSUM(0.01, 2*12, 1269.73)
```

Returns 787.57, the initial investment principal required to generate a future lump sum of $1,000, when invested for 2 years at 1% per month, compounded monthly:

```
PVLUMPSUM(0.01, 2*12, 1000)
```

Returns 21455.82, the initial investment principal required to generate a future lump sum of $27,243.20, when invested for 2 years at 1% per month, compounded monthly:

```
PVLUMPSUM(0.01, 2*12, 27243.20)
```

Interest compounded semi-annually

Returns 792.09, the initial investment principal required to generate a future lump sum of $1,000, when invested for 2 years at 12% per annum, compounded semi-annually:

```
PVLUMPSUM(0.12/2, 2*2, 1000)
```

Interest compounded annually

Returns 797.19, the initial investment principal required to generate a future lump sum of $1,000, when invested for 2 years at 12% per annum, compounded annually:

```
PVLUMPSUM(0.12, 2, 1000)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

What is present value?

The present value of an invested lump sum is the initial principal required to generate a specific future lump sum, within a particular time frame. The future value is the principal plus the accumulated compound interest.

Related functions

The FVLUMPSUM( ) function is the inverse of the PVLUMPSUM( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PYDATE( ) function

Returns a date value calculated by a function in a external Python script. Data processing in Python is external to Analytics.

Syntax

```
PYDATE("PyFile,PyFunction" <, field|value <,...n>>)
```

Parameters

Name	Type	Description
PyFile,PyFunction	character	The name of the Python script to run followed by a comma and then the name of the function that returns the value: `"myScript,myFunction"` When specifying the Python script, omit the file extension. The function you call may call other functions within the script or within other scripts, however all scripts that run must be placed inside a folder in the PYTHONPATH system environment variable prior to running. For more information, see "Install Python version 3.5.x (32-bit)" on page 924. **Note** Your *PyFunction* must return a Python datetime.date object.
field |value <,...n> optional	character numeric datetime logical	This list of fields, expressions, or literal values to use as arguments for the Python function. The values are passed into the function you call in the order you specify them. You may include as many arguments as necessary to satisfy the function definition in the Python script. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Output

Datetime.

Examples

Basic examples

Returns `20160630`:

```
PYDATE('hello,due_date', `20160531`, 30)
```

External Python script that accepts a date and a grace period as a number of days and calculates the date the invoice is due. For an invoice date of **2016-05-31** and a period of 30 days: "2016-06-30":

```
#! python
from datetime import timedelta

def due_date(inv_date, period):
   return(inv_date + timedelta(period))
```

Advanced examples

Defining a computed field

Defines a computed field in the `Ap_Trans` table using the Python script that calculates due date:

```
OPEN Ap_Trans
DEFINE FIELD due_date COMPUTED
WIDTH 27
   PYDATE( "hello,due_date" ,Invoice_Date, Pay_Period)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PYDATETIME( ) function

Returns a datetime value calculated by a function in an external Python script. Data processing in Python is external to Analytics.

Syntax

```
PYDATETIME("PyFile,PyFunction" <, field|value <,...n>>)
```

Parameters

Name	Type	Description
PyFile,PyFunction	character	The name of the Python script to run followed by a comma and then the name of the function that returns the value: `"myScript,myFunction"` When specifying the Python script, omit the file extension. The function you call may call other functions within the script or within other scripts, however all scripts that run must be placed inside a folder in the PYTHONPATH system environment variable prior to running. For more information, see "Install Python version 3.5.x (32-bit)" on page 924. **Note** Your *PyFunction* must return a Python datetime object.
field |value <,...n> optional	character numeric datetime logical	This list of fields, expressions, or literal values to use as arguments for the Python function. The values are passed into the function you call in the order you specify them. You may include as many arguments as necessary to satisfy the function definition in the Python script. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Output

Datetime.

Examples

Basic examples

Returns `20170101t0500`:

```
PYDATETIME("hello, combine_date_time", `20170101`, `t0500`)
```

External Python script that accepts a date argument and a time argument and returns a combined Datetime object:

```
# hello.py content
from datetime import datetime

def combine_date_time(d,t):
   return datetime.combine(d,t)
```

Advanced examples

Adding time to a datetime

Returns `20160101t2230`:

```
PYDATETIME("hello,add_time", `20160101 150000`, `t073000`)
```

External Python script that accepts a datetime and a time and adds the time to the datetime: 2016-01-01 15:00:00 + 7 hours, 30 minutes, 00 seconds = 2016-01-01 22:30:00.

```
# hello.py content
from datetime import timedelta
from datetime import datetime
from datetime import time
def add_time(start, time_to_add):
   return start + timedelta(hours=time_to_add.hour, minutes=time_to_add.minute, seconds=time_to_
add.second)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PYLOGICAL( ) function

Returns a logical value calculated by a function in an external Python script. Data processing in Python is external to Analytics.

Syntax

PYLOGICAL("*PyFile,PyFunction*" <, *field|value* <,...*n*>>)

Parameters

Name	Type	Description
PyFile,PyFunction	character	The name of the Python script to run followed by a comma and then the name of the function that returns the value: "myScript,myFunction" When specifying the Python script, omit the file extension. The function you call may call other functions within the script or within other scripts, however all scripts that run must be placed inside a folder in the PYTHONPATH system environment variable prior to running. For more information, see "Install Python version 3.5.x (32-bit)" on page 924. **Note** Your *PyFunction* must return a Python truth value.
field |value <,...n> optional	character numeric datetime logical	This list of fields, expressions, or literal values to use as arguments for the Python function. The values are passed into the function you call in the order you specify them. You may include as many arguments as necessary to satisfy the function definition in the Python script. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Output

Logical.

Examples

Basic examples

Returns F:

```
PYLOGICAL( "hello,str_compare", "basketball", "baseball", "b" )
```

External Python script that compares *str1* and *str2* using the count of the character that is passed in as *char*:

```
# hello.py content
def str_compare(str1, str2, char):
  return str1.count(char) > str2.count(char)
```

Advanced examples

Using fields

Returns a truth value when comparing Vendor_Name and Vendor_City:

```
PYLOGICAL( "hello,str_compare", Vendor_Name, Vendor_City, "b" )
```

External Python script that compares *str1* and *str2* using the count of the character that is passed in as *char*:

```
# hello.py content
def str_compare(str1, str2, char):
  return str1.count(char) > str2.count(char)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PYNUMERIC( ) function

Returns a numeric value calculated by a function in an external Python script. Data processing in Python is external to Analytics.

Syntax

```
PYNUMERIC(PyFile,PyFunction, decimal <, field|value <,...n>>)
```

Parameters

Name	Type	Description
PyFile,PyFunction	character	The name of the Python script to run followed by a comma and then the name of the function that returns the value: `"myScript,myFunction"` When specifying the Python script, omit the file extension. The function you call may call other functions within the script or within other scripts, however all scripts that run must be placed inside a folder in the PYTHONPATH system environment variable prior to running. For more information, see "Install Python version 3.5.x (32-bit)" on page 924. **Note** Your *PyFunction* must return a Python numeric type.
decimal	numeric	The number of decimal places to include in the return value. Must be a positive integer.
field |value <,...n> optional	character numeric datetime logical	This list of fields, expressions, or literal values to use as arguments for the Python function. The values are passed into the function you call in the order you specify them. You may include as many arguments as necessary to satisfy the function definition in the Python script. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Output

Numeric.

Examples

Basic examples

Returns 35.00:

```
PYNUMERIC("hello,get_nth_percent", 2, 80, 120, 30, 45, 30, 100, 35, 45)
```

External Python script that returns the value at the requested percentile from a dynamically sized list of values:

```
# hello.py content
from math import ceil
def get_nth_percent(percentage, *values):
   input_length = len(values)
   position = ceil((percentage/100.00) * input_length)
   return values[position-1]
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PYSTRING( ) function

Returns a character value calculated by a function in an external Python script. Data processing in Python is external to Analytics.

Syntax

```
PYSTRING("PyFile,PyFunction", length <,field|value <,...n>>)
```

Name	Type	Description
PyFile,PyFunction	character	The name of the Python script to run followed by a comma and then the name of the function that returns the value: `"myScript,myFunction"` When specifying the Python script, omit the file extension. The function you call may call other functions within the script or within other scripts, however all scripts that run must be placed inside a folder in the PYTHONPATH system environment variable prior to running. For more information, see "Install Python version 3.5.x (32-bit)" on page 924. **Note** Your *PyFunction* must return a Python string object.
length	numeric	The length to allocate for the return string.
field |value <,...n> optional	character numeric datetime logical	This list of fields, expressions, or literal values to use as arguments for the Python function. The values are passed into the function you call in the order you specify them. You may include as many arguments as necessary to satisfy the function definition in the Python script. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Output

Character.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Returns "my test":

```
PYSTRING('hello,main', 20, "my")
```

External Python script that accepts a string and concatenates " test" to the string:

```
#! python
# hello.py content
def main(str):
   str2 = str + ' test'
   return(str2)
```

Advanced examples

Returning a substring

This example removes the last two characters from the Vendor Name field and returns the substring:

```
PYSTRING( "hello,sub_set", LENGTH(Vendor_Name), ALLTRIM(Vendor_Name), LENGTH
(ALLTRIM(Vendor_Name)), 0, LENGTH(ALLTRIM(Vendor_Name)) - 2)
```

External Python script that accepts a string, a string length, and two character positions. The function returns a substring between position one and position two:

```
#hello.py content
def sub_set(str, length, p1, p2):
if p1 >= 0 and p2 < length and p1 < p2:
   str2 = str[p1:p2]
else:
   str2 = str
return str2
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PYTIME( ) function

Returns a time value calculated by a function in an external Python script. Data processing in Python is external to Analytics.

Syntax

PYTIME("*PyFile,PyFunction*" <, *field|value* <,...*n*>>)

Parameters

Name	Type	Description
PyFile,PyFunction	character	The name of the Python script to run followed by a comma and then the name of the function that returns the value: "myScript,myFunction" When specifying the Python script, omit the file extension. The function you call may call other functions within the script or within other scripts, however all scripts that run must be placed inside a folder in the PYTHONPATH system environment variable prior to running. For more information, see "Install Python version 3.5.x (32-bit)" on page 924. **Note** Your *PyFunction* must return a Python datetime.time object.
field |value <,...n> optional	character numeric datetime logical	This list of fields, expressions, or literal values to use as arguments for the Python function. The values are passed into the function you call in the order you specify them. You may include as many arguments as necessary to satisfy the function definition in the Python script. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Output

Datetime.

Examples

Basic examples

Returns `t2122`:

```
ASSIGN v_time_part = PYTIME("hello,get_time", `20160101 212223`)
```

External Python script:

```
# hello.py content
from datetime import time
from datetime import date

def get_time(timestamp):
  return timestamp.time();
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RAND( ) function

Returns a random number that falls within a specified boundary.

Syntax

```
RAND(number)
```

Parameters

Name	Type	Description

number	numeric	The numeric boundary for the random number.
		If you specify a number with decimal places the random number generated has the same number of decimal places.
		◦ **If you enter a positive number** - the random number returned is greater than or equal to zero, and less than the number you specified.
		Returns a number from 0 to 99:
		`RAND(100)`
		◦ **If you enter a negative number** - the random number returned is less than zero, and greater than or equal to the number you specified.
		Returns a number from -1 to -100:
		`RAND(-100)`

Output

Numeric.

Examples

Basic examples

Returns 278.61:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
RAND(1000.00)
```

Returns 3781:

```
RAND(10000)
```

> **Note**
> The return value will differ with each execution of the function.

Remarks

RAND( ) cannot replicate results

If you use the RAND( ) function consecutively with the same *number* value, it produces different results. Unlike the RANDOM command, the RAND( ) function has no seed value.

Duplicate random numbers possible

If you use RAND( ) to create a computed field that assigns a random number to every record in a table, it is possible that duplicate random numbers are generated. There is no guarantee that the random numbers will be unique.

The larger the *number* value in relation to the number of records in the table, the greater the chance that the generated numbers will be unique.

Random numbers dynamically update

A computed field with the RAND( ) function generates a new set of random numbers every time you perform actions such as quick sorting, applying a filter, rearranging columns, or scrolling through the view.

If you want to fix a set of random numbers, extract the data to a new table using the **View** or **Fields** option in the **Extract** dialog box.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RATE( ) function

Returns the interest rate per period.

Syntax

```
RATE(periods, payment, amount)
```

Parameters

Name	Type	Description
periods	numeric	The total number of payment periods.
payment	numeric	The payment per period.
amount	numeric	The principal amount of the loan.

> **Note**
> The RATE( ) function assumes that payments are made at the end of each period.

Output

Numeric. The rate is calculated to eight decimals places.

Examples

Basic examples

Returns 0.00541667 (0.54%), the monthly interest rate implied by a twenty-five year, $275,000 loan with monthly payments of $1,856.82:

```
RATE(12*25, 1856.82, 275000)
```

Returns 0.06500004 (6.5%), the annual interest rate implied by the same loan:

```
RATE(12*25, 1856.82, 275000)*12
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Advanced examples

Converting the nominal rate to the effective rate

The RATE( ) function calculates the nominal interest rate. You can use the EFFECTIVE( ) function to convert the result of RATE( ) to the effective interest rate.

Returns 0.06715155 (6.7%), the effective annual interest rate implied by the loan in the examples above:

```
EFFECTIVE((RATE(12*25, 1856.82, 275000)*12), 12*25)
```

Annuity calculations

Annuity calculations involve four variables:

- **present value, or future value** – $21,243.39 and $ 26,973.46 in the examples below
- **payment amount per period** – $1,000.00 in the examples below
- **interest rate per period** – 1% per month in the examples below
- **number of periods** – 24 months in the examples below

If you know the value of three of the variables, you can use an Analytics function to calculate the fourth.

I want to find:	Analytics function to use:
Present value	PVANNUITY( ) Returns 21243.39: `PVANNUITY(0.01, 24, 1000)`
Future value	FVANNUITY( ) Returns 26973.46: `FVANNUITY(0.01, 24, 1000)`
Payment amount per period	PMT( ) Returns 1000: `PMT(0.01, 24, 21243.39)`
Interest rate per period	RATE( ) Returns 0.00999999 (1%): `RATE(24, 1000, 21243.39)`
Number of periods	NPER( )

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

I want to find:	Analytics function to use:
	Returns 24.00: NPER(0.01, 1000, 21243.39)

Annuity formulas

The formula for calculating the **present value** of an ordinary annuity (payment at the end of a period):

$$PV_A = Pmt\left[\frac{1-\frac{1}{(1+i)^N}}{i}\right]$$

$$21243.39 = 1000\left[\frac{1-\frac{1}{(1+0.01)^{24}}}{0.01}\right]$$

The formula for calculating the **future value** of an ordinary annuity (payment at the end of a period):

$$FV_A = Pmt\left[\frac{(1+i)^N-1}{i}\right]$$

$$26973.46 = 1000\left[\frac{(1+0.01)^{24}-1}{0.01}\right]$$

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RDATE( ) function

Returns a date value calculated by an R function or script. Data processing in R is external to Analytics.

Syntax

RDATE(*rScript|rCode* <,*field|value* <,...*n*>>)

Parameters

Name	Type	Description
rScript | *rCode*	character	The full or relative path to the R script, or a snippet of R code to run. If you enter R code directly rather than use an external file, you cannot use the enclosing quotation character in your code, even if you escape it: ◦ **valid** - 'var <- "\"test\"" ' ◦ **invalid** - 'var <- "\'test\'" '
field | *value* <,...*n*> optional	character numeric datetime logical	The list of fields, expressions, or literal values to use as arguments for the R script or code snippet. The values are passed into the function you call in the order you specify them, and you reference them using value1, value2 ... value*N* in the R code. You may include as many arguments as necessary to satisfy the function definition in the R code. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Output

Datetime.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Returns `20160530`:

```
RDATE("as.Date(value1,'%m-%d-%Y')", "05-30-16")
```

Advanced examples

Using an external R script

Converts a string to a date and returns it:

```
RDATE("a<-source('c:\\scripts\\r_scripts\\sample.r');a[[1]]", dateText)
```

External R script (sample.r):

```
dateForm <- function(dateText) {
  return(as.Date(dateText,format='%y%m%d'))
}
dateForm(value1)
```

Remarks

Returning data from R

When calling R scripts, use the source function and assign the return object to a variable. You can then access the value returned from your R function from the return object:

```
# 'a' holds the response object and a[[1]] access the data value
"a<-source('c:\\scripts\\r_scripts\\sample.r');a[[1]]"
```

R log file

Analytics logs R language messages to an `aclrlang.log` file in the project folder. Use this log file for debugging R errors.

> **Tip**
>
> The log file is available in the Results folder of Analytics Exchange analytic jobs.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Running external R scripts on AX Server

If you are writing an analysis app to run on AX Server and you want to work with external R scripts:

1. Upload the file as a related file with the analysis app.
2. Use the FILE analytic tag to identify the file(s).
3. Reference the file(s) using the relative path `./filename.r`.

> **Note**
>
> Using a related file ensures that the TomEE application server account has sufficient permissions to access the file when running R with Analytics Exchange.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RDATETIME( ) function

Returns a datetime value calculated by an R function or script. Data processing in R is external to Analytics.

Syntax

```
RDATETIME(rScript|rCode <,field|value <,...n>>)
```

Parameters

Name	Type	Description
rScript | *rCode*	character	The full or relative path to the R script, or a snippet of R code to run. If you enter R code directly rather than use an external file, you cannot use the enclosing quotation character in your code, even if you escape it: ◦ **valid** - 'var <- "\"test\"" ' ◦ **invalid** - 'var <- "\'test\'" '
field | *value* <,...n> optional	character numeric datetime logical	The list of fields, expressions, or literal values to use as arguments for the R script or code snippet. The values are passed into the function you call in the order you specify them, and you reference them using value1, value2 ... value*N* in the R code. You may include as many arguments as necessary to satisfy the function definition in the R code. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Output

Datetime.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Adds 45 minutes to the current date and time:

```
RDATETIME("Sys.time() + value1",2700)
```

Advanced examples

Using an external R script

Adds 45 minutes to a datetime field by passing a field and a literal value to an external R function:

```
RDATETIME("a<-'c:\\scripts\\sample.r');a[[1]]", start_date, 2700)
```

External R script (sample.r):

```
add_time <- function(start, sec) {
  return(start + sec)
}
add_time(value1, value2)
```

Remarks

Returning data from R

When calling R scripts, use the source function and assign the return object to a variable. You can then access the value returned from your R function from the return object:

```
# 'a' holds the response object and a[[1]] access the data value
"a<-source('c:\\scripts\\r_scripts\\sample.r');a[[1]]"
```

R log file

Analytics logs R language messages to an `aclrlang.log` file in the project folder. Use this log file for debugging R errors.

> **Tip**
>
> The log file is available in the Results folder of Analytics Exchange analytic jobs.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Running external R scripts on AX Server

If you are writing an analysis app to run on AX Server and you want to work with external R scripts:

1. Upload the file as a related file with the analysis app.
2. Use the FILE analytic tag to identify the file(s).
3. Reference the file(s) using the relative path `./filename.r`.

> **Note**
>
> Using a related file ensures that the TomEE application server account has sufficient permissions to access the file when running R with Analytics Exchange.

System time zone

Greenwich Mean Time (GMT) is the default current time zone in the R environment used by Analytics.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RECLEN( ) function

Returns the length of the current record.

Syntax

```
RECLEN( )
```

Parameters

This function does not have any parameters.

Output

Numeric.

Examples

Basic examples

The following example extracts all records where the length is exactly 110:

```
EXTRACT RECORD IF RECLEN( ) = 110 TO "Extract.fil"
```

Remarks

You can use the RECLEN( ) function to identify to records of a particular length, or to test for shorter than expected records. This function is useful if you are working with Print Image (Report) files because it provides an easy way to examine the record lengths:

- For fixed-length records, the return value is a constant value (the record length).
- For variable-length records, the return value changes for each record.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RECNO( ) function

Returns the current record number.

Syntax

```
RECNO( )
```

Parameters

This function does not have any parameters.

Output

Numeric.

Examples

Basic examples

The following example extracts records numbered 10 through 20 to a new Analytics table:

```
EXTRACT RECORD IF BETWEEN(RECNO( ),10,20) TO "Subset.fil"
```

Remarks

You can use the RECNO( ) function to output record numbers to a table, or to determine the relative location of a particular record within a table.

Indexed tables vs Non-indexed tables

This function returns the current logical record number:

- If the table is not indexed, RECNO( ) starts with a value of 1 and increases by one for each record in the table. The logical and physical record numbers are identical.
- If the table is indexed, RECNO( ) behaves similarly, but counts the records using the logical, not physical order.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Using the SEEK or FIND command

If the SEEK or FIND commands are used, the record number is reset to 1 after you execute these commands.

Reordering records

When you reorder the records in a table, the record numbers generated by RECNO( ) are not reordered. To keep the record numbers with the records that they were originally associated with, extract the data to a new table using the **Fields** option before you reorder the records.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RECOFFSET( ) function

Returns a field value from a record that is a specified number of records from the current record.

Syntax

```
RECOFFSET(field, number_of_records)
```

Parameters

Name	Type	Description
field	character numeric datetime	The name of the field to retrieve the value from.
number_of_records	numeric	The number of records from the current record. A positive number specifies a record after the current record, and a negative number specifies a record before the current record.

Output

Character, Numeric, or Datetime. The return value belongs to the same data category as the input *field* parameter.

Examples

Basic examples

Returns an *Amount* value from the next record:

```
RECOFFSET(Amount,1)
```

Returns an *Amount* value from the previous record:

```
RECOFFSET(Amount, -1)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Advanced examples

Using RECOFFSET in a computed field

The computed field *Next_Amount* shows the value of the Amount field in the next record only if the next record has the same customer number.

To define this computed field in a script, use the following syntax:

```
DEFINE FIELD Next_Amount COMPUTED
RECOFFSET(Amount,1) IF RECOFFSET(Customer,1) = Customer
0
```

Next_Amount is the value of the next record's Amount field only if the customer number in the next record is the same as the customer number in the current record. Otherwise, *Next_Amount* is assigned a value of zero.

Remarks

The RECOFFSET( ) function returns a field value from a record that is a specified number of records above or below the current record.

When to use RECOFFSET( )

This function is commonly used for advanced comparison testing.

You can use this function to compare values in a field in the current record with a field in another record. For example, you can add a computed field that calculates the difference between an amount in the current record and an amount in the previous record.

The beginning or end of a table

If the beginning or end of the table is encountered, the function returns zero for numeric fields, a blank string for character fields, or 1900/01/01 for date fields. The function returns blank output in these instances because there is no further record to compare the current record to.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

REGEXFIND( ) function

Returns a logical value indicating whether the pattern specified by a regular expression occurs in a string.

Syntax

```
REGEXFIND(string, pattern)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to test for a matching pattern.
pattern	character	The pattern string (regular expression) to search for. *pattern* can contain literal characters, metacharacters, or a combination of the two. Literal characters include all alphanumeric characters, some punctuation characters, and blanks. The search is case-sensitive, which means that uppercase and lowercase alpha characters must be explicitly specified.

Output

Logical. Returns **T** (true) if the specified *pattern* value is found, and **F** (false) otherwise.

Examples

Basic examples

Alpha character patterns

Returns T for all records that contain "Phoenix", "Austin", or "Los Angeles" in the **Vendor_City** field. Returns F otherwise:

```
REGEXFIND(Vendor_City, "Phoenix|Austin|Los Angeles")
```

Returns T for all last names that start with "John" or "Jon". For example: John, Jon, Johnson, Johnston, Jonson, Jonston, Jones, and so on. Returns F otherwise:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
REGEXFIND(Last_Name,"^Joh?n")
```

Returns T for only those last names that are "John" or "Jon". Returns F otherwise:

```
REGEXFIND(Last_Name,"^Joh?n\b")
```

Numeric character patterns

Returns T for all records with invoice numbers that contain "98". Returns F otherwise:

```
REGEXFIND(Invoice_Number, "98")
```

Returns T for all records with invoice numbers that begin with "98". Returns F otherwise:

```
REGEXFIND(Invoice_Number, "\b98")
```

Returns T for all records with invoice numbers that end with "98". Returns F otherwise:

```
REGEXFIND(Invoice_Number, "98\b")
```

Returns T for all records with invoice numbers that contain "98" in the 5th and 6th positions. Returns F otherwise:

```
REGEXFIND(Invoice_Number, "\b\d\d\d\d98")
```

```
REGEXFIND(Invoice_Number, "\b\d{4}98")
```

Mixed character patterns

Returns T for all records with product codes that start with 3 numbers, followed by a hyphen and 6 letters. Returns F otherwise:

```
REGEXFIND(Product_Code, "\b\d{3}-[a-zA-Z]{6}\b")
```

Returns T for all records with product codes that start with 3 or more numbers, followed by a hyphen and 6 or more letters. Returns F otherwise:

```
REGEXFIND(Product_Code, "\b\d{3,}-[a-zA-Z]{6}")
```

Returns T for all records with alphanumeric invoice identifiers that contain "98" in the 5th and 6th positions. Returns F otherwise:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
REGEXFIND(Invoice_Number, "\b\w{4}98")
```

Returns T for all records with invoice identifiers that contain both of the following, otherwise returns F:

- any character in the first four positions
- "98" in the 5th and 6th positions

```
REGEXFIND(Invoice_Number, "\b.{4}98")
```

Returns T for all records with invoice identifiers that contain "98" preceded by 1 to 4 initial characters. Returns F otherwise:

```
REGEXFIND(Invoice_Number, "\b.{1,4}98")
```

Returns 'T' for all records with invoice identifiers that contain all of the following, otherwise returns F:

- any character in the first three positions
- "5" or "6" in the 4th position
- "98" in the 5th and 6th positions

```
REGEXFIND(Invoice_Number, "\b.{3}[56]98")
```

Returns T for all records with invoice identifiers that contain all of the following, otherwise returns F:

- any character in the first two positions
- "55" or "56" in the 3rd and 4th positions
- "98" in the 5th and 6th positions

```
REGEXFIND(Invoice_Number, "\b.{2}(55|56)98")
```

Remarks

How it works

The REGEXFIND( ) function uses a regular expression to search data in Analytics.

Regular expressions are powerful and flexible search strings that combine literal characters and metacharacters, which are special characters that perform a wide variety of search operations.

For example:

```
REGEXFIND(Last_Name,"Sm(i|y)the{0,1}")
```

uses the group ( ) , alternation | , and quantifier { } metacharacters to create a regular expression that finds "Smith", "Smyth", "Smithe", or "Smythe" in the **Last_Name** field.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Matching performed sequentially

Matching between *string* and *pattern* values is performed sequentially. In the example above:

- "S" is matched against the first position in the **Last_Name** field
- "m" is matched against the second position
- "i" and "y" are matched against the third position
- "t" is matched against the fourth position
- "h" is matched against the fifth position
- "e" is matched against the sixth position, if a sixth position exists in the source value

When to use REGEXFIND( )

Use REGEXFIND( ) to search data using simple or complex pattern matching.

Constructing regular expressions can be tricky, especially if you are new to the syntax. You may be able to achieve your search goals using simpler Analytics search functions such as FIND( ), MATCH( ), or MAP( ).

If your search requirements exceed the capabilities of these simpler functions, regular expressions give you almost unlimited flexibility in constructing search strings.

How REGEXFIND( ) handles spaces

Spaces (blanks) are treated as characters in both *string* and *pattern*, so you should exercise care when dealing with spaces.

In *pattern*, you can indicate a space either literally, by typing a space, or by using the metacharacter \s. Using the metacharacter makes spaces easier to read in *pattern*, and therefore less likely to be overlooked, especially when you construct more complex patterns.

Concatenating fields

You can concatenate two or more fields in *string* if you want to search across multiple fields simultaneously.

For example:

```
REGEXFIND(Vendor_Name+Vendor_Street,"Hardware.*Main")
```

searches both the **Vendor_Name** and the **Vendor_Street** fields for the words "Hardware" and "Main" separated by zero or more characters.

A business with the word "Hardware" in its name, located on a street called "Main", matches the regular expression. So does a business called "Hardware on Main".

The concatenated fields are treated like a single field that includes leading and trailing spaces from the individual fields, unless you use the ALLTRIM( ) function to remove spaces.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Order of concatenated fields matters

Because REGEXFIND( ) searches for the characters in *pattern* in the order in which you specify them, the order in which you concatenate the fields has an effect. If you reversed **Vendor_Name** and **Vendor_Street** in the expression above, you would be less likely to get any results.

Regular expression metacharacters

The table below lists metacharacters you can use with REGEXFIND( ) and REGEXREPLACE( ) and describes the operation that each one performs.

Additional regular expression syntax exists, and is supported by Analytics, but it is more complex. A full explanation of additional syntax is beyond the scope of this guide. Numerous resources explaining regular expressions are available on the Internet.

Analytics uses the **ECMAScript** implementation of regular expressions. Most implementations of regular expressions use a common core syntax.

> **Note**
>
> The current implementation of regular expressions in Analytics does not fully support searching languages other than English.

Metacharacter	Description
.	Matches any character (except a new line character)
?	Matches 0 or 1 occurrences of the immediately preceding literal, metacharacter, or element
*****	Matches 0 or more occurrences of the immediately preceding literal, metacharacter, or element
+	Matches 1 or more occurrences of the immediately preceding literal, metacharacter, or element
{}	Matches the specified number of occurrences of the immediately preceding literal, metacharacter, or element. You can specify an exact number, a range, or an open-ended range. For example: ◦ **a{3}** matches "aaa" ◦ **X{0,2}L** matches "L", "XL", and "XXL" ◦ **AB-\d{2,}-YZ** matches any alphanumeric identifier with the prefix "AB-", the suffix "-YZ", and two or more numbers in the body of the identifier
[]	Matches any single character inside the brackets For example: ◦ **[aeiou]** matches a, or e, or i, or o, or u ◦ **[^aeiou]** matches any character that is not a, or e, or i, or o, or u ◦ **[A-G]** matches any uppercase letter from A to G ◦ **[A-Ga-g]** matches any uppercase letter from A to G, or any lowercase letter from a to g ◦ **[5-9]** matches any number from 5 to 9

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Metacharacter	Description
()	Creates a group that defines a sequence or block of characters, which can then be treated as a single unit. For example: ◦ **S(ch)?mid?th?** matches "Smith" or "Schmidt" ◦ **(56A.*){2}** matches any alphanumeric identifier in which the sequence "56A" occurs at least twice ◦ **(56A).*-.*\1** matches any alphanumeric identifier in which the sequence "56A" occurs at least twice, with a hyphen located between two of the occurrences
****	An escape character that specifies that the character immediately following is a literal. Use the escape character if you want to literally match metacharacters. For example, **\(** finds a left parenthesis, and **** finds a backslash. Use the escape character if you want to literally match any of the following characters: **^ $. * + ? = ! : | \ ( ) [ ] { }** Other punctuation characters such as the ampersand (&) or the 'at sign' (@) do not require the escape character.
int	Specifies that a group, previously defined with parentheses **( )**, recurs. *int* is an integer that identifies the sequential position of the previously defined group in relation to any other groups. This metacharacter can be used in the *pattern* parameter in both REGEXFIND( ) and REGEXREPLACE ( ). For example: ◦ **(123).*\1** matches any identifier in which the group of digits "123" occurs at least twice ◦ **^(\d{3}).*\1** matches any identifier in which the first 3 digits recur ◦ **^(\d{3}).*\1.*\1** matches any identifier in which the first 3 digits recur at least twice ◦ **^(\D)(\d)-.*\2\1** matches any identifier in which the alphanumeric prefix recurs with the alpha and numeric characters reversed
$*int*	Specifies that a group found in a target string is used in a replacement string. *int* is an integer that identifies the sequential position of the group in the target string in relation to any other groups. This metacharacter can only be used in the *new_string* parameter in REGEXREPLACE( ). For example: ◦ If the pattern **(\d{3})[-]?(\d{3})[-]?(\d{4})** is used to match a variety of different telephone number formats, the *new_string***($1)-$2-$3** can be used to replace the numbers with themselves, and standardize the formatting. 999 123-4567 and 9991234567 both become (999)-123-4567.
|	Matches the character, block of characters, or expression before or after the pipe (|) For example: ◦ **a|b** matches a or b ◦ **abc|def** matches "abc" or "def" ◦ **Sm(i|y)th** matches Smith or Smyth ◦ **[a-c]|[Q-S]|[x-z]** matches any of the following letters: a, b, c, Q, R, S, x, y, z

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Metacharacter	Description
	◦ **\s|-** matches a space or a hyphen
\w	Matches any word character (a to z, A to Z, 0 to 9, and the underscore character _)
\W	Matches any non-word character (not a to z, A to Z, 0 to 9, or the underscore character _)
\d	Matches any number (any decimal digit)
\D	Matches any non-number (any character that is not a decimal digit)
\s	Matches a space (a blank)
\S	Matches any non-space (a non-blank character)
\b	Matches a word boundary (between \w and \W characters) Word boundaries consume no space themselves. For example: ◦ "United Equipment" contains 4 word boundaries - one either side of the space, and one at the start and the end of the string. "United Equipment" is matched by the regular expression **\b\w*\b\W\b\w*\b** **Tip** In addition to spaces, word boundaries can result from commas, periods, and other non-word characters. For example, the following expression evaluates to True: `REGEXFIND("jsmith@example.net", "\bexample\b")`
^	Matches the start of a string Inside brackets **[ ]**, **^** negates the contents
$	Matches the end of a string

Related functions

If you want to find and replace matching patterns, use the "REGEXREPLACE( ) function" on the facing page.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

REGEXREPLACE( ) function

Replaces all instances of strings matching a regular expression with a new string.

Syntax

```
REGEXREPLACE(string, pattern, new_string)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to test for a matching pattern.
pattern	character	The pattern string (regular expression) to search for. *pattern* can contain literal characters, metacharacters, or a combination of the two. Literal characters include all alphanumeric characters, some punctuation characters, and blanks. The search is case-sensitive, which means that uppercase and lowercase alpha characters must be explicitly specified.
new_string	character	The string to use to replace all values matching *pattern*. The replacement string can contain literal characters, groups of characters from the original string (using the $*int* element), or a combination of the two.

Output

Character.

Examples

Basic examples

Working with spaces

Returns "AB CD EF", by replacing multiple spaces between text characters with a single space:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
REGEXREPLACE("AB CD  EF", "\s+", " ")
```

Returns the character field data with the spacing between words standardized on a single space:

```
REGEXREPLACE(character_field, "\s+", " ")
```

Returns the character field data with the spacing between words standardized on a single space. Using the BLANKS( ) function in *new_string*, rather than a literal space, makes spaces easier to read and less likely to be overlooked:

```
REGEXREPLACE(character_field, "\s+", BLANKS(1))
```

Standardizing telephone numbers

Returns "(123) 456-7890". The formatting of the telephone number '1234567890' is standardized:

```
REGEXREPLACE(SUBSTR("1234567890",1,14), "(\d{3})[\s-]*(\d{3})[\s-]*(\d{4})","($1) $2-$3")
```

Returns the numbers in the **Telephone_Number** field and standardizes their formatting:

```
REGEXREPLACE(Telephone_Number, ".*(\d{3})[\s-\.\)]*(\d{3})[\s-\.]*(\d{4})", "($1) $2-$3")
```

Extracts "123-456-7890" from the surrounding text:

```
REGEXREPLACE("Tel num: 123-456-7890 (office)", "(.*)(\d{3}[\s-\)\.]*\d{3}[\s-\.]*\d{4})(.*)", "$2")
```

Extracts telephone numbers from surrounding text in the **Comment** field and standardizes their formatting:

```
REGEXREPLACE(Comment, "(.*)(\d{3})[\s-\)\.]*(\d{3})[\s-\.]*(\d{4})(.*)","($2) $3-$4")
```

Identifying generic formats

Returns "9XXX-999xx", which represents the generic format of the value specified by *string* ("1ABC-123aa"):

```
REGEXREPLACE(REGEXREPLACE(REGEXREPLACE("1ABC-123aa","\d","9"),"[a-z]","x"),"[A-Z]", "X")
```

Returns the generic format of all identifiers in the **Invoice_Number** field:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
REGEXREPLACE(REGEXREPLACE(REGEXREPLACE(Invoice_Number,"\d","9"),"[a-z]","x"),"[A-Z]", "X")
```

Standardizing name format

Returns "John David Smith":

```
REGEXREPLACE("Smith, John David", "^(\w+),(\s\w+)(\s\w+)?(\s\w+)?","$2$3$4 $1")
```

Returns the names in the **Full_Name** field in their regular order: *First* (*Middle*) (*Middle*) *Last*:

```
REGEXREPLACE(Full_Name, "^(\w+),(\s\w+)(\s\w+)?(\s\w+)?","$2$3$4 $1")
```

> **Note**
>
> Name data can present various complications, such as apostrophes in names. Accounting for variations in name data typically requires more complex regular expressions than the one provided in the example above.

Removing HTML markup

Returns "https://www.flgov.com/wp-content/uploads/orders/2020/EO_20-166.pdf":

```
REGEXREPLACE("<a href='https://www.flgov.com/wp-content/uploads/orders/2020/EO_20-166.pdf' target='blank'>https://www.flgov.com/wp-content/uploads/orders/2020/EO_20-166.pdf</a>", "<[^>]*>",' ')
```

Returns the hyperlinks in the **Source_URL_Link** field with the HTML markup removed:

```
REGEXREPLACE(Source_URL_Link, "<[^>]*>",' ')
```

Remarks

How it works

The REGEXREPLACE( ) function uses a regular expression to find matching patterns in data, and replaces any matching values with a new string.

For example:

```
REGEXREPLACE(character_field, "\s+", " ")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

standardizes spacing in character data by replacing one or more spaces between text characters with a single space.

The search portion of REGEXREPLACE( ) is identical to the REGEXFIND( ) function. For detailed information about the search capability common to both functions, see "REGEXFIND( ) function" on page 736.

When to use REGEXREPLACE( )

Use REGEXREPLACE( ) any time you want to find and replace data in Analytics using simple or complex pattern matching.

Replacing characters with themselves

You can use the $*int* element to replace characters with themselves, which allows you to preserve the meaningful parts of data, while standardizing or omitting surrounding or intermixed data.

Several examples using telephone numbers and names appear above.

To use the $*int* element you must first create groups by using parentheses ( ) in the *pattern* value. For more information, see "REGEXFIND( ) function" on page 736.

Avoiding sequential character matching

You can avoid sequential character matching, and replace substrings regardless of their position in relation to one another, by nesting REGEXREPLACE( ) functions.

The problem in the two examples below is to derive a generic format from alphanumeric source data in which numbers and letters can appear in any order. Without knowing the order of numbers and letters, how do you construct the *pattern* string?

The solution is to first find and replace numbers using the inner REGEXREPLACE( ) function, and then find and replace letters using the outer REGEXREPLACE( ) function.

Returns "999XXX":

```
REGEXREPLACE(REGEXREPLACE("123ABC","\d","9"),"[A-Z]","X")
```

Returns "9X9X9X":

```
REGEXREPLACE(REGEXREPLACE("1A2B3C","\d","9"),"[A-Z]","X")
```

Replacement string length and truncation

When you use REGEXREPLACE( ) to create a computed field, the computed field length is identical to the original field length.

If the replacement string length exceeds the target string length, overall string length increases, which results in truncation if the computed field length cannot accommodate the increased string length.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Characters that trail the target string are truncated first, followed by trailing replacement string characters. The examples below illustrate truncation:

string	*pattern*	*new_string*	Field length	Result	Truncated characters
x123x	123	A	5	"xAx"	none
x123x	123	ABC	5	"xABCx"	none
x123x	123	ABCD	5	"xABCD"	"x"
x123x	123	ABCDE	5	"xABCD"	"x", "E"
x123x	123	ABCDE	6	"xABCDE"	"x"
x123x	123	ABCDE	7	"xABCDEx"	none

How to avoid truncation

To avoid truncation, use the SUBSTR( ) function to increase field length, as demonstrated by the second example below.

Returns "xABCD", which truncates the replacement character "E" and the existing character "x":

```
REGEXREPLACE("x123x","123","ABCDE")
```

Returns "xABCDEx", which includes all replacement characters and unreplaced existing characters:

```
REGEXREPLACE(SUBSTR("x123x",1,10),"123","ABCDE")
```

Regular expression metacharacters

The table below lists metacharacters you can use with REGEXFIND( ) and REGEXREPLACE( ) and describes the operation that each one performs.

Additional regular expression syntax exists, and is supported by Analytics, but it is more complex. A full explanation of additional syntax is beyond the scope of this guide. Numerous resources explaining regular expressions are available on the Internet.

Analytics uses the **ECMAScript** implementation of regular expressions. Most implementations of regular expressions use a common core syntax.

> **Note**
>
> The current implementation of regular expressions in Analytics does not fully support searching languages other than English.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Metacharacter	Description
.	Matches any character (except a new line character)
?	Matches 0 or 1 occurrences of the immediately preceding literal, metacharacter, or element
*	Matches 0 or more occurrences of the immediately preceding literal, metacharacter, or element
+	Matches 1 or more occurrences of the immediately preceding literal, metacharacter, or element
{}	Matches the specified number of occurrences of the immediately preceding literal, metacharacter, or element. You can specify an exact number, a range, or an open-ended range. For example: ◦ **a{3}** matches "aaa" ◦ **X{0,2}L** matches "L", "XL", and "XXL" ◦ **AB-\d{2,}-YZ** matches any alphanumeric identifier with the prefix "AB-", the suffix "-YZ", and two or more numbers in the body of the identifier
[]	Matches any single character inside the brackets For example: ◦ **[aeiou]** matches a, or e, or i, or o, or u ◦ **[^aeiou]** matches any character that is not a, or e, or i, or o, or u ◦ **[A-G]** matches any uppercase letter from A to G ◦ **[A-Ga-g]** matches any uppercase letter from A to G, or any lowercase letter from a to g ◦ **[5-9]** matches any number from 5 to 9
()	Creates a group that defines a sequence or block of characters, which can then be treated as a single unit. For example: ◦ **S(ch)?mid?th?** matches "Smith" or "Schmidt" ◦ **(56A.*){2}** matches any alphanumeric identifier in which the sequence "56A" occurs at least twice ◦ **(56A).*-.*\1** matches any alphanumeric identifier in which the sequence "56A" occurs at least twice, with a hyphen located between two of the occurrences
\	An escape character that specifies that the character immediately following is a literal. Use the escape character if you want to literally match metacharacters. For example, **\(** finds a left parenthesis, and **** finds a backslash. Use the escape character if you want to literally match any of the following characters: **^ $. * + ? = ! : | \ ( ) [ ] { }** Other punctuation characters such as the ampersand (&) or the 'at sign' (@) do not require the escape character.
int	Specifies that a group, previously defined with parentheses **( )**, recurs. *int* is an integer that identifies

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Metacharacter	Description
	the sequential position of the previously defined group in relation to any other groups. This metacharacter can be used in the *pattern* parameter in both REGEXFIND( ) and REGEXREPLACE ( ). For example: ◦ **(123).*\1** matches any identifier in which the group of digits "123" occurs at least twice ◦ **^(\d{3}).*\1** matches any identifier in which the first 3 digits recur ◦ **^(\d{3}).*\1.*\1** matches any identifier in which the first 3 digits recur at least twice ◦ **^(\D)(\d)-.*\2\1** matches any identifier in which the alphanumeric prefix recurs with the alpha and numeric characters reversed
$*int*	Specifies that a group found in a target string is used in a replacement string. *int* is an integer that identifies the sequential position of the group in the target string in relation to any other groups. This metacharacter can only be used in the *new_string* parameter in REGEXREPLACE( ). For example: ◦ If the pattern **(\d{3})[-]?(\d{3})[-]?(\d{4})** is used to match a variety of different telephone number formats, the *new_string***($1)-$2-$3** can be used to replace the numbers with themselves, and standardize the formatting. 999 123-4567 and 9991234567 both become (999)-123-4567.
|	Matches the character, block of characters, or expression before or after the pipe (**|**) For example: ◦ **a|b** matches a or b ◦ **abc|def** matches "abc" or "def" ◦ **Sm(i|y)th** matches Smith or Smyth ◦ **[a-c]|[Q-S]|[x-z]** matches any of the following letters: a, b, c, Q, R, S, x, y, z ◦ **\s|-** matches a space or a hyphen
\w	Matches any word character (a to z, A to Z, 0 to 9, and the underscore character _)
\W	Matches any non-word character (not a to z, A to Z, 0 to 9, or the underscore character _)
\d	Matches any number (any decimal digit)
\D	Matches any non-number (any character that is not a decimal digit)
\s	Matches a space (a blank)
\S	Matches any non-space (a non-blank character)
\b	Matches a word boundary (between \w and \W characters) Word boundaries consume no space themselves. For example: ◦ "United Equipment" contains 4 word boundaries - one either side of the space, and one at the start and the end of the string. "United Equipment" is matched by the regular expression **\b\w*\b\W\b\w*\b**

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Metacharacter	Description
	Tip In addition to spaces, word boundaries can result from commas, periods, and other non-word characters. For example, the following expression evaluates to True: `REGEXFIND("jsmith@example.net", "\bexample\b")`
^	Matches the start of a string Inside brackets **[ ]**, **^** negates the contents
$	Matches the end of a string

Related functions

If you want to find matching patterns without replacing them, use the "REGEXFIND( ) function" on page 736.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

REMOVE( ) function

Returns a string that includes only the specified characters.

Syntax

```
REMOVE(string, valid_characters)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to remove characters from.
valid_characters	character	The characters to retain in *string*. If you specify double quotation marks in *valid_characters*, you must enclose the list of characters in single quotation marks. For example: `'"-/'` **Note** If a character you specify does not appear in *string*, it is not included in the return value.

Output

Character.

Examples

Basic examples

Returns "ABC123 ":

```
REMOVE("ABC 123 XX4","ABC123")
```

Returns "ABC123XX ":

```
REMOVE("zABC 123 XX4","ABCX123")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns "1234 ":

```
REMOVE("ABC 123 XX4", "1234567890")
```

Returns all the values in the **Product_Number** field with all non-numeric characters removed:

```
REMOVE(Product_Number,"0123456789")
```

Remarks

> **Note**
>
> The REMOVE( ) function has been superseded by the INCLUDE( ) and EXCLUDE( ) functions.
>
> REMOVE( ) is still available in the current version of Analytics for backwards compatibility with earlier versions.

How it works

The REMOVE( ) function removes unwanted characters from character data and returns a fixed length string.

When to use REMOVE( )

Use REMOVE( ) to normalize data fields that do not have a consistent format, such as address fields. You can also use REMOVE( ) to remove punctuation or other invalid information from poorly edited fields.

You can also use the function to clean data in fields before using the SORT or JOIN commands, for duplicate matching, or for report output.

Case sensitivity

The REMOVE( ) function is case-sensitive. If you specify "ID" in *valid_characters*, these characters are not included in "id#94022". If there is a chance the case may be mixed, use the UPPER( ) function to convert *string* to uppercase.

For example:

```
REMOVE(UPPER("id#94022"), "ID0123456789")
```

Related functions

REMOVE( ) is similar to the INCLUDE( ) function, with the following difference:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- REMOVE( ) adds blanks to the end of the output to replace the characters that have been removed. The original length of *string* is retained.
- INCLUDE( ) does not add any blanks.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

REPEAT( ) function

Returns a string that repeats a substring a specified number of times.

Syntax

```
REPEAT(string, count)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to repeat.
count	numeric	The number of times to repeat the value of *string*.

Output

Character.

Examples

Basic examples

Returns "ABCABCABC":

```
REPEAT("ABC",3)
```

Returns "000000000":

```
REPEAT("0",9)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

When to use REPEAT( )

Use the REPEAT( ) function to initialize a variable with constant values or blanks, or to set a default value for a computed field.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

REPLACE( ) function

Replaces all instances of a specified character string with a new character string.

Syntax

```
REPLACE(string, old_text, new_text)
```

Parameters

Name	Type	Description
string	character	The value to replace characters in.
old_text	character	The character string to replace. The search is case-sensitive.
new_text	character	The text to replace the value in *old_text* with.

Output

Character.

Examples

Basic examples

Returns "a12345efg":

```
REPLACE("abcdefg","bcd","12345")
```

Returns "Rd.":

```
REPLACE("Road","Road","Rd.")
```

Returns "ac":

```
REPLACE("abc","b","")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Advanced examples

Removing specified characters

Use REPLACE( ) to remove a specified character string from a source string, by replacing it with an empty character string ("").

Returns "1234 Scott":

```
REPLACE("1234 Scott rd.", "rd.", "")
```

Field length adjustment

If *new_text* ("ABC") is longer than *old_text* ("X"), the field length of the resulting string is automatically increased to accommodate the first replacement:

Returns "9ABC9", with a field length increased to 5 characters from 3 characters:

```
REPLACE("9X9", "X", "ABC")
```

The field length is not automatically increased for subsequent replacements, and truncation can result if the field is not long enough to accommodate all the new characters.

Returns "9ABC9A":

```
REPLACE("9X9X", "X", "ABC")
```

To avoid truncation, you can increase the length of *string* using the BLANKS( ) function, or literal blank spaces.

Returns "9ABC9ABC":

```
REPLACE("9X9X" + BLANKS(2), "X", "ABC")
```

```
REPLACE("9X9X" + "  ", "X", "ABC")
```

If the resulting string is shorter than *string*, the resulting string is padded with blanks to maintain the same field length.

Returns "9X9 ":

```
REPLACE("9ABC9", "ABC", "X")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

How it works

The REPLACE( ) function replaces every instance of an existing string with a new string.

Returns "1234 Scott Road":

```
REPLACE("1234 Scott rd.", "rd.", "Road")
```

When to use REPLACE( )

Use REPLACE( ) for normalizing data fields with inconsistent formats, such as address fields, or for replacing invalid information in poorly edited fields. To be performed accurately, operations such as duplicates testing, or joining or relating tables, require data with a normalized or standardized format.

Case sensitivity

The REPLACE( ) function is case-sensitive. If you specify `"RD."` in *old_text* and the values in *string* are lowercase, the *new_text* value will not be substituted because no matches will be found.

If there is a chance the case in *string* may be mixed, first use the UPPER( ) function to convert all characters to uppercase.

Returns "1234 SCOTT ROAD":

```
REPLACE(UPPER("1234 Scott rd."), "RD.", "ROAD")
```

Protecting against inadvertent replacements

When building a REPLACE( ) expression you must be aware of every possible instance of *old_text* in *string* so that you do not get inadvertent replacements.

Returns "645 RichaRoad Road", because the last two letters of "Richard" are "rd":

```
REPLACE("645 Richard rd  ", "rd", "Road")
```

By adding both a leading space and a trailing space to the value in *old_text* (`" rd "`), you prevent the function from replacing instances where "rd" occurs in a name, because in these instances there are no leading spaces.

Returns "645 Richard Road":

```
REPLACE("645 Richard rd  ", " rd ", " Road")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

REVERSE( ) function

Returns a string with the characters in reverse order.

Syntax

```
REVERSE(string)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to reverse the order of.

Output

Character.

Examples

Basic examples

Returns "E DCBA":

```
REVERSE("ABCD E")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RJUSTIFY( ) function

Returns a right-justified string the same length as a specified string, with any trailing spaces moved to the left of the string.

Syntax

```
RJUSTIFY(string)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to right-justify.

Output

Character.

Examples

Basic examples

Returns " ABC":

```
RJUSTIFY("ABC  ")
```

Remarks

When to use RJUSTIFY( )

Use the RJUSTIFY( ) function to right-justify a character field.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RLOGICAL( ) function

Returns a logical value calculated by an R function or script. Data processing in R is external to Analytics.

Syntax

RLOGICAL(*rScript|rCode* <,*field|value* <,...*n*>>)

Parameters

Name	Type	Description
rScript | *rCode*	character	The full or relative path to the R script, or a snippet of R code to run. If you enter R code directly rather than use an external file, you cannot use the enclosing quotation character in your code, even if you escape it: ◦ **valid** - 'var <- "\"test\"" ' ◦ **invalid** - 'var <- "\'test\'" '
field | *value* <,...*n*> optional	character numeric datetime logical	The list of fields, expressions, or literal values to use as arguments for the R script or code snippet. The values are passed into the function you call in the order you specify them, and you reference them using value1, value2 ... value*N* in the R code. You may include as many arguments as necessary to satisfy the function definition in the R code. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Output

Logical.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Returns T:

```
RLOGICAL("(value1>0.6) & (value2>0.7) & (value3>0.5)", 0.8, 0.9, 0.55)
```

Advanced examples

Using an external R script

Accepts an amount, and an upper and lower threshold value. The function returns a truth value based on a series of logical comparisons:

```
RLOGICAL("a<-'c:\\scripts\\sample.r');a[[1]]", expense_amt, threshold_low, threshold_hi)
```

External R script (sample.r):

```
test_truth <- function(amt, low, hi) {
   return(((amt > low) & (amt < hi)) | ((amt==low) | (amt==hi)))
}
test_truth(value1, value2, value3)
```

Using R code stored in a variable

Performs a logical test of three fields using AND logic:

```
v_rcode = "(value1>0.6) & (value2>0.7) & (value3>0.5)"
RLOGICAL(v_rcode, PACKED, MICRO_LONG, ACCPAC)
```

Remarks

Returning data from R

When calling R scripts, use the source function and assign the return object to a variable. You can then access the value returned from your R function from the return object:

```
# 'a' holds the response object and a[[1]] access the data value
"a<-source('c:\\scripts\\r_scripts\\sample.r');a[[1]]"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

R log file

Analytics logs R language messages to an `aclrlang.log` file in the project folder. Use this log file for debugging R errors.

> **Tip**
>
> The log file is available in the Results folder of Analytics Exchange analytic jobs.

Running external R scripts on AX Server

If you are writing an analysis app to run on AX Server and you want to work with external R scripts:

1. Upload the file as a related file with the analysis app.
2. Use the FILE analytic tag to identify the file(s).
3. Reference the file(s) using the relative path `./filename.r`.

> **Note**
>
> Using a related file ensures that the TomEE application server account has sufficient permissions to access the file when running R with Analytics Exchange.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RNUMERIC( ) function

Returns a numeric value calculated by an R function or script. Data processing in R is external to Analytics.

Syntax

RNUMERIC(*rScript|rCode*, *decimals* <,*field|value* <,...*n*>>)

Parameters

Name	Type	Description
rScript | *rCode*	character	The full or relative path to the R script, or a snippet of R code to run. If you enter R code directly rather than use an external file, you cannot use the enclosing quotation character in your code, even if you escape it: ◦ **valid** - 'var <- "\"test\"" ' ◦ **invalid** - 'var <- "\'test\'" '
decimals	numeric	The number of decimal places to include in the return value. Must be a positive integer.
field | *value* <,...*n*> optional	character numeric datetime logical	The list of fields, expressions, or literal values to use as arguments for the R script or code snippet. The values are passed into the function you call in the order you specify them, and you reference them using value1, value2 ... value*N* in the R code. You may include as many arguments as necessary to satisfy the function definition in the R code. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Output

Numeric.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Returns 100 with 10 decimals (100.0000000000):

```
RNUMERIC("print(value1)", 10, 100)
```

Advanced examples

Storing R code as a variable

Returns 100 with 10 decimals (100.0000000000):

```
ASSIGN v_rcode = "print(value1)"
RNUMERIC(v_rcode, 10, 100)
```

Writing to an external file

Performs a simple addition and writes the comment attached to the function to file using the `sink` function in R:

```
RNUMERIC("foo<-function(x,y){x+y};attr(foo, 'comment') <- 'foo performs simple addition';sink('c:/temp/result.txt');attributes(foo);sink(NULL);foo(value1,value2)",0, amt, gross)
```

Remarks

Returning data from R

When calling R scripts, use the `source` function and assign the return object to a variable. You can then access the value returned from your R function from the return object:

```
# 'a' holds the response object and a[[1]] access the data value
"a<-source('c:\\scripts\\r_scripts\\sample.r');a[[1]]"
```

R log file

Analytics logs R language messages to an `aclrlang.log` file in the project folder. Use this log file for debugging R errors.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Tip**
>
> The log file is available in the Results folder of Analytics Exchange analytic jobs.

Running external R scripts on AX Server

If you are writing an analysis app to run on AX Server and you want to work with external R scripts:

1. Upload the file as a related file with the analysis app.
2. Use the FILE analytic tag to identify the file(s).
3. Reference the file(s) using the relative path `./filename.r`.

> **Note**
>
> Using a related file ensures that the TomEE application server account has sufficient permissions to access the file when running R with Analytics Exchange.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ROOT( ) function

Returns the square root of a numeric expression.

Syntax

```
ROOT(number, decimals)
```

Parameters

Name	Type	Description
number	numeric	The numeric expression to find the square root of. This function returns zero if *number* is a negative number.
decimals	numeric	The number of decimals to use in the output.

Output

Numeric.

Examples

Basic examples

Returns 10.00:

```
ROOT(100, 2)
```

Returns 31.6228:

```
ROOT(1000, 4)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

How it works

The ROOT( ) function returns the square root of the numeric expression or field value with the specified number of decimal places. The result is rounded appropriately.

When to use ROOT( )

Use LOG( ) to perform other root functions, such as cube root.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ROUND( ) function

Returns a rounded whole number for a numeric value.

Syntax

```
ROUND(number)
```

Parameters

Name	Type	Description
number	numeric	The value to round to the nearest integer.

Output

Numeric.

Examples

Basic examples

Returns 7:

```
ROUND(7.2)
```

Returns 8:

```
ROUND(7.5)
```

Returns -8:

```
ROUND(-7.5)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Advanced examples

Rounding monetary values

Creates a field that is equal to the balance rounded to the nearest dollar value:

```
DEFINE FIELD Nearest_dollar_value COMPUTED ROUND(Balance)
```

Remarks

How it works

ROUND( ) returns a number equal to the *number* value rounded to the nearest integer:

	Positive values	Negative values
Rounds up to the next integer	>= 0.5	< 0.5
Rounds down to the next integer	< 0.5	>= 0.5

Rounding to a particular number of decimal places

If you want to round a number to a particular number of decimal places, use the "DEC( ) function" on page 542. The ROUND( ) function is the same as the DEC( ) function with zero decimal places specified.

```
ROUND(number)
```

is equivalent to:

```
DEC(number, 0)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RSTRING( ) function

Returns a string value calculated by an R function or script. Data processing in R is external to Analytics.

Syntax

RSTRING(*rScript|rCode*, *length* <,*field|value* <,...*n*>>)

Parameters

Name	Type	Description
rScript | *rCode*	character	The full or relative path to the R script, or a snippet of R code to run. If you enter R code directly rather than use an external file, you cannot use the enclosing quotation character in your code, even if you escape it: ◦ **valid** - 'var <- "\"test\"" ' ◦ **invalid** - 'var <- "\'test\'" '
length	numeric	The length to allocate for the return string.
field | *value* <,...*n*> optional	character numeric datetime logical	The list of fields, expressions, or literal values to use as arguments for the R script or code snippet. The values are passed into the function you call in the order you specify them, and you reference them using value1, value2 ... value*N* in the R code. You may include as many arguments as necessary to satisfy the function definition in the R code. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Output

Character.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Returns "abc123":

```
RSTRING("print(paste(value1,value2,sep=""))",6,"abc","123")
```

Advanced examples

Using an external R script

Concatenates x and y into a single string delimited by a space character:

```
RSTRING("a<-source('./sample.r');a[[1]]",50, FirstName, LastName)
```

External R script (sample.r):

```
conc <- function(x, y) {
   paste(x, y, sep=" ")
}
print(conc(value1, value2))
```

Using R code stored in a variable

Concatenates x and y into a single string delimited by a space character:

```
ASSIGN v_script = "conc <- function(x, y){paste(x, y, sep=' ')};conc(value1, value2)"
RSTRING(v_script, 50, FirstName, LastName)
```

Using R to generate a UUID for a table

You are preparing a table of exceptions to upload to Results and you require a guaranteed unique identifier for each record. To generate this field, you use the **uuid** package in R to create a unique primary key value for each record:

```
EXTRACT RSTRING("uuid::UUIDgenerate()", 36) AS "id", first_name, last_name, birthdate TO
export_table
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Tip**
>
> To install the uuid package, open R.exe and execute the following command:

```
install.packages("uuid")
```

Remarks

Returning data from R

When calling R scripts, use the source function and assign the return object to a variable. You can then access the value returned from your R function from the return object:

```
# 'a' holds the response object and a[[1]] access the data value
"a<-source('c:\\scripts\\r_scripts\\sample.r');a[[1]]"
```

R log file

Analytics logs R language messages to an `aclrlang.log` file in the project folder. Use this log file for debugging R errors.

> **Tip**
>
> The log file is available in the Results folder of Analytics Exchange analytic jobs.

Running external R scripts on AX Server

If you are writing an analysis app to run on AX Server and you want to work with external R scripts:

1. Upload the file as a related file with the analysis app.
2. Use the FILE analytic tag to identify the file(s).
3. Reference the file(s) using the relative path `./filename.r`.

> **Note**
>
> Using a related file ensures that the TomEE application server account has sufficient permissions to access the file when running R with Analytics Exchange.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RTIME( ) function

Returns a time value calculated by an R function or script. Data processing in R is external to Analytics.

Syntax

RTIME(*rScript|rCode* <,*field|value* <,...*n*>>)

Parameters

Name	Type	Description
rScript | *rCode*	character	The full or relative path to the R script, or a snippet of R code to run. If you enter R code directly rather than use an external file, you cannot use the enclosing quotation character in your code, even if you escape it: ◦ **valid** - 'var <- "\"test\"" ' ◦ **invalid** - 'var <- "\'test\'" '
field | *value* <,...*n*> optional	character numeric datetime logical	The list of fields, expressions, or literal values to use as arguments for the R script or code snippet. The values are passed into the function you call in the order you specify them, and you reference them using value1, value2 ... value*N* in the R code. You may include as many arguments as necessary to satisfy the function definition in the R code. **Note** Use the ALLTRIM() function to remove any leading or trailing spaces from character input: ALLTRIM(str). For more information, see "ALLTRIM( ) function" on page 473.

Output

Datetime.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Returns `t0545`:

```
RTIME("value1+2700",`t0500`)
```

Advanced examples

Using an external R script

Adds 45 minutes to a time field by passing a field and a literal value to an external R function:

```
RTIME("a<-source('c:\\scripts\\sample.r');a[[1]]", end_time, 2700)
```

External R script (sample.r):

```
add_time <- function(start, sec) {
  return(start + sec)
}
add_time(value1, value2)
```

Remarks

Returning data from R

When calling R scripts, use the source function and assign the return object to a variable. You can then access the value returned from your R function from the return object:

```
# 'a' holds the response object and a[[1]] access the data value
"a<-source('c:\\scripts\\r_scripts\\sample.r');a[[1]]"
```

R log file

Analytics logs R language messages to an `aclrlang.log` file in the project folder. Use this log file for debugging R errors.

> **Tip**
>
> The log file is available in the Results folder of Analytics Exchange analytic jobs.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Running external R scripts on AX Server

If you are writing an analysis app to run on AX Server and you want to work with external R scripts:

1. Upload the file as a related file with the analysis app.
2. Use the FILE analytic tag to identify the file(s).
3. Reference the file(s) using the relative path `./filename.r`.

> **Note**
>
> Using a related file ensures that the TomEE application server account has sufficient permissions to access the file when running R with Analytics Exchange.

System time zone

Greenwich Mean Time (GMT) is the default current time zone in the R environment used by Analytics.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SECOND( ) function

Extracts the seconds from a specified time or datetime and returns it as a numeric value.

Syntax

```
SECOND(time/datetime)
```

Parameters

Name	Type	Description
time/datetime	datetime	The field, expression, or literal value to extract the seconds from.

Output

Numeric.

Examples

Basic examples

Returns 30:

```
SECOND(`t235930`)
```

```
SECOND(`20141231 235930`)
```

Returns the seconds for each value in the **Call_start_time** field:

```
SECOND(Call_start_time)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Parameter details

A field specified for *time/datetime* can use any time or datetime format, as long as the field definition correctly defines the format.

Specifying a literal time or datetime value

When specifying a literal time or datetime value for *time/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231 235959` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Time values** – you can use any of the time formats listed in the table below. You must use a separator before a standalone time value for the function to operate correctly. Valid separators are the letter 't', or the letter 'T'. You must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).
- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.

Example formats	Example literal values
thhmmss	`` `t235959` ``
Thhmm	`` `T2359` ``
YYYYMMDD hhmmss	`` `20141231 235959` ``
YYMMDDthhmm	`` `141231t2359` ``
YYYYMMDDThh	`` `20141231T23` ``
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`` `20141231 235959-0500` ``
YYMMDD hhmm+/-hh (UTC offset)	`` `141231 2359+01` ``
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SHIFT( ) function

Returns a single character string with the bits in the first character of the input value shifted to the left or right.

Syntax

```
SHIFT(character, number_of_bits_to_left)
```

Parameters

Name	Type	Description
character	character	The value to shift bits for.
number_of_bits_to_left	numeric	Specifies the number of bits to shift the *character* value. ◦ **If the value is positive** - *character* is shifted to the left ◦ **If the value is negative** - *character* is shifted to the right If the specified value is greater than 15 or less than -15 the result is binary zero, CHR(0).

Output

Character.

Examples

Basic examples

Returns the letter "X", or CHR(88) (00010110 becomes 01011000):

```
SHIFT(CHR(22), 2)
```

Returns the backspace character, or CHR(8) (00010000 becomes 00001000):

```
SHIFT(CHR(16), -1)
```

Returns the grave accent character, or CHR(96) (10011011 becomes 01100000):

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
SHIFT(CHR(155), 5)
```

Remarks

When to use SHIFT( )

Use the SHIFT( ) function in conjunction with the BYTE( ), CHR( ) and MASK( ) functions to isolate and move individual bits in a record.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SIN( ) function

Returns the sine of an angle expressed in radians, with a precision of 15 decimal places.

Syntax

```
SIN(radians)
```

Parameters

Name	Type	Description
radians	numeric	The measurement of the angle in radians.

Output

Numeric.

Examples

Basic examples

Returns 0.500000000000000 (the sine of the specified number of *radians*, equivalent to 30 degrees):

```
SIN(0.523598775598299)
```

Returns 0.500000000000000 (the sine value of 30 degrees):

```
SIN(30 * PI( )/180)
```

Advanced examples

Using degrees as input

Returns 0.500 (the sine of 30 degrees, rounded to 3 decimal places):

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DEC(SIN(30 * PI( )/180),3)
```

Remarks

Performing the Mantissa Arc Test

The three trigonometric functions in Analytics – SIN( ), COS( ), and TAN( ) – support performing the Mantissa Arc Test associated with Benford's Law.

Converting degrees to radians

If your input is in degrees you can use the PI( ) function to convert the input to radians: (*degrees* * PI ( )/180) = *radians*. If required, you can round or truncate the return value using the DEC( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SOUNDEX( ) function

Returns the soundex code for the specified string, which can be used for phonetic comparisons with other strings.

Syntax

```
SOUNDEX(name)
```

Parameters

Name	Type	Description
name	character	The character expression to evaluate.

Output

Character. Returns a four-character soundex code.

Examples

Basic examples

Words that sound the same but are spelled differently

The two examples below return the same soundex code because they sound the same even though they are spelled differently.

Returns F634:

```
SOUNDEX("Fairdale")
```

Returns F634:

```
SOUNDEX("Faredale")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Words that sound similar

The two examples below return soundex codes that are different, but close to one another, because the two words sound similar.

Returns J525:

```
SOUNDEX("Jonson")
```

Returns J523:

```
SOUNDEX("Jonston")
```

Words that sound different

The two examples below return soundex codes that are quite different, because the two words sound nothing alike.

Returns S530:

```
SOUNDEX("Smith")
```

Returns M235:

```
SOUNDEX("MacDonald")
```

Field input

Returns the soundex code for each value in the **Last_Name** field:

```
SOUNDEX(Last_Name)
```

Advanced examples

Identifying matching soundex codes

Create the computed field **Soundex_Code** to display the soundex code for each value in the **Last_Name** field:

```
DEFINE FIELD Soundex_Code COMPUTED SOUNDEX(Last_Name)
```

Add the computed field **Soundex_Code** to the view, and then perform a duplicates test on the computed field to identify any matching soundex codes:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DUPLICATES ON Soundex_Code OTHER Last_Name PRESORT OPEN TO "Possible_Dupes.fil"
```

Matching soundex codes indicate that the associated character values in the **Last_Name** field are possible duplicates.

Remarks

When to use SOUNDEX( )

Use the SOUNDEX( ) function to find values that sound similar. Phonetic similarity is one way of locating possible duplicate values, or inconsistent spelling in manually entered data.

How it works

SOUNDEX( ) returns the American Soundex code for the evaluated string. All codes are one letter followed by three numbers. For example: "F634".

How the soundex code is derived

- The first character in the code represents the first letter of the evaluated string.
- Each number in the code represents one of the six American Soundex groups. The groups are composed of phonetically similar consonants.

  Based on these groups, the soundex process encodes the first three consonants in the evaluated string after the first letter.

What the soundex process ignores

The soundex process ignores:

- capitalization
- vowels
- the consonants "H", "W", and "Y"
- any consonants that appear after the three encoded consonants

One or more trailing zeros (0) in the returned code indicate an evaluated string with fewer than three consonants after the first letter.

Limitations of the soundex process

Both the SOUNDEX( ) and SOUNDSLIKE( ) functions have certain limitations:

- The soundex algorithm is designed to work with words pronounced in English, and has varying degrees of effectiveness when used with other languages.
- Although the soundex process performs a phonetic match, matching words must all begin with the same letter, which means that some words that sound the same are not matched.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

For example, a word that begins with "F", and a word that begins with a "Ph", could sound the same but they will never be matched.

Related functions

- **SOUNDSLIKE( )** – an alternate method for phonetically comparing strings.
- **ISFUZZYDUP( ) and LEVDIST** – compare strings based on an orthographic comparison (spelling) rather than on a phonetic comparison (sound).
- **DICECOEFFICIENT( )** – de-emphasizes or completely ignores the relative position of characters or character blocks when comparing strings.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SOUNDSLIKE( ) function

Returns a logical value indicating whether a string phonetically matches a comparison string.

Syntax

```
SOUNDSLIKE(name, sounds_like_name)
```

Parameters

Name	Type	Description
name	character	The first string in the comparison.
sounds_like_name	character	The second string in the comparison.

Output

Logical. Returns **T** (true) if the values being compared phonetically match, and **F** (false) otherwise.

Examples

Basic examples

Returns T, because "Fairdale" and "Faredale" both have a soundex code of F634:

```
SOUNDSLIKE("Fairdale","Faredale")
```

Returns F, because "Jonson" has a soundex code of J525, and "Jonston" has a soundex code of J523:

```
SOUNDSLIKE("Jonson","Jonston")
```

Returns a logical value (T or F) indicating whether the soundex code for each value in the **Last_Name** field matches the soundex code for the string "Smith":

```
SOUNDSLIKE(Last_Name,"Smith")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Advanced examples

Isolating values that sound like "Smith"

Create a filter that isolates all values in the **Last_Name** field that sound like "Smith":

```
SET FILTER TO SOUNDSLIKE(Last_Name,"Smith")
```

Remarks

When to use SOUNDSLIKE( )

Use the SOUNDSLIKE( ) function to find values that sound similar. Phonetic similarity is one way of locating possible duplicate values, or inconsistent spelling in manually entered data.

How it works

SOUNDSLIKE( ) converts the comparison strings to four-character American Soundex codes, which are based on the first letter, and the first three consonants after the first letter, in each string.

The function then compares each string's code and returns a logical value indicating whether they match.

For more information about soundex codes, see "SOUNDEX( ) function" on page 783.

Case sensitivity

The function is not case-sensitive, so "SMITH" is equivalent to "smith."

Limitations of the soundex process

Both the SOUNDSLIKE( ) and SOUNDEX( ) functions have certain limitations:

- The soundex algorithm is designed to work with words pronounced in English, and has varying degrees of effectiveness when used with other languages.
- Although the soundex process performs a phonetic match, matching words must all begin with the same letter, which means that some words that sound the same are not matched.

  For example, a word that begins with "F", and a word that begins with a "Ph", could sound the same but they will never be matched.

Related functions

- **SOUNDEX( )** – an alternate method for phonetically comparing strings.
- **ISFUZZYDUP( ) and LEVDIST** – compare strings based on an orthographic comparison (spelling) rather than on a phonetic comparison (sound).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- **DICECOEFFICIENT( )** – de-emphasizes or completely ignores the relative position of characters or character blocks when comparing strings.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SPLIT( ) function

Returns a specified segment from a string.

Syntax

SPLIT(*string*, *separator*, *segment* <,*text_qualifier*>)

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to extract the segment from.
separator	character	The character or characters that delimit segments. For more information, see "How the separator character works" on page 792.
segment	numeric	The segment to extract. Use a number to specify which segment to extract. For example, to extract the third segment, specify 3.
text_qualifier optional	character	The character or characters that indicate the start and end of segments of text. If the *separator* character occurs inside a paired set of text qualifiers, it is read as text and not as a separator. You must enclose the *text qualifier* in quotation marks. A single quotation mark *text qualifier* must be enclosed in double quotation marks, and a double quotation mark *text qualifier* must be enclosed in single quotation marks. **Tip** This optional parameter can be useful when working with imported source data that retains separators and text qualifiers.

Output

Character.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Comma-delimited segments

Returns "seg1":

```
SPLIT("seg1,seg2,seg3", ",", 1)
```

Returns "seg3":

```
SPLIT("seg1,seg2,seg3", ",", 3)
```

Returns "" (the third segment is empty):

```
SPLIT("seg1,seg2,,seg4", ",", 3)
```

Multi-character and space delimiters

Returns "seg3":

```
SPLIT("seg1/*seg2/*seg3", "/*", 3)
```

Returns "Doe":

```
SPLIT("Jane Doe", " ", 2)
```

Escaping delimiters with a text qualifier

Returns "Doe, Jane", which includes a comma that is read as text rather than as a separator:

```
SPLIT('"Doe, Jane","Smith, John"', ",", 1, '"')
```

Advanced examples

Extracting digits from a credit card number

Use the SPLIT( ) command to remove dashes from a credit card number.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Variables are used to capture each segment of the credit card number, and then the segments are concatenated together in an additional variable.

```
ASSIGN seg1 = SPLIT("4150-2222-3333-4444", "-", 1)
ASSIGN seg2 = SPLIT("4150-2222-3333-4444", "-", 2)
ASSIGN seg3 = SPLIT("4150-2222-3333-4444", "-", 3)
ASSIGN seg4 = SPLIT("4150-2222-3333-4444", "-", 4)
ASSIGN ccNum = seg1 + seg2 + seg3 + seg4
```

The value of ccNum is "4150222233334444".

The example illustrates the SPLIT( ) function, but note that the dashes can be removed more efficiently using the EXCLUDE( ) function.

Remarks

How it works

The SPLIT( ) function breaks character data into segments based on separators such as spaces or commas and returns a specified segment.

When to use SPLIT( )

Use the SPLIT( ) function to extract a particular segment of data from a record or field. The segment must appear in the same position in each record or field.

How the separator character works

The separator character delimits, or indicates, the segments of data in a source string.

In a string with a number of segments, most of the segments appear between two separators. However, the first segment may not have a separator character preceding it, and the last segment may not have a separator character following it.

If the source string does not begin with a separator, the segment preceding the first separator is treated as segment 1.

Returns "seg1":

```
SPLIT("seg1,seg2,seg3", ",", 1)
```

If the source string begins with a separator, segment 1 is consider to be null. The segment that follows the separator is treated as segment 2.

Returns "seg1":

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
SPLIT(",seg1,seg2,seg3", ",", 2)
```

Case sensitivity

If *separator* or *text_qualifier* specify characters that have both an uppercase and a lowercase version, the case used must match the case of the separator or text qualifier in the data.

Related functions

SPLIT( ) and SUBSTR( ) both return a segment of data from a longer source string.

- SPLIT( ) identifies the segment based on a separator character.
- SUBSTR( ) identifies the segment based on a numeric character position.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

STOD( ) function

Converts a serial date – that is, a date expressed as an integer – to a date value. Abbreviation for "Serial to Date".

Syntax

```
STOD(serial_date <,start_date>)
```

Parameters

Name	Type	Description
serial_date	numeric	The field, expression, or literal value to convert. *serial_date* can be a serial date or a serial datetime. Only the date portion of serial datetimes is considered. The time portion is ignored.
start_date optional	datetime	The start date from which serial dates are calculated. If omitted, the default start date of 01 January 1900 is used.

Output

Datetime. The date value is output using the current Analytics date display format.

Examples

Basic examples

Returns `20141231` displayed as 31 Dec 2014 assuming a current Analytics date display format of DD MMM YYYY:

```
STOD(42003)
```

Returns `20181231` displayed as 31 Dec 2018 assuming a current Analytics date display format of DD MMM YYYY:

```
STOD(42003, `19040101`)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns the equivalent date for each serial date value in the **Invoice_Date** field:

```
STOD(Invoice_Date)
```

Advanced examples

Adjusting for a start date before 1900-01-01

Use date arithmetic to adjust the start date to a value that is earlier than the Analytics minimum date of January 1, 1900:

1. Convert the serial date using the default start date.
2. Subtract the number of days before 1900-01-01 that the actual start date falls.

To use 1899-01-01 as the start date (evaluates to `20131231`):

```
STOD(42003) - 365
```

Remarks

How it works

The STOD( ) function allows you to convert serial dates to regular dates. Analytics serial dates represent the number of days that have elapsed since 01 January 1900.

Serial date	Regular date equivalent
1	02 January 1900
365	31 December 1900
42003	31 December 2014
0	not valid

For more information about serial dates, see Serial datetimes.

Analytics serial dates compared to Excel serial dates

Analytics serial dates are similar to Microsoft Excel serial dates. You should be aware of one key point of similarity and one key point of difference. The two points are unrelated.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Point of similarity

Both Analytics and Excel treat the year 1900 as a leap year, with 366 days. Although 1900 was not in fact a leap year, Excel treated it as one in order to maintain compatibility with Lotus 1-2-3.

Point of difference

Analytics serial dates are offset from Excel serial dates by one day. In Excel, 01 January 1900 has a serial date of '1'. In Analytics, 01 January 1900 is not counted, and 02 January 1900 has a serial date of '1'.

The *start_date*

Some source data files may use a start date other than 01 January 1900. The *start_date* allows you to match the start date in a source data file. The start date is the date from which serial dates are calculated.

Start date in source data file	Specify:	Details
01 January 1900	STOD(*date_field*)	You do not need to specify a *start_date*, because 01 January 1900 is the default start date.
01 January 1901	STOD(*date_field*, `19010101`)	You specify a *start_date* of `19010101` to match the start date of 01 January 1901 used in the source data file.
01 January 1899	STOD(*date_field*) - 365	You cannot specify a *start_date* earlier than 01 January 1900. If a source data file uses a start date earlier than 01 January 1900, you can create a datetime expression that subtracts an appropriate number of days from the output results of the STOD ( ) function.

Other datetime conversion functions

Serial to Datetime conversion

Function	Description
STODT( )	Converts a serial datetime - that is, a datetime expressed as an integer, and a fractional portion of 24 hours - to a datetime value. Abbreviation for "Serial to Datetime".
STOT( )	Converts a serial time - that is, a time expressed as a fractional portion of 24 hours, with 24 hours equaling 1 - to a time value. Abbreviation for "Serial to Time".

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Character or Numeric to Datetime conversion

Function	Description
CTOD( )	Converts a character or numeric date value to a date. Can also extract the date from a character or numeric datetime value and return it as a date. Abbreviation for "Character to Date".
CTODT( )	Converts a character or numeric datetime value to a datetime. Abbreviation for "Character to Datetime".
CTOT( )	Converts a character or numeric time value to a time. Can also extract the time from a character or numeric datetime value and return it as a time. Abbreviation for "Character to Time".

Datetime to Character conversion

Function	Description
DATE( )	Extracts the date from a specified date or datetime and returns it as a character string. Can also return the current operating system date.
DATETIME( )	Converts a datetime to a character string. Can also return the current operating system datetime.
TIME( )	Extracts the time from a specified time or datetime and returns it as a character string. Can also return the current operating system time.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

STODT( ) function

Converts a serial datetime – that is, a datetime expressed as an integer, and a fractional portion of 24 hours – to a datetime value. Abbreviation for "Serial to Datetime".

Syntax

```
STODT(serial_datetime <,start_date>)
```

Parameters

Name	Type	Description
serial_datetime	numeric	The field, expression, or literal value to convert. Serial datetime values with the date and time portions separated by a decimal point are required. For example, 42003.75000
start_date optional	datetime	The start date from which serial dates are calculated. If omitted, the default start date of 01 January 1900 is used.

Output

Datetime. The datetime value is output using the current Analytics date and time display formats.

Examples

Basic examples

Unadjusted start dates

Returns `20141231t060000` displayed as 31 Dec 2014 06:00:00 AM assuming current Analytics date and time display formats of DD MMM YYYY and hh:mm:ss PM:

```
STODT(42003.25000)
```

Returns `20141231t191530` displayed as 31 Dec 2014 07:15:30 PM assuming current Analytics date and time display formats of DD MMM YYYY and hh:mm:ss PM:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
STODT(42003.802431)
```

Adjusted start dates

Returns `20181231t120000` displayed as 31 Dec 2018 12:00:00 PM assuming current Analytics date and time display formats of DD MMM YYYY and hh:mm:ss PM:

```
STODT(42003.50000, `19040101`)
```

Fields as input

Returns the equivalent datetime for each serial datetime value in the **Receipt_datetime** field:

```
STODT(Receipt_datetime)
```

Advanced examples

Adjusting for a start date before 1900-01-01

Use date arithmetic to adjust the start date to a value that is earlier than the Analytics minimum date of January 1, 1900:

1. Convert the serial datetime using the default start date.
2. Subtract the number of days before 1900-01-01 that the actual start date falls.

To use 1899-01-01 as the start date (evaluates to `20131231t180000`):

```
STODT(42003.75000) - 365
```

Remarks

How it works

The STODT( ) function allows you to convert serial datetimes to regular datetimes. Analytics serial datetimes represent the number of days that have elapsed since 01 January 1900, and following the decimal point, represent a fractional portion of 24 hours, with 24 hours equaling 1.

Serial datetime	Regular datetime equivalent
1.25	02 January 1900 06:00:00 AM
365.75000	31 December 1900 06:00:00 PM

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Serial datetime	Regular datetime equivalent
42003.79167	31 December 2014 07:00:00 PM
42003.802431	31 December 2014 07:15:30 PM
42003.00000	31 December 2014 12:00:00 AM
42003.50000	31 December 2014 12:00:00 PM
0.0	not valid

For more information about serial datetimes, see Serial datetimes.

Analytics serial dates compared to Excel serial dates

Analytics serial dates are similar to Microsoft Excel serial dates. You should be aware of one key point of similarity and one key point of difference. The two points are unrelated.

Point of similarity

Both Analytics and Excel treat the year 1900 as a leap year, with 366 days. Although 1900 was not in fact a leap year, Excel treated it as one in order to maintain compatibility with Lotus 1-2-3.

Point of difference

Analytics serial dates are offset from Excel serial dates by one day. In Excel, 01 January 1900 has a serial date of '1'. In Analytics, 01 January 1900 is not counted, and 02 January 1900 has a serial date of '1'.

The *start_date*

Some source data files may use a start date other than 01 January 1900. The *start_date* allows you to match the start date in a source data file. The start date is the date from which serial datetimes are calculated.

Start date in source data file	Specify:	Details
01 January 1900	STODT(*datetime_field*)	You do not need to specify a *start_date*, because 01 January 1900 is the default start date.
01 January 1901	STODT(*datetime_field*, `19010101`)	You specify a *start_date* of `19010101` to match the start date of 01 January 1901 used in the source data file.
01 January 1899	STODT(*datetime_field*) - 365	You cannot specify a *start_date* earlier than 01 January 1900. If a source data file uses a start date earlier than 01 January 1900, you can create a datetime expression that subtracts an

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Start date in source data file	Specify:	Details
		appropriate number of days from the output results of the STODT( ) function.

Other datetime conversion functions

Serial to Datetime conversion

Function	Description
STOD( )	Converts a serial date - that is, a date expressed as an integer - to a date value. Abbreviation for "Serial to Date".
STOT( )	Converts a serial time - that is, a time expressed as a fractional portion of 24 hours, with 24 hours equaling 1 - to a time value. Abbreviation for "Serial to Time".

Character or Numeric to Datetime conversion

Function	Description
CTOD( )	Converts a character or numeric date value to a date. Can also extract the date from a character or numeric datetime value and return it as a date. Abbreviation for "Character to Date".
CTODT( )	Converts a character or numeric datetime value to a datetime. Abbreviation for "Character to Datetime".
CTOT( )	Converts a character or numeric time value to a time. Can also extract the time from a character or numeric datetime value and return it as a time. Abbreviation for "Character to Time".

Datetime to Character conversion

Function	Description
DATE( )	Extracts the date from a specified date or datetime and returns it as a character string. Can also return the current operating system date.
DATETIME( )	Converts a datetime to a character string. Can also return the current operating system datetime.
TIME( )	Extracts the time from a specified time or datetime and returns it as a character string. Can also return the current operating system time.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

STOT( ) function

Converts a serial time – that is, a time expressed as a fractional portion of 24 hours, with 24 hours equaling 1 – to a time value. Abbreviation for "Serial to Time".

Syntax

```
STOT(serial_time)
```

Parameters

Name	Type	Description
serial_time	numeric	The field, expression, or literal value to convert. *serial_time* can be a serial time or a serial datetime. Only the time portion of serial datetimes is considered. The date portion is ignored.

Output

Datetime. The time value is output using the current Analytics time display format.

Examples

Basic examples

Returns `t060000` displayed as 06:00:00 AM assuming a current Analytics time display format of hh:mm:ss PM:

```
STOT(0.25000)
```

Returns `t191530` displayed as 07:15:30 PM assuming a current Analytics time display format of hh:mm:ss PM:

```
STOT(0.802431)
```

Returns the equivalent regular time for each serial time value in the **Login_time** field:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
STOT(Login_time)
```

Remarks

When to use STOT( )

Use the STOT( ) function to convert serial times to regular times.

What are serial times?

Analytics serial times represent a fractional portion of 24 hours, with 24 hours equaling 1.

For example:

- the serial time equivalent of 1 hour is 1/24, or 0.04167
- the serial time equivalent of 1 minute is 1/1440, or 0.0006945

Serial times can be prefaced with a '0' (zero) and a decimal point, or just a decimal point.

1.000000 is not a valid serial time

Although 24 hours equals 1 for the purposes of calculating serial times, 1.000000 is not a valid serial time. Valid serial times are all decimal fractions less than 1. For example: 0.75000 (06:00:00 PM).

Analytics treats the serial number 1.000000 as the serial datetime equivalent to 02 Jan 1900 12:00:00 AM. Because STOT( ) ignores the date portion of datetimes, `STOT(1.000000)` is equivalent to `STOT(0.000000)` and both are equivalent to the regular time 12:00:00 AM.

Serial times and regular time equivalents

Serial time	Regular time equivalent
0.00	12:00:00 AM
0.0006945	12:01:00 AM
0.04167	01:00:00 AM
0.0423645	01:01:00 AM
0.042998	01:01:55 AM
0.25	06:00:00 AM
0.50	12:00:00 PM

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Serial time	Regular time equivalent
0.75	06:00:00 PM
0.79167	07:00:00 PM
0.802431	07:15:30 PM
1.00	12:00:00 AM

Other datetime conversion functions

Serial to Datetime conversion

Function	Description
STOD( )	Converts a serial date - that is, a date expressed as an integer - to a date value. Abbreviation for "Serial to Date".
STODT( )	Converts a serial datetime - that is, a datetime expressed as an integer, and a fractional portion of 24 hours - to a datetime value. Abbreviation for "Serial to Datetime".

Character or Numeric to Datetime conversion

Function	Description
CTOD( )	Converts a character or numeric date value to a date. Can also extract the date from a character or numeric datetime value and return it as a date. Abbreviation for "Character to Date".
CTODT( )	Converts a character or numeric datetime value to a datetime. Abbreviation for "Character to Datetime".
CTOT( )	Converts a character or numeric time value to a time. Can also extract the time from a character or numeric datetime value and return it as a time. Abbreviation for "Character to Time".

Datetime to Character conversion

Function	Description
DATE( )	Extracts the date from a specified date or datetime and returns it as a character string. Can also return the current operating system date.
DATETIME( )	Converts a datetime to a character string. Can also return the current operating system datetime.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Function	Description
TIME( )	Extracts the time from a specified time or datetime and returns it as a character string. Can also return the current operating system time.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

STRING( ) function

Converts a numeric value to a character string.

Syntax

```
STRING(number, length<,format>)
```

Parameters

Name	Type	Description
number	numeric	The numeric value to convert to a string.
length	numeric	The length of the output string in characters. ◦ If *length* is longer than *number*, leading spaces are added to the output string ◦ If *length* is shorter than *number*, the output string is truncated from the left side Ensure that the length you specify provides enough space for the longest numeric value in a field, including any non-numeric format characters if you specify the optional *format* parameter.
format optional	character	The format to apply to the output string. *format* must be enclosed in double quotation marks. For example, "(9,999.99)" Use the optional *format* parameter to add formatting to the output string that is not present in the source data. You can add a dollar sign, a percent sign, one or more decimal placeholders, a thousands separator, parentheses, and so on. **Note** Non-numeric format characters that you specify increase the length of *number*.

Output

Character.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples, output not formatted

Numeric value 125.2

Returns " 125.2":

```
STRING(125.2, 6)
```

The output string is padded with one leading space because the *length* value is 6, which is one character longer than the number of digits and format characters in *number*.

Numeric value -125.2

Returns "25.2":

```
STRING(-125.2, 4)
```

The output string is truncated because the *length* value is 4, which is two characters shorter than the number of digits and format characters in *number*.

Returns " -125.2":

```
STRING(-125.2, 7)
```

The output string is padded with one leading space because the *length* value is 7, which is one character longer than the number of digits and format characters in *number*.

Basic examples, output formatted

Numeric value 125.2

Returns "25.20":

```
STRING(125.2, 6, "(9,999.99)")
```

The output string is truncated because the *length* value is 6, which is one character shorter than the *number* value after the specified format is applied.

Returns "125.20":

```
STRING(125.2, 7, "(9,999.99)")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> Starting from the right, the characters you specify for *format* are included in the calculation of the length of *number* even if a format character is not required for a specific instance of *number*. In the examples above, the right-hand parenthesis is counted as a character even though it is not required for a positive value in *number*.

Numeric value -125.2

Returns " (125.20)":

```
STRING(-125.2, 10, "(9,999.99)")
```

The output string is padded with two leading spaces because the *length* value is 10, which is two characters longer than the *number* value after the specified format is applied.

Basic example, field input

Returns numeric values in the **Employee_number** field as character strings with a length of 10 characters. If required, the output string is padded or truncated:

```
STRING(Employee_number, 10)
```

Remarks

Formatting the output string

You can format the output string to display formatting that might be missing in the source data.

Placeholder digits in the format

In the format that you specify, the digit 9 acts as a placeholder for digits. Ensure that you specify enough placeholder digits to account for the longest numeric value in a field. For example, if a field contains amounts up to $5,000,000, with two decimal places, you need to specify nine placeholder digits: "$9,999,999.99"

How the format affects the minimum required output string length

The value you specify for *length* must, at a minimum, be long enough to contain all the digits in the longest value in a field, as well as any format characters that you specify.

If you want to add a dollar sign, and thousands separators, to the values in the field containing amounts up to $5,000,000, you need to specify at least 13 for *length*: 9 digits + 4 non-numeric format characters.

Returns numeric values in the **Amount** field as character strings with the specified format displayed.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
STRING(Amount, 13, "$9,999,999.99")
```

Returns $4,789,123.50, as a character string:

```
STRING(4789123.50, 13, "$9,999,999.99")
```

Related functions

The STRING( ) function is the opposite of the VALUE( ) function, which converts character data to numeric data.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

SUBSTR( ) function

Returns a specified substring from a string.

Syntax

```
SUBSTR(string, start, length)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to extract the substring from.
start	numeric	The starting character position of the substring. The numeric positions of the characters in *string* start at `1` . To extract a substring beginning with `C` from the string `ABCDEF` , you would specify a *start* value of `3`.
length	numeric	The number of characters in the substring. If *length* is 0, the output is blank.

Output

Character.

Examples

Basic examples

Literal character input

Returns "BCD":

```
SUBSTR("ABCDEF", 2, 3)
```

Returns "EF":

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
SUBSTR("ABCDEF", 5, 10)
```

Parsing structured character data

Returns "189543":

```
SUBSTR("***189543***", 4, 6)
```

Returns the four-digit year out of a character field containing dates formatted as "MM/DD/YYYY":

```
SUBSTR(DATE, 7, 4)
```

Advanced examples

Increasing field length

Use SUBSTR( ) to increase the length of a character field. Increasing the length of a field is a common harmonization task that you may need to perform before joining or appending two fields.

The example below pads the **Product_Description** field with blank spaces to create the computed field **Product_Description_Long** with a length of 50 characters.

```
DEFINE FIELD Product_Description_Long COMPUTED SUBSTR(Product_Description, 1, 50)
```

Remarks

How it works

The SUBSTR( ) function returns characters from the *string* value starting at the character position specified by *start*. The number of characters returned is specified by *length*.

How SUBSTR( ) handles spaces

Leading, trailing, or internal spaces in the *string* value are treated like characters. Spaces captured by *start* and *length* are included in the output string.

How padding works

If the *length* value exceeds the number of characters, including trailing spaces, from the *start* position to the end of *string*, the output string may or may not be padded on the right with spaces.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Padded with spaces

If you use SUBSTR( ) within a command that creates a field, the output is padded with spaces.

Padding when creating a computed field

Creates the computed field **Product_Description_Long**, with a length of 50 characters, based on the physical field **Product_Description**, with a length of 24 characters:

```
DEFINE FIELD Product_Description_Long COMPUTED SUBSTR(Product_Description, 1, 50)
```

Padding when extracting a physical field

Extracts the field **Product_Description_Long**, with a length of 50 characters, to a new table, based on the physical field **Product_Description**, with a length of 24 characters:

```
EXTRACT FIELDS SUBSTR(Product_Description, 1, 50) AS "Product_Description_Long" TO New_Table
```

Not padded with spaces

If you use SUBSTR( ) in a variable definition or an expression, the output is not padded with spaces.

No padding when defining a variable

Creates the variable *v_prod_desc*, with a length of 24 characters, based on the field length of **Product_Description**:

```
ASSIGN v_prod_desc = SUBSTR(Product_Description, 1, 50)
```

> **Note**
>
> Even though SUBSTR( ) specifies a *length* of 50 characters, the output is limited to the field length of **Product_Description**.

Related functions

SUBSTR( ) and SPLIT( ) both return a segment of data from a longer source string.

- SUBSTR( ) identifies the segment based on a numeric character position.
- SPLIT( ) identifies the segment based on a separator character.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

TAN( ) function

Returns the tangent of an angle expressed in radians, with a precision of 15 decimal places.

Syntax

```
TAN(radians)
```

Parameters

Name	Type	Description
radians	numeric	The measurement of the angle in radians.

Output

Numeric.

Examples

Basic examples

Returns 0.999999999999999 (the tangent of the specified number of *radians*, equivalent to 45 degrees):

```
TAN(0.785398163397448)
```

Returns 0.999999999999999 (the tangent of 45 degrees):

```
TAN(45 * PI( )/180)
```

Advanced examples

Using degrees as input

Returns 1.000 (the tangent of 45 degrees, rounded to 3 decimal places):

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
DEC(TAN(45 * PI( )/180),3)
```

Remarks

Performing the Mantissa Arc Test

The three trigonometric functions in Analytics – SIN( ), COS( ), and TAN( ) – support performing the Mantissa Arc Test associated with Benford's Law.

Converting degrees to radians

If your input is in degrees you can use the PI( ) function to convert the input to radians: (*degrees* * PI ( )/180) = *radians*. If required, you can round or truncate the return value using the DEC( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

TEST( ) function

Returns a logical value indicating whether a specified string occurs at a specific byte position in a record.

Syntax

```
TEST(byte_position, string)
```

Parameters

Name	Type	Description
byte_position	numeric	The sequential number from the left in the table layout that identifies the location of the first character of *string*. The function evaluates to F if the start of *string* is not identified at this position, even if *string* appears at another position in the record.
string	character	The character string to search for. The search is case-sensitive. If there is a chance the case may be mixed, use the UPPER( ) function to convert all characters to upper-case.

Output

Logical. Returns **T** (true) if the specified string starts at the specified byte location within a record, and **F** (false) otherwise.

Examples

Basic examples

Given a record containing:

```
  Department: Marketing
....|....|....|....|....|
```

Returns T:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
TEST(5, "Department")
```

Returns F, because in the record, "Department" starts at the fifth byte position, not the sixth:

```
TEST(6, "Department")
```

Returns F, because the function is case-sensitive:

```
TEST(5, "DEPARTMENT")
```

Advanced examples

Isolating records that are page headings

Use TEST( ) to create a filter that isolates all records that start with "Page:":

```
SET FILTER TO TEST(1, "Page:")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

TIME( ) function

Extracts the time from a specified time or datetime and returns it as a character string. Can also return the current operating system time.

Syntax

```
TIME(<time/datetime> <,format>)
```

Parameters

Name	Type	Description
time/datetime optional	datetime	The field, expression, or literal value to extract the time from. If omitted, the current operating system time is returned in the format hh:mm:ss.
format optional	character	The format to apply to the output string, for example "hh:mm:ss". If omitted, the current Analytics time display format is used. You cannot specify *format* if you have omitted *time/datetime*.

Output

Character.

Examples

Basic examples

Literal input values

Returns "23:59:59" assuming an Analytics time display format of hh:mm:ss:

```
TIME(`20141231 235959`)
```

Returns "11:59 P":

```
TIME(`20141231 235959`, "hh:mm A")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns the current operating system time returned as a character string in hh:mm:ss format (24-hour clock):

```
TIME()
```

Field as input values

Returns a character string for each value in the **Receipt_timestamp** field, using the current Analytics time display format:

```
TIME(Receipt_timestamp)
```

Returns a character string for each value in the **Receipt_timestamp** field, using the specified time display format:

```
TIME(Receipt_timestamp, "hh:mm:ss")
```

Advanced examples

Calculating the elapsed time for a command or a script to execute

Use the TIME( ) function to help calculate the amount of time a particular Analytics command, or an entire script, takes to execute.

Immediately before the command you want to time, or at the start of the script, specify this line to create a variable that stores the current operating system time:

```
ASSIGN Time_started = TIME()
```

Immediately after the command, or at the end of the script, specify the two lines below.

The first line creates a variable that stores the operating system time after the command or script completes. The second line calculates the difference between the finish and start times, and converts the result to an easily readable format.

> **Tip**
>
> You can double-click the CALCULATE log entry to see the elapsed time for the command or the script.

```
ASSIGN Time_finished = TIME()
CALCULATE STOT(CTOT(Time_finished) - CTOT(Time_started))
```

If the command or script will run over the boundary of midnight, use this second line instead:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
CALCULATE `T000000` - (CTOT(Time_started) - CTOT(Time_finished))
```

Remarks

Output string length

The length of the output string is always 14 characters. If the specified output *format*, or the Analytics time display format, is less than 14 characters, the output string is padded with trailing blank spaces.

Parameter details

A field specified for *time/datetime* can use any time or datetime format, as long as the field definition correctly defines the format.

If you use *format* to control how the output string is displayed, you are restricted to the formats in the table below. You can use any combination of time and AM/PM formats. The AM/PM format is optional, and is placed last.

Specify *format* using single or double quotation marks. For example: "hh:mm:ss AM".

Time formats	AM/PM formats	Examples
hh:mm:ss	none 24-hour clock	"hh:mm:ss"
hhmmss	AM, or PM 12-hour clock	"hhmmss PM"
hh:mm	A, or P 12-hour clock	"hh:mm A"
hhmm		
hh		

Specifying a literal time or datetime value

When specifying a literal time or datetime value for *time/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231 235959` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Time values** – you can use any of the time formats listed in the table below. You must use a separator before a standalone time value for the function to operate correctly. Valid separators are the letter 't', or the letter 'T'. You must specify times using the 24-hour clock. Offsets from Coordinated Universal

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

- **Datetime values** - you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.

Example formats	Example literal values
thhmmss	`t235959`
Thhmm	`T2359`
YYYYMMDD hhmmss	`20141231 235959`
YYMMDDthhmm	`141231t2359`
YYYYMMDDThh	`20141231T23`
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`20141231 235959-0500`
YYMMDD hhmm+/-hh (UTC offset)	`141231 2359+01`
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Related functions

If you need to return the current operating system time as a datetime value, use NOW( ) instead of TIME( ).

Other datetime conversion functions

Datetime to Character conversion

Function	Description
DATE( )	Extracts the date from a specified date or datetime and returns it as a character string. Can also return the current operating system date.
DATETIME( )	Converts a datetime to a character string. Can also return the current operating system datetime.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Character or Numeric to Datetime conversion

Function	Description
CTOD( )	Converts a character or numeric date value to a date. Can also extract the date from a character or numeric datetime value and return it as a date. Abbreviation for "Character to Date".
CTODT( )	Converts a character or numeric datetime value to a datetime. Abbreviation for "Character to Datetime".
CTOT( )	Converts a character or numeric time value to a time. Can also extract the time from a character or numeric datetime value and return it as a time. Abbreviation for "Character to Time".

Serial to Datetime conversion

Function	Description
STOD( )	Converts a serial date - that is, a date expressed as an integer - to a date value. Abbreviation for "Serial to Date".
STODT( )	Converts a serial datetime - that is, a datetime expressed as an integer, and a fractional portion of 24 hours - to a datetime value. Abbreviation for "Serial to Datetime".
STOT( )	Converts a serial time - that is, a time expressed as a fractional portion of 24 hours, with 24 hours equaling 1 - to a time value. Abbreviation for "Serial to Time".

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

TODAY( ) function

Returns the current operating system date as a Datetime data type.

Syntax

```
TODAY()
```

Parameters

This function does not have any parameters.

Output

Datetime.

Examples

Basic examples

Returns the current operating system date as a datetime value, for example `20141231`, displayed using the current Analytics date display format:

```
TODAY()
```

Remarks

Related functions

If you need to return the current operating system date as a character string, use DATE( ) instead of TODAY( ).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

TRANSFORM( ) function

Reverses the display order of bi-directional text within a specified string.

Syntax

```
TRANSFORM(original_string)
```

Parameters

Name	Type	Description
original_string	character	The field, expression, or literal value containing bi-directional text.

Output

Character.

Examples

Basic examples

In the input string, the characters "XZQB" represent Hebrew/bidirectional characters in an input string that otherwise contains regular characters.

In the output string, the direction of "XZQB" is reversed, and returns "BQZX". The other characters are unmodified.

Returns "ABC BQZX 123":

```
TRANSFORM("ABC XZQB 123")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

How it works

The TRANSFORMS( ) function identifies bi-directional data and displays it correctly in the view, from right to left.

All other characters processed by the function are unmodified and continue to display from left to right.

When to use TRANSFORMS( )

Use TRANSFORMS( ) to adjust the display order of Arabic or Hebrew characters, so they display correctly.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

TRIM( ) function

Returns a string with trailing spaces removed from the input string.

Syntax

```
TRIM(string)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to remove trailing spaces from.

Output

Character.

Examples

Basic examples

Note that in both examples the leading spaces and spaces between words are not removed by the TRIM( ) function.

Returns " Vancouver":

```
TRIM("  Vancouver  ")
```

Returns " New York":

```
TRIM("  New York")
```

Advanced examples

Removing non-breaking spaces

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Non-breaking spaces are not removed by the TRIM( ) function.

If you need to remove trailing non-breaking spaces, create a computed field using the following expression:

```
DEFINE FIELD Description_cleaned COMPUTED TRIM(REPLACE(Description, CHR(160), CHR(32)))
```

The REPLACE( ) function replaces any non-breaking spaces with regular spaces, and then TRIM( ) removes any trailing regular spaces.

Remarks

How it works

The TRIM( ) function removes trailing spaces only. Spaces inside the string, and leading spaces, are not removed.

Related functions

TRIM( ) is related to the LTRIM( ) function, which removes any leading spaces from a string, and to the ALLTRIM( ) function, which removes both leading and trailing spaces.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

UNSIGNED( ) function

Returns numeric data converted to the Unsigned data type.

Syntax

```
UNSIGNED(number, length_of_result)
```

Parameters

Name	Type	Description
number	numeric	The value to convert.
length_of_result	numeric	The number of bytes to use in the output string.

Output

Numeric.

Examples

Basic examples

Returns 000075:

```
UNSIGNED(75, 3)
```

```
UNSIGNED(-75, 3)
```

```
UNSIGNED(7.5, 3)
```

Returns 2456 (1 is truncated because only 4 digits can be stored when the *length_of_result* is 2):

```
UNSIGNED(12456, 2)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns 000000012456:

```
UNSIGNED(-12.456, 6)
```

Remarks

What is Unsigned data?

The Unsigned data type is used by mainframe operating systems to store numeric values in a format that uses minimal space, storing two digits in each byte. The Unsigned data type is the same as the Packed data type, but it does not use the last byte to specify whether the value is positive or negative.

When to use UNSIGNED( )

Use the UNSIGNED( ) function to convert numeric data to the Unsigned format for export to mainframe systems.

Truncated return values

If the *length_of_result* value is shorter than the length of the *number* value, the additional digits are truncated.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

UPPER( ) function

Returns a string with alphabetic characters converted to uppercase.

Syntax

```
UPPER(string)
```

Parameters

Name	Type	Description
string	character	The field, expression, or literal value to convert to uppercase.

Output

Character.

Examples

Basic examples

Returns "ABC":

```
UPPER("abc")
```

Returns "ABC 123 DEF":

```
UPPER("abc 123 DEF")
```

Returns "ABCD 12":

```
UPPER("AbCd 12")
```

Returns all the values in the **Last_Name** field converted to uppercase:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
UPPER(Last_Name)
```

Remarks

How it works

The UPPER( ) function converts all alphabetic characters in *string* to uppercase. All non-alphabetic characters are left unchanged.

When to use UPPER( )

Use UPPER( ) when you need to ensure that all characters in a field, variable, or expression have consistent casing. Consistent casing is particularly important when you are comparing values.

UPPER( ) can also be used for formatting values in reports as uppercase text.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

UTOD( ) function

Converts a Unicode string containing a formatted date to an Analytics date value. Abbreviation for "Unicode to Date".

> **Note**
>
> This function is specific to the Unicode edition of Analytics. It is not a supported function in the non-Unicode edition.
>
> Use this function when working with dates in languages and formats that are different from your default installation. If the string you want to convert is in your default language, use CTOD( ) instead.

Syntax

```
UTOD(string<,locale><,style>)
```

Parameters

Name	Type	Description
string	character	The Unicode string to convert to a date. The Unicode string can contain a datetime value, but the time portion of the value is ignored. Standalone time values are not supported. *string* must match the input format required by the *style* value for the date's locale.
locale optional	character	The code that specifies the language and locale of the output string, and optionally the version of the language associated with a particular country or region. For example, `"zh"` specifies Chinese, and `"pt_BR"` specifies Brazilian Portuguese. If omitted, the default locale for your computer is used. If a language is specified, but no country is specified, the default country for the language is used. You cannot specify *locale* if you have not specified *date*. For more information about locale codes, see www.unicode.org.
style optional	numeric	The date format style to use for the Unicode string. The format style matches the standard for the locale you specify: ◦ `0` - full specification format, such as "Sunday, September 18, 2016" ◦ `1` - long format, such as "September 18, 2016"

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Type	Description
		◦ 2 - medium format, such as "Sep 18, 2016" ◦ 3 - short, numeric format such as "9/18/16" If omitted, the default value of **2** is used. You cannot specify *style* if you have not specified *locale*. **Tip** For help determining the expected format of your input string, do one of the following: • Use the DTOU( ) function to generate an example value using the style and locale. In the command line, use the DISPLAY command to print the value: `DISPLAY DTOU(`20160909`, "es_MX", 3)` • Consult an authoritative source on the standard date format for the style in the specific locale.

Output

Datetime. The date value is output using the current Analytics date display format.

Examples

Basic examples

> **Note**
>
> All examples assume a current Analytics date display format of DD MMM YYYY.
>
> In the examples below, the locale code for Chinese ("zh") and Simplified Chinese ("zh_CN") match different input strings and are not interchangeable.
>
> You must also specify the correct *style*. A long Unicode date string (that is, *style* is 1) does not return an Analytics date if you specify a *style* of 2.

Literal input values

Returns `20141231` displayed as 31 Dec 2014:

```
UTOD("31 de dezembro de 2014", "pt_BR", 1)
```

Returns `20141231` displayed as 31 Dec 2014:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
UTOD("31 grudnia 2014", "pl", 1)
```

Field input values

Returns the date equivalent for each Unicode string in the **Invoice_date** field:

```
UTOD(Invoice_date, "zh", 1)
```

Input uses full date style

Returns `20141231` displayed as 31 Dec 2014 (no region identifier specified):

```
UTOD("星期三, 2014 十二月 31", "zh", 0)
```

Returns `20141231` displayed as 31 Dec 2014 (region identifier specified):

```
UTOD("2014年12月31日星期三", "zh_CN", 0)
```

Input uses long date style

Returns `20141231` displayed as 31 Dec 2014 (no region identifier specified):

```
UTOD("2014 十二月 31", "zh", 1)
```

Returns `20141231` displayed as 31 Dec 2014 (region identifier specified):

```
UTOD("2014年12月31日", "zh_CN", 1)
```

Remarks

Converting Unicode strings successfully

To successfully convert Unicode strings containing dates into Analytics dates you must specify *locale* and *style* parameters that match the language country/region (if applicable), and style of the date in the Unicode string.

Related functions

UTOD( ) is the inverse of DTOU( ), which converts a date to a Unicode string. If you are uncertain which country/region and style to specify for UTOD( ), you can use DTOU( ) and experiment with different para-

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

meters to produce an output Unicode string that matches the form of the input Unicode strings you want to convert with UTOD( ).

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

VALUE( ) function

Converts a character string to a numeric value.

Syntax

```
VALUE(string, decimals)
```

Parameters

Name	Type	Description
string	character	The field, literal, or expression to convert.
decimals	numeric	The number of decimal places to include in the output.

Output

Numeric.

Examples

Basic examples

Returns -123.400:

```
VALUE("123.4-", 3)
```

Returns 123456.00:

```
VALUE("$123,456", 2)
```

Returns -77.45:

```
VALUE("77.45CR", 2)
```

Returns -123457:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
VALUE(" (123,456.78)", 0)
```

Field input

Returns character values in the **Salary** field as numbers without any decimal places:

```
VALUE(Salary, 0)
```

Remarks

How it works

This function converts character data to numeric data. You can use the VALUE( ) function if you need to convert character expressions or field values to numeric values for use in Analytics commands.

Numeric input formatting

VALUE( ) accepts numbers in any format. You can use any numeric formatting accepted by the Print data type such as punctuation, leading or trailing signs, and parentheses as input.

Negative values

The VALUE( ) function can interpret different indicators of negative values such as parentheses and the minus sign. It can also interpret CR (credit) and DR (debit). For example:

Returns -1000.00:

```
VALUE("(1000)", 2)
```

```
VALUE("1000CR", 2)
```

Decimal vs integer values

If the *string* value does not include decimals, Analytics treats the number as an integer. For example:

Returns 123.00:

```
VALUE("123", 2)
```

If the number of decimals specified by *decimals* is less than the number in the field or expression, the result is rounded. For example:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Returns "10.6":

```
VALUE("10.56", 1)
```

Related functions

The VALUE( ) function is the opposite of the STRING( ) function, which converts numeric data to character data.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

VERIFY( ) function

Returns a logical value indicating whether the data in a physical data field is valid.

Syntax

```
VERIFY(field)
```

Parameters

Name	Type	Description
field	character numeric datetime	Must be a physical data field.

Output

Logical. Returns **T** (true) if data in the field is valid, and **F** (false) otherwise.

Examples

Basic examples

Extracts any records where the VERIFY( ) function evaluates to false to a new Analytics table:

```
EXTRACT RECORD IF NOT VERIFY(Address) TO "InvalidEntries.fil"
```

Remarks

The VERIFY( ) function determines whether the data in a field is consistent with the specified data type for the field.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

When to use VERIFY( )

The function provides more precise control over the fields you want to verify than either the VERIFY command or the **Verify Data** option in the **Numeric** tab in the **Options** dialog box (**Tools > Options**). You can use the function to detect errors in individual fields and proceed in a situation-specific manner.

When the function evaluates to true

For the function to evaluate to true:

- character fields must contain only valid, printable characters, such as letters, numbers, and symbols
- numeric fields must contain only valid numeric characters, such as numbers, decimals, and currency symbols
- datetime fields must contain only valid dates, datetimes, or times

Computed fields and expressions

Computed fields and expressions always evaluate to **T** (true), so this function cannot be used with computed fields or expressions unless they are first converted to physical fields. Using the **Fields** option in the **Extract** dialog box to extract computed fields or expressions converts them to physical fields.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

WORKDAY( ) function

Returns the number of workdays between two dates.

Syntax

```
WORKDAY(start_date, end_date <,nonworkdays>)
```

Parameters

Name	Type	Description
start_date	datetime	The start date of the period for which workdays are calculated. The start date is included in the period.
end_date	datetime	The end date of the period for which workdays are calculated. The end date is included in the period.
nonworkdays optional	character	The days of the week that are weekend days, or non workdays, and excluded from the calculation. If *nonworkdays* is omitted, Saturday and Sunday are used as the default non workdays. Enter *nonworkdays* using the following abbreviations, separated by a space or a comma: ◦ Mon ◦ Tue ◦ Wed ◦ Thu ◦ Fri ◦ Sat ◦ Sun *nonworkdays* is not case-sensitive. The abbreviations must be entered in English even if you are using a non-English version of Analytics: `"Fri, Sat, Sun"`

> **Note**
>
> You can specify a datetime value for *start_date* or *end_date* but the time portion of the value is ignored.
>
> If *start_date* is more recent than *end_date*, a negative value is returned.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Output

Numeric. The number of workdays in the period for which workdays are calculated.

Examples

Basic examples

Literal input values

Returns 5 (the number of workdays between Monday, March 02, 2015 and Sunday, March 08, 2015 inclusive):

```
WORKDAY(`20150302`, `20150308`)
```

Returns 6 (the number of workdays between Monday, March 02, 2015 and Sunday, March 08, 2015 inclusive, when Sunday is the only non workday):

```
WORKDAY(`20150302`, `20150308`, "Sun")
```

Returns 5 (the number of workdays between Sunday, March 01, 2015 and Saturday, March 07, 2015 inclusive, when Friday and Saturday are the non workdays):

```
WORKDAY(`20150301`, `20150307`, "Fri, Sat")
```

You can also use the function to calculate the number of weekend days in a date range.

Returns 2 (the number of weekend days between Monday, March 02, 2015 and Sunday, March 08, 2015 inclusive):

```
WORKDAY(`20150302`, `20150308`, "Mon, Tue, Wed, Thu, Fri")
```

Field input values

Returns the number of workdays between each date in the **Start_date** field and December 31, 2015 inclusive:

```
WORKDAY(Start_date, `20151231`)
```

Returns the number of workdays between each date in the **Start_date** field and a corresponding date in the **End_date** field inclusive:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- Statutory holidays are included in the workdays total and may need to be factored out using a separate calculation
- A negative return value indicates a start date that is more recent than an end date

```
WORKDAY(Start_date, End_date)
```

Remarks

Date formats

A field specified for *start_date* or *end_date* can use any date format, as long as the field definition correctly defines the format.

When specifying a literal date value for *start_date* or *end_date*, you are restricted to the formats `YYYYMMDD` and `YYMMDD`, and you must enclose the value in backquotes – for example, `` `20141231` ``.

Non workdays other than Saturday and Sunday

The ability to specify non workdays other than Saturday and Sunday allows you to use the WORKDAY( ) function with data that is not based on a Monday-to-Friday work week, or on a five-day work week.

For example, if you specify `"Sun"` by itself as a non workday, you create a six-day work week from Monday to Saturday.

Accounting for statutory holidays

The WORKDAY( ) function does not take into account statutory holidays, which means that the return value may not reflect the actual number of workdays in a period if the period contains one or more statutory holidays.

"Calculate Working Days" script in ScriptHub

If you need to account for statutory holidays, one option is to use the Calculate Working Days script in ScriptHub, which accepts a list of user-defined holidays.

For data that covers longer time periods and includes a number of holidays, using the script is probably the easier approach. For more information, see "Importing scripts from ScriptHub" in the Analytics Help.

For shorter time periods with only three or four holidays, such as a quarter, you may find creating the conditional computed field described below is not too laborious.

Conditional computed field for deducting statutory holidays

If required, you can create a conditional computed field to deduct statutory holidays from the value returned by the WORKDAY( ) function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

For example, for first-quarter data for 2015, you could decrement the WORKDAY( ) return value by 1 for each of these holidays that falls into a specified period:

- 01 January 2015
- 19 January 2015
- 16 February 2015

The example that follows accommodates periods that have any start date and end date during the quarter.

You first create a computed field, for example **Workdays**, that calculates the workdays for a specified period during the quarter:

```
DEFINE FIELD Workdays COMPUTED WORKDAY(Start_date, End_date)
```

You then create a conditional computed field, for example **Workdays_no_holidays**, that adjusts the value returned by the first computed field (**Workdays**):

```
DEFINE FIELD Workdays_no_holidays COMPUTED

Workdays-1 IF Start_date = `20150101` AND End_date < `20150119`
Workdays-2 IF Start_date = `20150101` AND End_date < `20150216`
Workdays-3 IF Start_date = `20150101` AND End_date <= `20150331`
Workdays   IF Start_date < `20150119` AND End_date < `20150119`
Workdays-1 IF Start_date < `20150119` AND End_date < `20150216`
Workdays-2 IF Start_date < `20150119` AND End_date <= `20150331`
Workdays-1 IF Start_date = `20150119` AND End_date < `20150216`
Workdays-2 IF Start_date = `20150119` AND End_date <= `20150331`
Workdays   IF Start_date < `20150216` AND End_date < `20150216`
Workdays-1 IF Start_date < `20150216` AND End_date <= `20150331`
Workdays-1 IF Start_date = `20150216` AND End_date <= `20150331`
Workdays   IF Start_date < `20150331` AND End_date <= `20150331`
Workdays
```

> **Note**
>
> The order of the conditions in the conditional computed field is important.
>
> Analytics evaluates multiple conditions starting at the top. The first condition that evaluates to true for a record assigns the value of the conditional computed field for that record. A subsequent condition that evaluates to true does not change the assigned value.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

YEAR( ) function

Extracts the year from a specified date or datetime and returns it as a numeric value using the YYYY format.

Syntax

```
YEAR(date/datetime)
```

Parameters

Name	Type	Description
date/datetime	datetime	The field, expression, or literal value to extract the year from.

Output

Numeric.

Examples

Basic examples

Returns 2014:

```
YEAR(`20141231`)
```

```
YEAR(`141231 235959`)
```

Returns the year for each value in the **Invoice_date** field:

```
YEAR(Invoice_date)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Parameter details

A field specified for *date/datetime* can use any date or datetime format, as long as the field definition correctly defines the format.

Specifying a literal date or datetime value

When specifying a literal date or datetime value for *date/datetime*, you are restricted to the formats in the table below, and you must enclose the value in backquotes – for example, `` `20141231` ``.

Do not use any separators such as slashes (/) or colons (:) between the individual components of dates or times.

- **Datetime values** – you can use any combination of the date, separator, and time formats listed in the table below. The date must precede the time, and you must use a separator between the two. Valid separators are a single blank space, the letter 't', or the letter 'T'.
- **Time values** – you must specify times using the 24-hour clock. Offsets from Coordinated Universal Time (UTC) must be prefaced by a plus sign (+) or a minus sign (-).

Example formats	Example literal values
YYYYMMDD	`` `20141231` ``
YYMMDD	`` `141231` ``
YYYYMMDD hhmmss	`` `20141231 235959` ``
YYMMDDthhmm	`` `141231t2359` ``
YYYYMMDDThh	`` `20141231T23` ``
YYYYMMDD hhmmss+/-hhmm (UTC offset)	`` `20141231 235959-0500` ``
YYMMDD hhmm+/-hh (UTC offset)	`` `141231 2359+01` ``
Note Do not use hh alone in the main time format with data that has a UTC offset. For example, avoid: hh+hhmm. Results can be unreliable.	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ZONED( ) function

Converts numeric data to character data and adds leading zeros to the output.

Syntax

```
ZONED(number, length)
```

Parameters

Name	Type	Description
number	numeric	The numeric value to convert to a string. **Note** If you want to add leading zeros to a character value that contains a numeric string, you must use the VALUE( ) function to convert the character to the numeric data type before using the value as input for ZONED( ). For more information, see "VALUE( ) function" on page 835.
length	numeric	The length of the output string.

Output

Character.

Examples

Basic examples

Integer input

Returns "235":

```
ZONED(235, 3)
```

Returns "00235", because *length* is greater than the number of digits in *number* so two leading zeros are added to the result:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
ZONED(235, 5)
```

Returns "35", because *length* is less than the number of digits in *number* so the leftmost digit is truncated from the result:

```
ZONED(235, 2)
```

Decimal input

Returns "23585", because the zoned data format does not support a decimal point:

```
ZONED(235.85, 5)
```

Negative input

Returns "64489M", because the number is negative and "M" represents the final digit 4:

```
ZONED(-6448.94, 6)
```

Returns "489J", because *length* is less than the number of digits in *number* so the two leftmost digits are truncated from the result, and the number is negative and "J" represents the final digit 1:

```
ZONED(-6448.91, 4)
```

Advanced examples

Adding leading zeros to a character field containing numbers

The Employee_Number field contains the value "254879". You need to convert the value to a 10-digit string with leading zeros.

> **Tip**
>
> You must use the VALUE( ) function to convert the character to numeric data before using the numeric as input for ZONED( ).

```
COMMENT returns "0000254879"
ASSIGN v_str_length = 10
ASSIGN v_num_decimals = 0
ZONED(VALUE(Employee_Number, v_num_decimals), v_str_length)
```

Harmonizing a key field when joining tables

You have two tables, **Ar** and **Customer**, and you need to join them on the **CustNo** field for further analysis.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

The two tables each have a **CustNo** field, but the data format is different:

- **Ar** - numeric field (for example, 235)
- **Customer** - 5 character field that pads numbers with leading zeros (for example, "00235")

To harmonize the fields when joining so that the data types and lengths are equal, you use the ZONED( ) function to convert the **Ar** key field **CustNo** to a character field of length 5 so that it matches the format of the key field in **Customer**:

```
OPEN Ar PRIMARY
OPEN Customer SECONDARY
JOIN PKEY ZONED(CustNo,5) FIELDS CustNo Due Amount SKEY CustNo UNMATCHED TO Ar_
Cust OPEN PRESORT SECSORT
```

Remarks

How it works

This function converts numeric data to character data and adds leading zeros to the output. The function is commonly used to harmonize fields that require leading zeros, for example, check number, purchase order number, and invoice number fields.

When to use ZONED( )

Use the function to convert a positive numeric value to a character value containing leading zeros. This is useful for normalizing data in fields to be used as key fields.

For example, if one table contains invoice numbers in the form 100 in a numeric field, and another table contains invoice numbers in the form "00100" in a character field, you can use ZONED( ) to convert the numeric value 100 to the character value "00100". This allows you to compare like invoice numbers.

String length and return values

Leading zeros are added to the output value when the *length* value is more than the number of digits in *number*. When *length* is less than the number of digits in *number*, the output is truncated from the left side. If the *number* value is the same length as *length*, then no zeros are added.

Decimal numbers

The zoned data format does not include an explicit decimal point.

Negative numbers

If the input *number* is negative, the rightmost digit is displayed as a character in the result:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- "}" for 0
- a letter between "J" and "R" for the digits 1 to 9

ZONED( ) and the Unicode edition of Analytics

If you are working with the Unicode edition of Analytics, you need to use the BINTOSTR( ) function to correctly display the return value of the ZONED( ) function. You also need to use the BINTOSTR( ) function if you want to use the return value of the ZONED( ) function as a parameter in another function.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ZSTAT( ) function

Returns the standard Z-statistic.

Syntax

ZSTAT(*actual*, *expected*, *population*)

Parameters

Name	Type	Description
actual	numeric	◦ **When specifying parameters as numbers** - represents the actual count, such as a leading digit or a leading digit combination. ◦ **When specifying parameters as proportions** - represents the expected proportion of the value being tested and must be between 0 and 1 inclusive (i.e., greater than or equal to 0 and less than or equal to 1).
expected	numeric	◦ **When specifying parameters as numbers** - represents the expected count, such as a leading digit or a leading digit combination. ◦ **When specifying parameters as proportions** - represents the expected proportion of the value being tested and must be between 0 and 1 exclusive (i.e., greater than 0 and less than 1).
population	numeric	The total number of items being tested. This parameter must be a positive whole number greater than 0.

Output

Numeric.

Examples

Advanced examples

Parameters expressed as numbers

Based on 10 years of previous data, you know that the distribution of worker disability claims per month is

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

normally highly uniform. In April, May, and June of this year, claims were higher by about 10 percent, averaging 220 per month instead of 200. Claims in July and August were slightly low, at 193 and 197. The total claims for the year were 2,450. To test whether these high and low results were significant, use the Z-statistic.

The actual number of claims for April to June is higher than expected at 660. The expected number of claims for this period is 25 percent of the 2,450 annual claims, or 612.5. The Z-statistic for these counts is calculated as 2.193:

```
ZSTAT(660, 612.5, 2450)
```

A Z-statistic of 1.96 has a significance of 0.05, and 2.57 has a significance of 0.01. Thus, the probability that the higher rates of claims are due to chance is between 1:20 and 1:100.

The actual number of claims for July and August is lower than expected at 390. The expected number of claims for this period is one sixth of the 2,450 annual claims, or 408.33. The Z-statistic for these proportions is calculated as 0.967:

```
ZSTAT(390, 408.33, 2450)
```

This is not a very significant result. Z-statistics of 1.000 and less are very common and can typically be ignored.

Parameters expressed as proportions

Based on 10 years of previous data, you know that the distribution of worker disability claims per month is normally highly uniform. In April, May, and June of this year, claims were higher by about 10 percent, averaging 220 per month instead of 200. Claims in July and August were slightly low, at 193 and 197. The total claims for the year were 2,450. To test whether these high and low results were significant, use the Z-statistic.

The actual number of claims for April to June is represented by the proportion 660/2450, which is higher than expected. The expected number of claims for this period should be 25 percent of the 2,450 annual claims. The Z-statistic for these proportions is 2.193:

```
ZSTAT((1.00000000 * 660 / 2450), 0.25, 2450)
```

A Z-statistic of 1.96 has a significance of 0.05, and 2.57 has a significance of 0.01. Thus, the probability that the higher rates of claims are due to chance is between 1:20 and 1:100.

The actual number of claims for July and August is low at 390. The expected number of claims for this period should be one sixth, or 16.6667 percent of the 2,450 annual claims. The Z-statistic for these proportions is 0.967:

```
ZSTAT((1.00000000 * 390 / 2450), 0.16667, 2450)
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

This is not a very significant result. Z-statistics of 1.000 and less are very common and can typically be ignored.

Remarks

How it works

The ZSTAT( ) function calculates the standard Z-statistic for use in many problem-solving tasks, including digital analysis. It outputs the result with a precision of three decimal places.

Using ZSTAT( )

Use ZSTAT( ) to evaluate the likely frequency of occurrence of a given result in a specified period or category. The larger the resulting Z-statistic, the more unlikely the occurrence.

For example, a Z-statistic of 1.96 has a significance of 0.05, representing the likelihood of a one time in 20 occurrence, whereas a Z-statistic of 2.57 has a significance of 0.01, representing the likelihood of a one time in 100 occurrence. For information on the Z-statistic, consult a statistics textbook.

Specifying input for ZSTAT( )

You can specify the parameters for ZSTAT( ) as either numbers or proportions:

- When you specify both input values as numbers, the function computes the Z-statistic using floating-point arithmetic
- When you specify both input values as proportions, the function computes the Z-statistic using fixed-point arithmetic, and you need to use a decimal multiplier to control rounding
- When using an expression within an expression to calculate the *actual* or *expected* value, you must specify the level of precision you want in the result by using a decimal multiplier. Analytics has a precision of 8 digits, therefore a multiplier of 1.00000000 will return the greatest precision attainable

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytic scripts

Scripts are not limited to running in Analytics only. By converting regular scripts to **analytic scripts**, you can schedule and run scripts in the Robots app on the HighBond platform, or in Analytics Exchange. You can also run analytic scripts in the Analysis App window, a freestanding component of Analytics.

What are analytic scripts?

An analytic script, or "an analytic", is a regular script **with an analytic header**. The analytic header is a series of declarative tags that allow the script to run in Robots, on AX Server, or in the Analysis App window. The analytic header includes input parameters that a user populates in advance, which allows the analytic script to run unattended, either immediately, or at a scheduled time.

> **Tip**
>
> Analytic scripts are almost exclusively developed and tested in Analytics, which supports easier development. Use AX Client to make simple updates to existing analytic scripts that are stored on AX Server.

What are analysis apps?

An **analysis app** is an Analytics project that is packaged for use in Analytics Exchange or the Analysis App window. Analysis apps contain one or more analytic scripts, and can also contain data tables and interpretations.

> **Note**
>
> Analysis apps are typically created or developed by an organization's in-house scripting experts, or by arrangement with Galvanize consultants.

Turning regular scripts into analytic scripts

Analytic scripts begin as regular scripts. To run a regular script in Robots, on AX Server, or in the Analysis App window, you must convert the regular script into an analytic script:

1. Create and test a script in Analytics.
2. Add the appropriate analytic header tags to make the script an analytic script.
3. Package the analytic script to run on AX Server or in the Analysis App window. You do not package analytic scripts run in Robots.

For more information, see "Developing analytic scripts" on page 857.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Adding analytic headers

Analytic headers are defined in a comment block that starts on the first line of the script. At a minimum, an analytic header declares the script is an analytic script:

```
COMMENT
//ANALYTIC Identify missing checks
This analytic script identifies missing check numbers
END
```

For more information, see "Working with analytic headers" on page 863.

What are auxiliary scripts?

An auxiliary script is a regular script **without an analytic header** that is designed to work in conjunction with an analytic script. In a typical design, an analytic script uses the DO SCRIPT command to call one or more auxiliary scripts. Once an auxiliary script completes, processing returns to the analytic script, which continues to execute.

Auxiliary scripts can also be referred to as subscripts, secondary scripts, utility scripts, or helper scripts. You do not have to use auxiliary scripts. They offer an option for compartmentalizing blocks of script logic that might be conditional, reusable, or simply unwieldy to include in the parent analytic script.

Auxiliary script restrictions

Auxiliary scripts can be used in a variety of different ways, but because no analytic header exists, two restrictions apply:

- **No input or output tags** – you cannot specify input or output analytic tags, which means you cannot create input or output parameters in the auxiliary script itself. Any required parameters must be created in the analytic header in the parent analytic script.
- **Cannot be run directly** – users cannot schedule or run auxiliary scripts directly. They can only be called from an analytic script – either directly, or indirectly through another auxiliary script.

Distributing and running analytic scripts

There are several options for distributing and running analytic scripts, depending on which Galvanize products and components your organization uses.

App/product/component	Method for distributing and running an analytic script
Robots	◦ commit one or more analytic scripts, and any auxiliary scripts, as a **script version** to development mode in Robots, and schedule and run an activated script version in production mode

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

App/product/component	Method for distributing and running an analytic script
AX Server	Either of these methods: ◦ import the Analytics project (`.acl` file) directly into AX Server, and schedule and run an analytic script using AX Client ◦ package the project into a compressed analysis app file (`.aclapp` file), import it into AX Server, and run an analytic script using AX Web Client For more information, see "Packaging analysis apps" on page 877.
Analysis App window	◦ package the project into a compressed analysis app file (`.aclapp` file), open the project as an analysis app (`.aclx` file), and run the analytic script in the Analysis App window For more information, see "Packaging analysis apps" on page 877.

Determining the environment where an analytic script is running

If you want to create an analytic script that can run in Analytics, Analytics Exchange, or the Analysis App window, you can determine the runtime environment during script execution. You can use this information to make decisions about which commands to run based on where the script is running.

Use the FTYPE( ) function to determine where the script is running:

```
FTYPE("ax_main") = "b"
```

If the script is running in either Analytics Exchange or the Analysis App window, the expression evaluates to true (T). For scripts running in Analytics, the expression evaluates to false (F). For more information, see "FTYPE( ) function" on page 584.

Identifying the user running the script on AX Server

For analytic scripts run on AX Server, you can use the system variable *AXRunByUser* to identify the name of the user who is currently running the script, in the format *domain**username*:

```
EXTRACT FIELDS TIME() AS "Time", DATE() AS "Date", AXRunByUser AS "Current User" TO R_
RunRecord APPEND
```

> **Note**
>
> *AXRunByUser* is only available when running analytic scripts on AX Server. The variable is unrecognized when running scripts in Analytics.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Developing analytic scripts

The recommended method for developing an analytic script is to first create and test a regular script in Analytics. Once the script is working correctly, add the analytic header to convert the script to an analytic script. Analytic scripts can run in Robots, on AX Server, or in the Analysis App window.

For information about creating regular scripts, see "Getting started" on page 12.

Identify any script inputs and outputs

In the analytic header, you must declare any script inputs, and any script outputs that you want to make available to end users, or to use as input for subsequent scripts.

Identifying the required inputs and outputs before you begin will make development go more smoothly.

Different types of inputs and outputs are described below, with the associated analytic tags in parentheses.

Inputs	Outputs
◦ Analytics tables and fields ("TABLE" on page 892, "FIELD" on page 894) ◦ non-Analytics files such as Excel or delimited ("FILE" on page 889) ◦ input parameters such as cutoff amount, date, or ID codes ("PARAM" on page 896) ◦ passwords ("PASSWORD" on page 908)	◦ Analytics output tables that will be used as input for subsequent scripts ("DATA" on page 911) ◦ Analytics and non-Analytics results tables ("RESULT" on page 915) ◦ log files for successful scripts ("RESULT" on page 915)

Accessing source data

There are two basic approaches to accessing the source data required by an analytic script:

- Automated connectivity
- Manual upload

You are free to use both approaches in the same analytic script, if required.

Automated connectivity

The advantage of this approach is that data imports to Robots or AX Server can be fully automated including being run on a schedule.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

In the body of the analytic script, use one of the ACLScript commands for connecting to an external data source, importing data, and creating an Analytics table with a copy of the data:

- ACCESSDATA
- IMPORT ODBC
- IMPORT GRCRESULTS
- IMPORT GRCPROJECT
- DEFINE TABLE DB

> **Note**
>
> These commands do not require any corresponding analytic tag in the analytic header.
>
> Use ACCESSDATA unless you have a reason for using one of the other commands. DEFINE TABLE DB is an older command that is maintained for backward compatibility with older scripts.

Manual upload

Manual upload provides a simple way to import data to Robots or AX Server, and it may be appropriate when users have source data files stored locally.

Robots

You can manually upload non-Analytics files such as Excel or delimited to Robots. You need to use a different method to make Analytics tables available.

- **non-Analytics files** – You can manually upload non-Analytics files such as Excel or delimited to the **Input/Output** tab in a robot. To access the uploaded data in an analytic script, use a "FILE" on page 889 tag in the analytic header, and an appropriate IMPORT command, such as IMPORT EXCEL, in the body of the script.
- **Analytics tables** – You cannot manually upload Analytics tables to the **Input/Output** tab. Instead, use a "DATA" on page 911 tag in the analytic header to save an Analytics output table to the **Input/Output** tab. To access the Analytics table in a subsequent script, use the OPEN command in the body of the script.

AX Server

You can manually upload non-Analytics files such as Excel or delimited, and Analytics tables, to AX Server:

- **non-Analytics files** – You can import files such as Excel and delimited to the **Related Files** subfolder. To access the imported data in an analytic script, use a "FILE" on page 889 tag in the analytic header, and an appropriate IMPORT command, such as IMPORT EXCEL, in the body of the script.
- **Analytics tables** – When you import an Analytics project into AX Server, the project's tables are imported to the **Data** subfolder. To access an imported table in an analytic script, use the OPEN command in the body of the script.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Workflow for creating and testing an analytic script

> **Note**
>
> The following workflow is a suggested approach for developing analytic scripts, however you are free to develop analytic scripts in whatever manner best suits you.

Create the Analytics script

Create a script in Analytics without using any custom dialog boxes for user input, or any other features that require user interaction during the running of the script. Analytic scripts allow user input prior to running the script, but unlike regular scripts do not support user interaction during script execution.

To store test input values in the Analytics script, temporarily create variables at the top of the script. For example:

```
ASSIGN v_AnalysisTable = "Trans_May"
```

Test and debug the script until it executes without errors.

Add the analytic header and tags

Add an analytic header to the script. Copy the variable names from the top of the script to the corresponding tags in the Analytic Header Designer.

An example of a resulting tag in the analytic header:

```
//TABLE v_AnalysisTable "Table to classify"
```

For more information, see "Working with analytic headers" on page 863.

Include the log in the analytic script results

The log is a crucial tool for diagnosing the cause of analytic script failures. It can also be important when analytic scripts succeed but give unexpected results. The log is automatically output when an analytic script fails, but it is only output when an analytic script succeeds if you specify the RESULT analytic tag.

In the Analytic Header Designer, turn on **Keep log file** to ensure that a log is available every time the analytic script is run. The corresponding tag is added to the analytic header:

```
//RESULT LOG
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Validate the analytic header

Validate the analytic header. You can validate the analytic header as frequently as you want.

For more information, see "Validating analytic headers" on page 869.

Assign temporary test values to the analytic tags

Using the special assignment operator (:=), assign temporary test values to all analytic tags that require user input. You can copy the test values from the temporary variable assignments at the top of the script. For example:

```
//TABLE v_AnalysisTable "Table to classify" := "Trans_May"
```

To assign temporary test values using the Analytic Header Designer, enter the value in the **Test value** field for all analytic tags that require user input.

For more information about assigning temporary test values, see "Specifying test input values in Analytics" on page 885.

Delete the temporary variables

Delete the temporary variables from the top of the script, or comment them out if you think you might want to use them again.

Step through the analytic script

Step through the analytic script by clicking **Step** , or by pressing **F10** repeatedly. Review the contents of the **Variables** tab in the **Navigator** to ensure that all variables in the analytic header are being created correctly, with the correct assignment of test values.

Test and debug the analytic script until it executes without errors.

> **Note**
>
> If you want to exit the analytic script before it completes, press **Esc** and click **Yes** in the confirmation prompt.

> **Tip**
>
> You can delete all stored variables and variable assignments from the Analytics project by entering `DELETE ALL OK` in the command line. Clearing the **Variables** tab prior to stepping through an analytic script gives you a clean start.

Delete the temporary test values

When you are finished testing, you can delete the temporary test values and the special assignment operator from all analytic tags. Or you can choose to keep them if you anticipate that additional testing might be

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

required. Test values are ignored in deployment environments.

Deploy the analytic script

To deploy the analytic script to the target environment, commit the script to Robots, or import the Analytics project into AX Server.

Workflow for testing an analysis app

For analytic scripts that will be run in AX Web Client, or the Analysis App window, you also need to test the analysis app.

Delete redundant table layouts

Once all analytic scripts and any subscripts in the analysis app are tested, debugged, and executing correctly, delete any table layouts from the Analytics project that you are not going to include in the analysis app.

Redundant table layouts will clutter the analysis app in AX Client, AX Web Client, and the Analysis App window, and could be confusing to end users.

Open the analysis app in the Analysis App window

Open the completed analysis app in the Analysis App window by right-clicking the Analytics project entry in the **Overview** tab and selecting **Open as Analysis App**.

> **Note**
>
> If the analysis app fails to open and you get an error message stating that analytic scripts have identical names, check the *name* value in the ANALYTIC tag for each analytic script specified in the error message. The *name* values of analytic scripts must be unique in an Analytics project.

Run the analytic scripts

Run all the analytic scripts in the analysis app to confirm that they are functioning correctly.

Observe the correct order for running the analytic scripts if you are using the TYPE option with the ANALYTIC tag and creating import, preparation, and analysis analytic scripts.

Check the log

If an analytic script fails, open and review the log file (*analytic_name*.log). The log should include an entry, marked with a red X, that indicates why the analytic script failed:

- for incorrectly entered input values, immediately re-run the analytic script with a correctly entered input value

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- for syntax or logical errors in the script body, correct the error in Analytics, and then reopen the analysis app in the Analysis App window

An analytic script may succeed but the result table may not contain the results you were expecting. In this situation, review the log entries in sequence, and check the input values that have been passed to the analytic script, to ensure that the analytic script is functioning in the manner you intended.

Packaging and validating an analysis app

Package or import the analysis app

Once you are satisfied the analysis app is working as intended, package it for distribution and use in the Analysis App window, or import it to AX Server for use in AX Client or AX Web Client. For more information, see "Packaging analysis apps" on page 877.

Run AX Server analysis apps

If you are developing analytic scripts for use in AX Server, run all analytic scripts using both AX Client and AX Web Client to ensure they work as intended.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Working with analytic headers

An **analytic header** is a series of analytic tags enclosed in a comment block at the beginning of a script. The tags specify either script inputs or script outputs.

Tags that specify user-facing input parameters let a user specify script input values in advance, which means an analytic script can run unattended, either immediately, or at a scheduled time.

After you develop scripts in an Analytics project, you must add an analytic header to at least one script before you can commit the scripts to Robots, or use the project as an analysis app on AX Server or in the Analysis App window.

Using the Analytic Header Designer is the easiest way to add or modify an analytic header. You can also add or modify an analytic header manually.

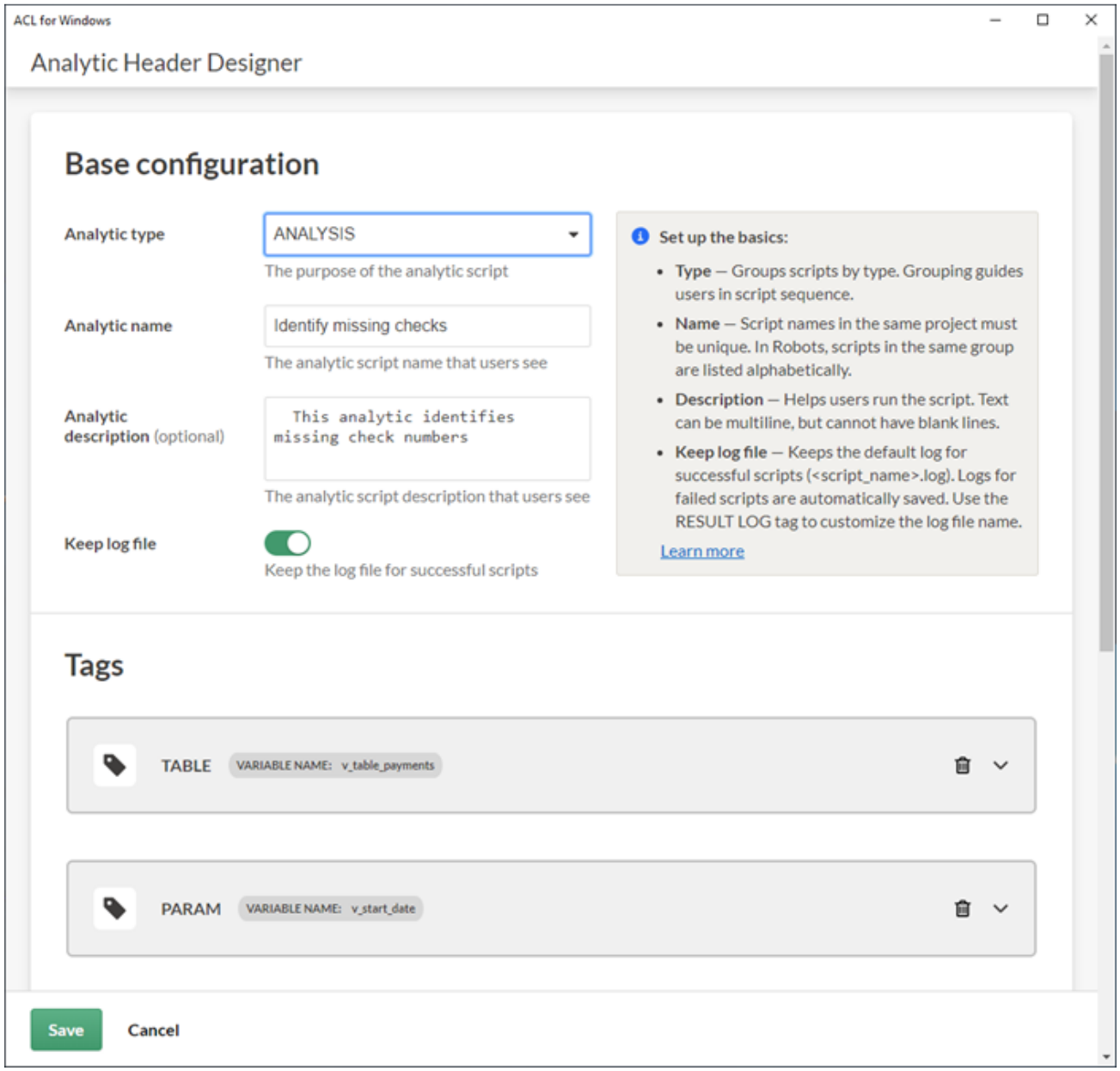
The Analytic Header Designer

The Analytic Header Designer has an intuitive interface for progressively adding the analytic tags that make up an analytic header. You are free to add, modify, or delete tags as you build an analytic header.

Automated error checking and embedded guidance in the Designer help ensure that the header you build is valid and works correctly.

When you click **Save** in the Designer, the tags that you have configured are automatically translated into an analytic header at the top of the script. You can manually edit the analytic header if you want to, but the recommended approach is to reopen the Designer to perform edits.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize



Sample analytic header

The analytic header shown below, for an analytic script that identifies missing checks, was created using the tags shown in the Analytic Header Designer above. To save space, the screen capture of the Designer is sized to show only a subset of the tags in the analytic header.

```
COMMENT
//ANALYTIC TYPE ANALYSIS Identify missing checks
  This analytic script identifies missing check numbers
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

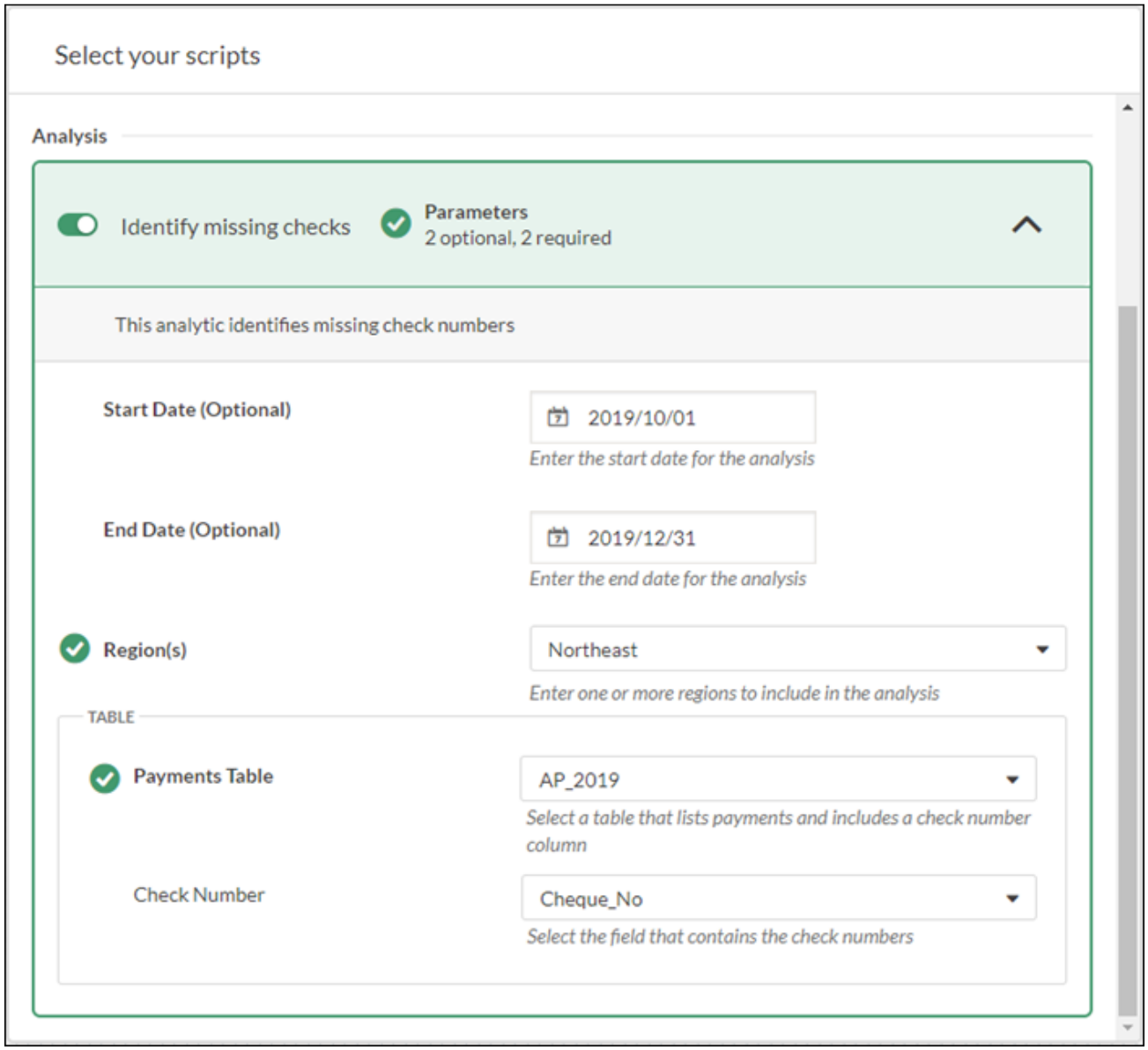
```
//TABLE v_table_payments Payments Table
  Select a table that lists payments and includes a check number column
//FIELD v_check_num CN Check Number
  Select the field that contains the check numbers
//PARAM v_start_date D OPTIONAL Start Date (Optional)
  Enter the start date for the analysis
//PARAM v_end_date D OPTIONAL End Date (Optional)
  Enter the end date for the analysis
//PARAM v_region C MULTI SEPARATOR , QUALIFIER ' VALUES |North-
east|Southeast|Central|Western|West Coast| Region(s)
  Enter one or more regions to include in the analysis
//RESULT TABLE Missing_Checks
//RESULT FILE Missing_Checks.xls
//RESULT LOG
END

COMMENT The body of the script starts here.
SET SAFETY OFF
OPEN %v_table_payments%
.
.
.
SET SAFETY ON
```

What the script inputs look like in a client application

The input tags in the sample analytic header above create input parameters that a user must populate when scheduling or running the analytic script in a client application.

The way the input parameters are displayed in Robots is shown below.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize



What each tag does

Each analytic tag in the sample analytic header above performs a specific task when a user schedules or runs the associated analytic script in a client application.

Analytic header syntax	Description
COMMENT . . . END	Encloses the block of analytic tags. Every analytic header must be enclosed by a COMMENT command that starts on the first line of the script.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytic header syntax	Description
//ANALYTIC	Creates the base configuration of the analytic header, including the type and name of the analytic script. Every analytic header must start with an //ANALYTIC tag.
//TABLE v_table_payments	Creates an input parameter that allows a user to select a payments table. Because table names vary, the name of the table selected by the user is stored in the v_table_payments variable.
//FIELD v_check_num	Creates an input parameter that allows a user to select a check number field from the payments table. Because field names vary, the name of the field selected by the user is stored in the v_check_num variable.
//PARAM v_start_date	Creates an input parameter that allows a user to specify a start date for the range of records being analyzed. Because users will specify different start dates, the actual date specified by the user is stored in the v_start_date variable.
//PARAM v_end_date	Creates an input parameter that allows a user to specify an end date for the range of records being analyzed. Because users will specify different end dates, the actual date specified by the user is stored in the v_end_date variable.
//PARAM v_region	Creates an input parameter that allows a user to specify which region or regions are included in the analysis. Because users will specify different regions, the actual regions specified by the user are stored in the v_region variable.
//RESULT TABLE Missing_Checks	Creates an output parameter that specifies the Missing_Checks results table is made available to users in client applications. Output results from scripts, even if they exist, are not automatically made available. Availability must be specified in the analytic header.
//RESULT FILE Missing_Checks.xls	Creates an output parameter that specifies the Missing_Checks.xls results file is made available to users in client applications. Output results from scripts, even if they exist, are not automatically made available. Availability must be specified in the analytic header.
//RESULT LOG	Specifies that a log file is output for scripts that run successfully. A log file is automatically output if a script fails.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Build an analytic header

To build an analytic header you must know in advance what script inputs and outputs you need. For more information, see "Identify any script inputs and outputs" on page 857.

Set up the base configuration of the analytic header

1. Open a new or existing script in the Script Editor.
2. Click **Edit Analytic Header** .

   The Analytic Header Designer opens.
3. Select an **Analytic type**.

   Analytic scripts are grouped by type in Robots, AX Web Client, and the Analysis App window. Grouping guides users in script sequence.

   - IMPORT – a script that retrieves data from a data source.
   - PREPARE – a script that transforms raw data in whatever way is necessary to make it suitable for analysis.
   - ANALYSIS – a script that performs analysis on data.
4. Specify an **Analytic name**.

   > **Note**
   >
   > Names of analytic scripts in the same Analytics project must be unique.
   >
   > The name identifies the analytic script in client applications. The analytic script name is not the same as the script name that you specify in Analytics when you initially create the script.
5. Choose whether to keep a log file for successful scripts:
   - **Keep log file on** – a log file is automatically output when the script runs successfully
   - **Keep log file off** – a log file is not output when the script runs successfully

   Regardless of the **Keep log file** setting, a log file is automatically output if the script fails.

   > **Tip**
   >
   > If you want to customize the name of the log file for successful scripts, use the RESULT LOG tag.

Add additional analytic tags

After setting up the base configuration of the analytic header, you can add as many additional analytic tags as you need.

You can add tags in any order.

1. In the Analytic Header Designer, click **Add tag**.
2. Select a **Tag type**.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

3. To configure the tag, complete all the required fields in the tag configuration section, and any optional fields that you need.

   Guidance for configuring tags is embedded in the configuration section for each tag.

   For detailed information about analytic header syntax, and a full list of analytic tags, see "Analytic headers and tags" on page 884.
4. Repeat the process for each additional tag that you need in the analytic header.
5. Click **Save** when you are finished.

Validating analytic headers

After you add an analytic header to one or more scripts, use tools in Analytics to validate the header syntax to ensure that it is correct. Perform the validation before committing scripts to Robots, or packaging analysis apps, so that the analytic scripts do not fail when they run.

One tool validates individual analytic headers at the script level. The other tool validates all the analytic headers in a project at once. The two types of validation focus on different things.

Validate an individual analytic header

Script-level validation of an analytic header focuses on the syntax of individual analytic tags and reports errors with accompanying line numbers.

1. Open the script containing the analytic header.
2. On the Script Editor toolbar, click **Validate Analytic Header** .

   A message appears telling you that the analytic header is valid, or specifying an error and the line number where the error occurs.
3. If the analytic header contains an error, correct the error and click **Validate Analytic Header** again to ensure that there are no further errors.

   > **Tip**
   >
   > If the nature of the error is not apparent based on the error message, review the Help topic for the associated analytic tag. Carefully compare the syntax in the topic with the syntax in the line of the analytic header. Errors can be caused by minor discrepancies in the analytic header syntax.

Validate all the analytic headers in a project

Project-level validation of the analytic headers checks two things:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- at least one analytic header exists in the project
- names of multiple analytic scripts are unique

> **Note**
>
> The name of the analytic script is the name specified in the ANALYTIC tag, not the script name in the **Overview** tab in the **Navigator**.

Project-level validation is performed automatically when you commit scripts to Robots. You can also perform the validation manually if you add the **Check Scripts** button to the Analytics toolbar.

1. If necessary, add the **Check Scripts** button to the Analytics toolbar:
   a. Double-click an empty spot on the toolbar to open the **Customize Toolbar** dialog box.
   b. In the **Available toolbar buttons** list, select the **Check Scripts** button and click **Add**.
   c. In the **Current toolbar buttons** list, select the **Check Scripts** button and click **Move Up** or **Move Down** to change the location of the button.

      The order of the buttons from top to bottom corresponds to their location from left to right on the toolbar.

   d. Click **Close** to save your changes.
2. On the toolbar, click **Check Scripts** .

   A message appears telling you that the analytic headers in the project are valid, or specifying one or more errors.

3. If the analytic headers contain an error, correct the error and click **Check Scripts** again to ensure there are no further errors.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytic development best practices

Analytic scripts support most of the commands that you can use in a regular script. However, you must ensure that analytic scripts run without user interaction, and that they do not include commands not supported by the engine that processes the analytic scripts in the deployment environment.

Analytic scripts support all ACLScript functions.

General best practices

Use one Analytics project per robot or analysis app

Create a new Analytics project for each robot or analysis app. The project must contain all the analytic scripts that make up the robot or the analysis app, and any required subscripts.

For an analysis app, the project must also contain any Analytics tables required by any of the analytic scripts.

Test locally

Test all analytic scripts locally before deploying them to the target environment. Ensure that analytic scripts run as expected, and that they do not require user interaction.

For more information, see "Developing analytic scripts" on page 857.

Use consistent data connections for testing

To test an analytic script locally if it uses an ODBC data source, you must configure an ODBC connection on your local computer that is identical to the connection in the environment where the analytic script will run.

For analytic scripts distributed for use in the Analysis App window, end users must configure an identical ODBC connection on their computers.

Avoid absolute file paths

Avoid using absolute file paths in analytic scripts (for example, `C:\results`) unless you are certain that identical file paths exist in the environment where the analytic script will run.

Using relative file paths such as `\results` allows you to develop and test analytic scripts locally and then deploy them in another environment without requiring that the other environment has an identical directory structure.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Use SET for preference settings

Use the SET command to specify any preference settings required by the analytic script. If you do not specify preferences in the analytic script, the default Analytics preferences are used. Position the SET command after the analytic header but before any of the analytic script logic.

Do not use computed fields in results or data output tables

Do not use computed fields in any output tables that you intend to keep beyond the session in which the analytic script runs.

Results and data tables that are kept for use in interpretations or as input for subsequent scripts may display unexpected values if they contain computed fields. Computed values are dependent on settings defined in the preference file (`.prf`), or by the SET command, and therefore different environments may produce different values.

If you need to retain the values in a computed field, use the EXTRACT command with the FIELDS or ALL option to convert the field to a physical field in a result or data table. For more information, see "EXTRACT command" on page 203.

Encrypt data connection passwords

To avoid having a data source password in plain text in an analytic script, use the PASSWORD analytic tag. This tag prompts the user for a password before running the analytic script, and encrypts the entered value.

Use a password when importing from or exporting to HighBond

The PASSWORD parameter is required in any command that imports from or exports to HighBond:

- IMPORT GRCRESULTS
- IMPORT GRCPROJECT
- EXPORT... ACLGRC

Without the PASSWORD parameter, the commands fail in Robots, Analytics Exchange, or the Analysis App window.

When you use the PASSWORD parameter in a command, you must also specify an associated password tag in the analytic header. For more information, see "PASSWORD" on page 908.

> **Note**
>
> The PASSWORD parameter is not required when running the import and export commands in Analytics because the current user's HighBond access token is automatically used.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Avoiding user interaction

Analytic scripts must be able to run without user interaction. If a command in an analytic script tries to create a dialog box, the engine in the deployment environment stops processing the analytic script, and an error is entered in the log.

Replace user interaction commands with analytic tags

Do not use Analytics commands that require user interaction. Replace them with equivalent analytic tags in the analytic header. Analytic tags allow users to provide input values before the analytic script runs.

Do not use	Replace with
DIALOG	//TABLE, //FIELD, //PARAM
ACCEPT	//TABLE, //FIELD, //PARAM
PASSWORD	//PASSWORD
PAUSE	no equivalent

Guidelines

- **Interactive commands** – To prevent analytic script processing failures, remove all interactive commands.
- **SET SAFETY** – To ensure files can be overwritten as necessary without displaying a confirmation dialog box, add the SET SAFETY OFF command at the beginning of an analytic script.

  Add the SET SAFETY ON command at the end of the analytic script to restore the default behavior.
- **OK parameter** – To prevent confirmation dialogs from crashing the analytic script, add the OK parameter after any commands that normally display a confirmation dialog box:
  - RENAME
  - DELETE

Checking script syntax

Analytics provides a tool for detecting script syntax that causes analytic scripts to fail, or that requires alignment between your local environment and the environment where the analytic scripts are deployed. The tool provides a warning only, and you are still free to commit or import analytic scripts that have warnings.

What the tool checks

The tool checks all scripts in a project for the following items:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- any command that requires user interaction
- any absolute file path
- any call of an external script

When the check is performed

Script syntax checking is performed automatically when you commit scripts to Robots.

Automatic syntax checking is enabled by default. If you want to turn it off, select **Disable Script Syntax Check Before Commit Scripts** in the **Options** dialog box (**Tools > Options > Interface**).

Perform checking manually

You can perform script syntax checking manually. You may need to first add the **Check Scripts** button to the Analytics toolbar.

1. If necessary, add the **Check Scripts** button to the Analytics toolbar:
   a. Double-click an empty spot on the toolbar to open the **Customize Toolbar** dialog box.
   b. In the **Available toolbar buttons** list, select the **Check Scripts** button and click **Add**.
   c. In the **Current toolbar buttons** list, select the **Check Scripts** button and click **Move Up** or **Move Down** to change the location of the button.

      The order of the buttons from top to bottom corresponds to their location from left to right on the toolbar.

   d. Click **Close** to save your changes.
2. On the toolbar, click **Check Scripts** .

   A message appears telling you that the script syntax in the project is valid, or specifying one or more warnings.
3. Do one of the following:
   - Correct any script syntax that generates a warning, and click **Check Scripts** again to ensure that the warnings no longer appear.
   - Ensure that the deployment environment contains a directory structure, or external scripts, that align with the paths or external scripts specified in the analytic script.

Best practices for analytic scripts run on AX Server

Develop in Analytics

Develop analytic scripts and their supporting scripts primarily in Analytics before importing them to AX Server.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

As a convenience feature, the AX Client script editor does allow you to add new analytic scripts or sub-scripts, or edit existing analytic scripts or subscripts. This feature is useful for fine-tuning the behavior of an analytic script without having to export it to Analytics and then reimport it to AX Server. However, analytic script development work beyond minor adjustments is easier to accomplish in Analytics.

Store related files with the Analytics project

Related files such as database profile files should be stored in the same folder as the Analytics project, but must be imported to AX Server separately.

Avoid commands not supported on AX Server

- direct database server tables linked to Analytics Server Edition for z/OS
- the NOTIFY command supports only SMTP messaging. The MAPI and VIM mail protocols are not supported
- to use the PRINT or TO PRINT command, a default printer must be configured on the server
- the SAVE GRAPH and PRINT GRAPH commands are not supported
- do not use the SET LEARN command in analytic scripts

Minimize AX Server table transactions

Optimize the performance of analytic scripts by minimizing the number of times tables on AX Server are accessed:

1. Use the FILTER command to select the records you need.
2. Use the EXTRACT command to extract only the required fields.

The reduced data set will be processed locally on the server where the analytic script is being run by the AX Engine.

Optimizing analytic scripts in this way is important when the data files are not located on the same server as AX Server or the AX Engine Node processing the analytic script, and the **Copy analytic data** option is not selected in the AX Server Configuration web application.

Inefficient analytic script example

```
OPEN LargeTable
SET FILTER TO trans_date >= `20091201` AND trans_date < `20100101`
COUNT
TOTAL amount
CLASSIFY ON account ACCUMULATE amount TO TransClassAccount
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Efficient analytic script example

```
OPEN LargeTable
SET FILTER TO trans_date >= `20091201` AND trans_date < `20100101`
EXTRACT FIELDS trans_date desc account type amount TO AnalysisTable
OPEN AnalysisTable
COUNT
TOTAL amount
CLASSIFY ON account ACCUMULATE amount TO TransClassAccount
```

Access SAP data in background mode

Use Background mode to access data from SAP ERP systems using Direct Link.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Packaging analysis apps

To run analytic scripts in the Analysis App window or on AX Server, create a packaged analysis app (`.aclapp` file). You can package an Analytics project into a new `.aclapp` file, or merge an Analytics project with an existing analysis app (`.aclx` file) to include existing interpretations.

> **Note**
>
> If at least one script in an Analytics project contains an analytic header, the project can be packaged as an analysis app.
>
> Before packaging an analysis app make sure you validate the analytic header of each analytic script in the analysis app.

Why use a packaged analysis app?

Distributing to Analysis App window users

Use packaged analysis apps to distribute a project to users who can extract the `.aclx` file and open the contents in the Analysis App window.

In the Analysis App window, you can run the analytic scripts and create interpretations based on tables and output results.

Importing into AX Server

Use packaged analysis apps to prepare an Analytics project for import into AX Server. You can choose which tables and data files to import along with the analytic scripts in the project.

You can also use a packaged analysis app (`.aclapp`) to import existing interpretations. To include the interpretations from an existing analysis app (`.aclx`) you must repackage your Analytics project and merge this packaged analysis app (`.aclapp`) with the existing `.aclx` file in your project folder.

> **Note**
>
> When you use an existing analysis app (`.aclx` file), the contents of the Analytics project take precedence, so if there are scripts or tables in the `.aclx` that no longer exist in the `.acl` file, they are not included in the resulting packaged analysis app (`.aclapp` file).

File size limitation

To use a packaged analysis app successfully, you must ensure that the sum of all file sizes included in the package does not exceed 800 MB before packaging the analysis app. If your prepackaged files exceed this combined size, data files may be corrupted when unpacking the analysis app.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Package a new analysis app

1. In Analytics, right-click the project entry in the **Overview** tab of the **Navigator** and select **Package Analysis App**.

   The Analytics project is the top-level folder in the tree view.

2. In the **Select Tables** dialog box, do the following:
   a. If you want to include one or more of the project tables and associated data files in the analysis app, select the table(s) and the data file(s) to include.

      > **Note**
      >
      > Generally you should include only static tables and data files that are required by one or more of the analytic scripts in the analysis app, such as a master vendor table, or a list of merchant category codes.

   b. Click **To** and navigate to the location where you want to save the packaged analysis app.
   c. In the **Save As** dialog box, enter a **File name** with the `.aclapp` file extension and click **Save**.
   d. Click **OK**.

Result - the packaged analysis app is saved to the location you specified. Other users can retrieve the packaged analysis app from this location, or you can distribute it by email, or by other appropriate method. You can also import this file into AX Server.

Package an analysis app with interpretations

1. Ensure that the analysis app (`.aclx` file) that contains the interpretations you want to import exists in the Analytics project folder on your computer and uses the same file name as the Analytics project (`.acl` file).

2. In Analytics, right-click the project entry in the **Overview** tab of the **Navigator** and select **Package Analysis App**.

   The Analytics project is the top-level folder in the tree view.

3. In the **Select Tables** dialog box, do the following:
   a. If you want to include one or more of the project tables and associated data files in the analysis app, select the table(s) and the data file(s) to include.

      > **Note**
      >
      > Generally you should include only static tables and data files that are required by one or more of the analytic scripts in the analysis app, such as a master vendor table, or a list of merchant category codes.

   b. Optional. To include the interpretations from the existing analysis app, select **Include Interpretations**.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Interpretations that are associated with tables or scripts that do not exist in the new package are not included.

c. Click **To** and navigate to the location where you want to save the packaged analysis app.
d. In the **Save As** dialog box, enter a **File name** with the `.aclapp` file extension and click **Save**.
e. Click **OK**.

Result - the packaged analysis app is saved to the location you specified. Other users can retrieve the packaged analysis app from this location, or you can distribute it by email, or by other appropriate method. You can also import this file into AX Server.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Sample analytic scripts (analysis app)

The sample analytic scripts contain one import script (two versions), one preparation script, and one analysis script. The analytic scripts can be run in any of the following environments or client applications:

- Robots
- AX Server :
  - AX Client
  - AX Web Client
- the Analysis App window

Sequence of the analytic scripts

The three analytic scripts are designed to work in conjunction, and need to be run in the following sequence:

Sequence	ANALYTIC TYPE	Analytic script name
1	IMPORT	Sample Import analytic Robots_AX or Sample Import analytic Web_AA_Window
2	PREPARE	Sample Preparation analytic
3	ANALYSIS	Sample Analysis analytic

Sample import analytic script

Imports data from the sample Excel file `Trans_May.xls` and saves it to the new Analytics table `Trans_May_raw` (the raw data table).

Two versions of this analytic script are provided.

Analytic script name	Use in	Import file requirement
Sample Import analytic Robots_AX	◦ Robots ◦ AX Client	◦ **Robots** - `Trans_May.xls` must be located in the **Input/Output** tab in the same robot as the analytic script ◦ **AX Client** - `Trans_May.xls` must be located in the **Related Files** subfolder in the AX folder where the analytic script is located
Sample Import analytic Web_AA_Window	◦ AX Web Client ◦ Analysis App window	

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Sample import analytic script for use in Robots or AX Client

```
COMMENT
//ANALYTIC TYPE IMPORT Sample Import analytic Robots_AX
  This analytic script imports data from the sample Excel file Trans_May.xls and saves it to the new Analytics table "Trans_May_raw" (the raw data table).
//FILE Trans_May.xls
//DATA Trans_May_raw
//RESULT LOG
END

SET SAFETY OFF
IMPORT EXCEL TO Trans_May_raw Trans_May_raw.fil FROM "Trans_May.xls" TABLE "Trans2_May$" KEEPTITLE FIELD "CARDNUM" C WID 22 AS "" FIELD "CODES" C WID 4 AS "" FIELD "DATE" D WID 10 PIC "YYYY-MM-DD" AS "" FIELD "CUSTNO" C WID 6 AS "" FIELD "DESCRIPTION" C WID 95 AS "" FIELD "AMOUNT" N WID 9 DEC 2 AS ""
SET SAFETY ON
```

Sample import analytic script for use in AX Web Client or the Analysis App window

```
COMMENT
//ANALYTIC TYPE IMPORT Sample Import analytic Web_AA_Window
  This analytic script imports data from the sample Excel file Trans_May.xls and saves it to the new Analytics table "Trans_May_raw" (the raw data table).
//PARAM v_input_file F Input File
  Select an input file
//DATA Trans_May_raw
//RESULT LOG
END

SET SAFETY OFF
IMPORT EXCEL TO Trans_May_raw Trans_May_raw.fil FROM "%v_input_file%" TABLE "Trans2_May$" KEEPTITLE FIELD "CARDNUM" C WID 22 AS "" FIELD "CODES" C WID 4 AS "" FIELD "DATE" D WID 10 PIC "YYYY-MM-DD" AS "" FIELD "CUSTNO" C WID 6 AS "" FIELD "DESCRIPTION" C WID 95 AS "" FIELD "AMOUNT" N WID 9 DEC 2 AS ""
SET SAFETY ON
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Sample preparation analytic script

Prepares the raw data table for analysis and saves it to the new Analytics table `Trans_May_prepared` (the analysis table). The analytic script defines a shorter version of the "Description" field because classifying only supports field lengths up to 64 characters.

```
COMMENT
//ANALYTIC TYPE PREPARE Sample Preparation analytic
  This analytic script prepares the raw data table for analysis and saves it to the new Analytics table
"Trans_May_prepared" (the analysis table). The analytic script defines a shorter version of the
"Description" field because classifying only supports field lengths up to 64 characters.
//TABLE v_RawTable Table to prepare
  Select the raw data table you want to prepare
//RESULT TABLE Trans_*_prepared
//DATA Trans_*_prepared
//RESULT LOG
END

SET SAFETY OFF
OPEN %v_RawTable%
DEFINE FIELD DESC_SHORT    ASCII    43  64
EXTRACT RECORD TO "Trans_May_prepared"
SET SAFETY ON
```

Sample analysis analytic script

Classifies the analysis table and outputs the results to the new Analytics table `Classified_Trans_May_prepared` (the results table). Users can specify which field to use for classifying the table, and can specify merchant category codes, customer numbers, and date and transaction amount ranges to restrict which records are processed.

```
COMMENT
//ANALYTIC TYPE ANALYSIS Sample Analysis analytic
  This analytic script classifies the analysis table and outputs the results to the new Analytics table "Classified_Trans_May_prepared" (the results table). You can specify merchant category codes, customer numbers, and date and transaction amount ranges, to restrict which records are processed.
//TABLE v_AnalysisTable Table to classify
  Select the analysis table you want to classify
  //FIELD v_FieldA C Field to classify on
  Select the field you want to classify on
//PARAM v_codes C MULTI SEPARATOR , QUALIFIER ' VALUES |4112 Passenger Railways|4121 Taxis/Limousines|4131 Bus travel|4215 Courier Services - Air or Ground|4411 Cruise Lines|4457
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
Boat Leases and Boat Rentals|4722 Travel Agencies and Tour Operations|4814 Local/long-distance
calls|5812 Restaurants|5813 Drinking Places (Alcoholic Beverages)|5814 Fast food restaurants|5921
Package Stores, Beer, Wine, Liquor|5993 Cigar Stores & Stands|5994 Newsstand|7216 Dry cleaners|
MC code(s) to include
  Specify one or more merchant category codes to include
//PARAM v_cust_no C OPTIONAL MULTI SEPARATOR , QUALIFIER ' Customer Number(s) to
exclude (optional)
  Specify one or more customer numbers to exclude. Press "Enter" after each number, so that each num-
ber is on a separate line. Do not enclose numbers in quotation marks.
//PARAM v_start_date D VALUES
|05/01/2003|05/02/2003|05/03/2003|05/04/2003|05/05/2003|05/06/2003|05/07/2003|05/08/2003|05/0-
9/2003|05/10/2003|05/11/2003|05/12/2003|05/13/2003|05/14/2003|05/15/2003|05/16/2003|05/17/200-
3|05/18/2003|05/19/2003|05/20/2003|05/21/2003|05/22/2003|05/23/2003|05/24/2003|05/25/2003|05/-
26/2003|05/27/2003|05/28/2003|05/29/2003|05/30/2003|05/31/2003|Start Date
  Select a start date
//PARAM v_end_date D End Date
  Enter an end date or pick one from the calendar
//PARAM v_min_amount N Minimum Amount
  Enter a minimum amount
//PARAM v_max_amount N Maximum Amount
  Enter a maximum amount
//RESULT TABLE Classified_*
//RESULT LOG
END

SET SAFETY OFF
OPEN %v_AnalysisTable%
IF NOT ISDEFINED("v_cust_no") v_cust_no = ""
GROUP IF v_cust_no = ""
  CLASSIFY ON %v_FieldA% IF MATCH(CODES, %v_codes%) AND BETWEEN(DATE, v_start_
date, v_end_date) AND BETWEEN(AMOUNT, v_min_amount, v_max_amount) SUBTOTAL
AMOUNT TO "Classified_%v_AnalysisTable%.FIL" OPEN
ELSE
  CLASSIFY ON %v_FieldA% IF MATCH(CODES, %v_codes%) AND NOT MATCH(CUSTNO, %v_
cust_no%) AND BETWEEN(DATE, v_start_date, v_end_date) AND BETWEEN(AMOUNT, v_min_
amount, v_max_amount) SUBTOTAL AMOUNT TO "Classified_%v_AnalysisTable%.FIL" OPEN
END
SET SAFETY ON
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytic headers and tags

An analytic header is a series of tags enclosed in a comment block at the start of an Analytics script. The analytic tags specify input parameters that a user populates in advance of scheduling or running an analytic script, and output parameters.

An analytic header is required for any analytic script that you intend to run in Robots, on AX Server, or in the Analysis App window.

Basic analytic header requirements

Analytic headers must be defined in a comment block that starts on the first line of the script. Tags can be in any order in the analytic header, except for:

- the ANALYTIC tag, which must be first
- FIELD tags, which must immediately follow their associated TABLE tag

Example

This analytic header identifies a table and field to use in the script, as well as a start date parameter:

```
COMMENT
//ANALYTIC Identify missing checks
  This analytic script identifies missing check numbers
//TABLE v_table_payments Payments Table
  Select a table that lists payments and includes a check number column
//FIELD v_check_num CN Check Number
  Select the field that contains the check numbers
//PARAM v_start_date D OPTIONAL Start Date (Optional)
  Enter the start date for the analysis
END
```

Tag format

Each tag in the header uses the following format:

```
//tag_name attributes
  optional_descriptive_text
```

The // tag indicator must be the first non-whitespace character on the script line. The tag name must immediately follow the tag indicator, without any space or characters in between.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

The optional descriptive text must be entered on the next line after the tag. The text can be multiline, but it cannot skip lines. Line breaks are not preserved when the descriptive text is displayed in client applications.

Tag conventions

Component	Convention
Tag names	Tag names are not case-sensitive. Unlike Analytics command and function names, tag names cannot be abbreviated.
Tag attributes	When specifying attribute values for a tag, you may include spaces and optionally enclose the value in quotation marks.
Tag descriptions	Descriptions are optional. If a description is specified it can be multi-line, but line breaks are not preserved in client applications.

Specifying test input values in Analytics

You can use a special assignment operator (`:=`) to specify test input values for any analytic tag that requires definition:

- FILE
- TABLE
- FIELD
- PARAM

Use this syntax to test analytic scripts in Analytics:

```
//TABLE v_AnalysisTable "Table to classify" := "Trans_May"
```

When the script runs in Analytics, the parameter takes the value specified in the assignment. When the analytic script runs in a client application, the test value is ignored and the user-defined input parameters are used.

You must leave a space between the assignment operator and the tag syntax preceding it. Assignment values must use the correct qualifier for the data type as required throughout Analytics. For more information, see "Data types" on page 20.

Full list of available analytic tags

Tag	Description
"ANALYTIC" on page 887	Designates a script as an analytic script that can run in Robots, on AX Server, or in the Analysis App window.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Tag	Description
Input tags	
"FILE" on page 889	Specifies a non-Analytics file, such as an Excel file, or a delimited file, that provides input for an analytic script running in Robots, or on AX Server. ◦ **Robots** - the file must be located in the **Input/Output** tab in the same robot as the analytic script ◦ **AX Server** - the file must be located in the **Related Files** subfolder in the folder where the analytic script is located
"TABLE" on page 892	Defines an Analytics table that the user selects as input for an analytic script. The TABLE tag can be followed by zero or more FIELD tags entered on sequential lines.
"FIELD" on page 894	Defines a field that the user selects as input for an analytic script. The field must be part of the table defined in the preceding TABLE tag. The first FIELD tag must immediately follow a TABLE tag, and can be followed by additional FIELD tags entered on sequential lines.
"PARAM" on page 896	Creates an input parameter for an analytic script, and defines the requirements for the input value. An input parameter is a placeholder that allows the user to specify the actual value when scheduling or running an analytic script.
"PASSWORD" on page 908	Creates a password input parameter for an analytic script. The parameter provides encrypted storage of a password for subsequent use in an ACLScript command. The user is prompted to specify the required password value when they schedule or start the analytic script so that user intervention is not required as the analytic script is running.
Output tags	
"DATA" on page 911	Specifies that an Analytics table output by an analytic script is copied to a data subfolder (a storage location) in the deployment environment. Typically, you store Analytics tables so that they can be used as input tables for subsequent analytic scripts.
"RESULT" on page 915	Specifies the output results of an analytic script that you want to make available to end users in client applications. Output results, even if they exist, are not automatically made available.
"PUBLISH" on page 920	Specifies a file that contains metadata defining which Analytics tables to publish to AX Exception when an analytic script is finished processing.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

ANALYTIC

Designates a script as an analytic script that can run in Robots, on AX Server, or in the Analysis App window.

Syntax

```
//ANALYTIC <TYPE IMPORT|PREPARE|ANALYSIS> name
 <description>
```

Parameters

Name	Description
TYPE optional	Identifies the kind of analytic script as one of the following three types: ◦ IMPORT – retrieves data from a data source. The output of an import analytic script is a raw data table. ◦ PREPARE – transforms raw data in whatever way is necessary to make it suitable for analysis. The output of a preparation analytic script is an analysis table. ◦ ANALYSIS – performs tests on data in analysis tables. The output of an analysis analytic script is one or more results tables. Analytic scripts with a specified TYPE are organized in the corresponding **Import**, **Preparation**, or **Analysis** areas in Robots, AX Web Client, and the Analysis App window. This placement guides the user in the appropriate sequence for running the analytic scripts. The sequence is not enforced, nor is the type of functionality within the analytic script. If you omit TYPE, the analytic script appears in the **Analysis** section.
name	The name of the analytic script. The name identifies the analytic script in client applications. The analytic script name is not the same as the script name that you specify in Analytics when you initially create the script. **Note** Analytic scripts in the same project or analysis app must be uniquely named. If the same *name* is used in two or more analytic scripts, an error occurs when you try to commit the analytic scripts, or import or open the analysis app. To ensure that an analytic script name does not cause a processing problem, a best practice is to use only these characters in a name: A-Z , a-z , 0-9 , underscore (_), or dash (-) The following characters cannot be used in an analytic script name: < > : " / \ | ? * Do not use the value TYPE as the name. In client applications, analytic script names are listed in alphanumeric order. To guide

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	users in the correct sequence for running multiple analytic scripts in a single robot or analysis app, you can add a prefix to order the analytic script names within each area. For example: 01_analyze_POs , 02_analyze_invoices , and so on. The sequence implied by the order of the names is not enforced.
description optional	A description of the analytic script or other information that the user might need to run the analytic script successfully. The description appears with the analytic script in client applications. The description can be multiline, but it cannot skip lines. The description must be entered on the line below the associated ANALYTIC tag.

Examples

Basic analytic header

The following analytic header contains a name and a description of the analytic script:

```
COMMENT
//ANALYTIC Identify missing checks
  This analytic script identifies missing check numbers.
END
```

Analytic header with type

The following analytic header specifies a preparation analytic script with a description of what the script does:

```
COMMENT
//ANALYTIC TYPE PREPARE Standardize address data
  This analytic script cleans and standardizes the address field in preparation for duplicates analysis.
END
```

Remarks

An ACLScript COMMENT command must be entered on the first line in the script, followed by the ANALYTIC tag on the second line. If the ANALYTIC tag is used in any other location it is ignored.

One or more scripts in an Analytics project can include an ANALYTIC tag.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FILE

Specifies a non-Analytics file, such as an Excel file, or a delimited file, that provides input for an analytic script running in Robots, or on AX Server.

- **Robots** – the file must be located in the **Input/Output** tab in the same robot as the analytic script
- **AX Server** – the file must be located in the **Related Files** subfolder in the folder where the analytic script is located

> **Note**
>
> To specify a non-Analytics input file for an analytic script run in the Analysis App window, see "PARAM" on page 896.

Syntax

```
//FILE filename
  <description>
```

Parameters

Name	Description
filename	The name of the file in the **Input/Output** tab, or in the **Related Files** subfolder, to use as input for an analytic script. **Note** The *filename* value must exactly match the name of the file in the **Input/Output** tab, or in the **Related Files** subfolder. *filename* cannot include a path. You can use wildcard characters in *filename* to assist with matching a name. You cannot use a variable for *filename*. ### Wildcard characters Wildcards are supported when specifying the file name. Use a single asterisk (*) to substitute for zero or more consecutive characters. For example: ◦ Inv12* matches all of the following: `Inv12`, `Inv123`, and `Inv1234` ◦ *.* matches all files of all extensions ◦ Inv_*.* matches `Inv_Jan.pdf` and `Inv_Feb.xls` ◦ *.xlsx matches all Excel files with a `.xlsx` extension ### Analytics preferences file (AX Server)

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	You can use the //FILE tag to reference a `.prf` Analytics preferences file. When you do this, the preferences file in the **Related Files** subfolder is used to set the runtime environment settings rather than the global preferences file on AX Server. The preferences file must be from the latest version of Analytics that is compatible with your Analytics Exchange installation.
description optional	Descriptive text about the non-Analytics file or other information. The description can be multiline, but it cannot skip lines. The description appears in the analytic header only and is not visible to end users in client applications.

Examples

Basic examples

Specifies a specific file:

```
//FILE FlaggedAccounts.csv
```

Specifies all CSV files starting with "Flagged":

```
//FILE Flagged*.csv
```

Specifies all files:

```
//FILE *.*
```

Advanced examples

Import data from a related file

You run a monthly analysis of employee data contained in a delimited file that is manually uploaded to the **Related Files** subfolder on AX Server on a monthly basis. One of the analytic scripts in the analysis app imports the data from the delimited file to an Analytics table:

```
COMMENT
//ANALYTIC TYPE IMPORT employee_import
  Imports employee records from delimited file stored in Related Files folder.
//FILE Employees.csv
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
END
IMPORT DELIMITED TO Employees "Employees.fil" FROM "Employees.csv" 0 SEPARATOR ","
QUALIFIER "" CONSECUTIVE STARTLINE 1 KEEPTITLE FIELD "First_Name" C AT 1 DEC 0 WID
11 PIC "" AS "First Name"  FIELD "Last_Name" C AT 12 DEC 0 WID 12 PIC "" AS "Last Name"
```

Remarks

To be used in a script, a non-Analytics file must first be imported into an Analytics table. Non-Analytics files cannot be used directly in a script.

The FILE tag is not supported for use in analytic scripts run in the Analysis App window. To specify an input file for analytic scripts run in the Analysis App window, use the PARAM tag. For more information, see "PARAM" on page 896.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

TABLE

Defines an Analytics table that the user selects as input for an analytic script.

The TABLE tag can be followed by zero or more FIELD tags entered on sequential lines.

> **Note**
>
> The TABLE tag requires that a table pre-exists in the storage location in order to be available to be selected. For more information, see "DATA" on page 911.

Syntax

```
//TABLE variable_name label
  <description>
```

Parameters

Name	Description
variable_name	The name of the variable that stores the input table name selected by the user. Use *variable_name* in the analytic script to reference the table.
label	In client applications, the interface label that users see when prompted to select the table. For example, Payments Table
description optional	In client applications, descriptive text associated with the input field that users see. The description can be multiline, but it cannot skip lines. The description can help the user select the correct table. For example, Select a table that lists payments and includes a check number column. The description must be entered on the line below the associated TABLE tag.

Examples

Basic examples

TABLE tag with description to help user select the correct input table:

```
//TABLE v_table_payments Payments Table
  Select a table that lists payments and includes a check number column.
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Advanced examples

Using a table defined in a TABLE tag in the script

The following script runs an AGE command on a table that is selected by the user from the data tables in the project:

```
COMMENT
//ANALYTIC example_script
//TABLE v_table_payments Payments Table
  Select a table that lists payments and includes a check number column.
END

OPEN %v_table_payments%
AGE ON payment_date CUTOFF 20141231 INTERVAL 0,30,60,90,120,10000 SUBTOTAL Pay-
ment_Amount TO r_output
CLOSE %v_table_payments%
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

FIELD

Defines a field that the user selects as input for an analytic script.

The field must be part of the table defined in the preceding TABLE tag. The first FIELD tag must immediately follow a TABLE tag, and can be followed by additional FIELD tags entered on sequential lines.

> **Note**
>
> The TABLE tag requires that a table pre-exists in the storage location in order to be available to be selected. For more information, see "DATA" on page 911.

Syntax

```
//FIELD variable_name type label
  <description>
```

Parameters

Name	Description
variable_name	The name of the variable that stores the input field name selected by the user. Use *variable_name* in the analytic script to reference the field.
type	The types of fields that can be selected. Any type, or combination of types, from the following list can be selected: ◦ C - character data ◦ N - numeric data ◦ D - date, datetime, or time subtype of datetime data ◦ L - logical data Any computed fields in a table can be selected regardless of the *type* specified.
label	In client applications, the interface label that users see when prompted to select the field. For example, Payment Date Field
description optional	In client applications, descriptive text associated with the input field that users see. The description can be multiline, but it cannot skip lines. The description can help the user select the correct field. For example, Select the column that contains the check payment date. The description must be entered on the line below the associated FIELD tag.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

Specifies a character field:

```
//FIELD v_last_name C Last Name Field
```

Specifies a character or numeric field:

```
//FIELD v_inv_num CN Invoice Number
```

Advanced Examples

TABLE with two accompanying FIELD tags

The following analytic header allows the user to specify two input fields from the *v_table_payments* table when the script runs:

```
COMMENT
//ANALYTIC test analytic
//TABLE v_table_payments Payments Table
  Select a table that lists payments and includes a check number column.
//FIELD v_check_num CN Check Number Field
//FIELD v_payment_date D Payment Date Field
  Select the column that contains the check payment date.
END

OPEN %v_table_payments%
EXTRACT FIELDS %v_check_num%, %v_payment_date% TO t_analyze
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PARAM

Creates an input parameter for an analytic script, and defines the requirements for the input value.

An input parameter is a placeholder that allows the user to specify the actual value when scheduling or running an analytic script.

Syntax

```
//PARAM variable_name type <OPTIONAL> <MULTI> <SEPARATOR value> <QUALIFIER value>
<VALUES value_list> label
  <description>
```

Parameters

Name	Description
variable_name	The name of the variable that stores the input value(s) selected or specified by the user. Use *variable_name* in the analytic script to reference the input value. For example: ◦ v_start_date ◦ v_regions ◦ v_input_file Also serves as the unique identifier for the parameter. **Note** When an analytic script is run, the variable is created only if the user provides an input value. If a parameter is optional, and the user skips it, the variable is not created. If subsequent logic in the analytic script requires the variable to exist, you can test for its existence, and if it does not exist, create it and initialize it. For more information, see "Designing optional input parameters" on page 903.
type	The data type of the parameter, which controls what sort of input values can be entered. The following types can be specified using uppercase letters: ◦ C - character data ◦ N - numeric data ◦ D - date subtype of datetime data ◦ DT - datetime subtype of datetime data ◦ T - time subtype of datetime data ◦ L - logical data

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>
<blockquote>Note
Qualifying character input values is required for an analytic script to run successfully.</blockquote>
How PARAM... F works
You can also specify that a file upload utility, or a Windows file browser, opens:
◦ F - opens a file upload utility, or a Windows file browser, and allows a user to select a non-Analytics input file for the analytic script when running in AX Web Client or the Analysis App window
Upon selection, the file name is automatically entered as a Character input value. Specify F only. Do not specify F C.
For example:
<pre>//PARAM v_input_file F...</pre>
For more information, see "Specifying or selecting a non-Analytics input file for an analytic script" on page 906.
<blockquote>Note
A type of F is not supported for use in analytic scripts run in Robots or AX Client. To specify an input file for these environments, use the FILE tag. For more information, see "FILE" on page 889.</blockquote>
</td></tr>
<tr><td>OPTIONAL
optional</td><td>Specifies that the parameter is optional and the user does not need to enter a value.
For more information, see "Designing optional input parameters" on page 903.</td></tr>
<tr><td>MULTI
optional</td><td>
Specifies that the parameter accepts multiple input values.
<blockquote>Note
MULTI cannot be used if type is L (Logical) or F (File).</blockquote>
MULTI and VALUES
MULTI can be used with or without the VALUES option:
<table>
<tr><td>MULTI ✔
VALUES ✔</td><td>The user can select one or more values from a list of values.</td></tr>
<tr><td>MULTI ✔
VALUES ✘</td><td>The user can manually enter one or more values.</td></tr>
</table>
For more information, see "Summary of the MULTI and VALUES options" on page 903.
Multiple character input values
If you specify MULTI , and type is C (Character), you can also specify the SEPARATOR and QUALIFIER options to automatically insert separators (delimiters) and text qualifiers in a string of input values.
</td></tr>
</table>

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	Note Delimiting and qualifying multiple character input values is required for an analytic script to run successfully. The separators and qualifiers can be inserted automatically, or manually by the user.
SEPARATOR *value* optional	SEPARATOR can be used only when MULTI is specified, and *type* is C (Character). Specifies that a separator character is automatically inserted between multiple character input values, creating a delimited list that is passed to the analytic script for processing. *value* specifies the separator character to use. A commonly used separator, or delimiter, is the comma , . If SEPARATOR is omitted, a single space is used as the separator by default. The space character cannot be specified as *value*. For more information, see "Delimiting and qualifying character input values" on page 904.
QUALIFIER *value* optional	QUALIFIER can be used only when MULTI is specified, and *type* is C (Character). Specifies that a text qualifier character is automatically inserted at the start and end of each character input value in a delimited list that is passed to the analytic script for processing. Any text enclosed within the qualifier characters is treated as plain text. *value* specifies the qualifier character to use. A commonly used qualifier is the single quotation mark ' . If QUALIFIER is omitted, there is no default qualifier used. You cannot specify a space character as *value*. For more information, see "Delimiting and qualifying character input values" on page 904. **Note** Analytic input parameters currently do not support the use of the double quotation mark (") as a text qualifier. You can use the single quotation mark (') instead. Specifying a double quotation mark qualifier will cause the PARAM tag to malfunction.
VALUES *value_list* optional	A list of values that the user can select from when running the analytic script. Use the following syntax to specify the values: VALUES |Value 1|Value 2|Value 3|Value *n*| **VALUES and MULTI** VALUES can be used with or without the MULTI option: VALUES ✔ MULTI ✔ — The user can select one or more values from the list of values.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

<table>
<tr><th>Name</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><td>VALUES ✔
MULTI ✘</td><td>The user can select a single value from the list of values.</td></tr>
</table>
For more information, see "Summary of the MULTI and VALUES options" on page 903.
Format of values in value_list
<table>
<tr><td>Character values</td><td>◦ can contain spaces and punctuation</td></tr>
<tr><td>Numeric values</td><td>◦ can be positive or negative
◦ must be specified using decimal notation, and without a thousands separator
For example, 1500.00 or -1500.00</td></tr>
<tr><td>Datetime values</td><td>◦ Date - must be specified using the format MM/DD/YYYY
For example, 12/31/2014
◦ Datetime - must be specified using the format MM/DD/YYYY hh:mm:ss
For example, 12/31/2014 23:59:59
◦ Time - must be specified using the format hh:mm:ss
For example, 23:59:59</td></tr>
<tr><td>Logical values</td><td>VALUES cannot be used if type is L (Logical)</td></tr>
</table>
</td></tr>
<tr><td>label</td><td>The user interface label for the parameter.
In client applications, label is displayed with the input field.</td></tr>
<tr><td>description
optional</td><td>Descriptive text that provides additional information about the parameter.
In client applications, description is displayed with the input field.
description can provide instructions that assist the user. For example, "Enter the cutoff date for the payroll period".
description must be entered on the next line after the associated PARAM tag. The text can be multiline, but it cannot skip lines. Line breaks are not preserved when displayed in client applications.</td></tr>
</table>

Examples

Basic examples

Allows the user to optionally specify a date range:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
//PARAM v_start_date D OPTIONAL Start Date (Optional)
  Enter the start date for the analysis
//PARAM v_end_date D OPTIONAL End Date (Optional)
  Enter the end date for the analysis
```

Requires the user to select a maximum number of transactions to process:

```
//PARAM v_maxTrans N VALUES |250|500|750|1000| Maximum transactions to process
```

Requires the user to specify one or more merchant category codes:

```
//PARAM v_codes C MULTI SEPARATOR , QUALIFIER ' MC Codes to include
  Specify one or more merchant category codes. Press "Enter" after each code, so that each code is on a separate line. Do not enclose codes in quotation marks.
```

Requires the user to select one or more merchant category codes:

```
//PARAM v_codes C MULTI SEPARATOR , QUALIFIER ' VALUES |4121 Taxis/Limousines|5812 Restaurants|5813 Drinking Places - Alcoholic Beverages|5814 Fast food restaurants| MC Codes to include
  Select one or more merchant category codes.
```

Advanced examples

Require a user to specify an amount range

You need to classify the records in a table that fall between a minimum and maximum amount range. This range changes occasionally, so you provide input parameters that allow the user who runs the analytic script to define the range when scheduling or running the script:

```
COMMENT
//ANALYTIC test_analytic
//PARAM v_min_amount N Minimum Amount
  Enter a minimum amount
//PARAM v_max_amount N Maximum Amount
  Enter a maximum amount
END

CLASSIFY ON %v_FieldA% IF BETWEEN(AMOUNT, v_min_amount, v_max_amount) SUBTOTAL AMOUNT TO "Classified_%v_AnalysisTable%.FIL"
```

Allow the user to optionally exclude one or more customer numbers

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

You need to classify the records in a table but you want to give the user the option to exclude some customers from the analysis.

To do this, you provide an optional character parameter. Your script tests whether or not the value is provided, and if so, those customer numbers are excluded from the classify command:

```
COMMENT
//ANALYTIC test_analytic
//PARAM v_cust_no C OPTIONAL MULTI SEPARATOR , QUALIFIER ' Customer Number(s) to
exclude (optional)
  Specify one or more customer numbers. Press "Enter" after each number, so that each number is on a
separate line. Do not enclose numbers in quotation marks.
END

IF FTYPE("v_cust_no") = "U" v_cust_no = ""
GROUP IF v_cust_no = ""
  CLASSIFY ON %v_FieldA% SUBTOTAL AMOUNT TO "Classified_%v_AnalysisTable%.FIL"
ELSE
  CLASSIFY ON %v_FieldA% IF NOT MATCH(CUSTNO, %v_cust_no%) SUBTOTAL AMOUNT TO
"Classified_%v_AnalysisTable%.FIL"
END
```

Allow the user to select an input file (AX Web Client or the Analysis App window only)

You are distributing an analysis app to colleagues who will run it in the Analysis App window. When they run the analytic script in the app, you want to provide them with a Windows file browser to select a Microsoft Excel file to import data from:

```
COMMENT
//ANALYTIC test_analytic
//PARAM v_input_file F Input File
  Select an input file
END

IMPORT EXCEL TO Trans_May_raw Trans_May_raw.fil FROM "%v_input_file%" TABLE "Trans2_
May$" CHARMAX 100 KEEPTITLE
```

Require the user to specify an input file path and file name (the Analysis App window only)

You are distributing an analysis app to colleagues who will run it in the Analysis App window. When they run the analytic script in the app, you want them to specify a filepath and filename to use as an import file:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
COMMENT
//ANALYTIC test_analytic
//PARAM v_input_file C Input File Path and Name
  Enter an absolute file path and a file name, for example: C:\Users\username\Documents\ACL
Data\Sample Data Files\ Trans_May.xls
END

IMPORT EXCEL TO Trans_May_raw Trans_May_raw.fil FROM "%v_input_file%" TABLE "Trans2_
May$" CHARMAX 100 KEEPTITLE
```

Using default values for optional parameters

You are creating an analytic script that extracts transaction records to a results table. You want to give the user who runs the script the option of providing a date range as well as a list of entities for filtering the records to extract.

To do this, you create three optional parameters:

- v_start_date
- v_end_date
- v_entity_list

In the opening lines of the script, you test if these values are set. If they are not set, you set default values of the minimum and maximum dates as well as a default flag to test for with v_entity_list.

In your EXTRACT command, you use the values to filter the records:

```
COMMENT
//ANALYTIC test
  This analytic script tests the PARAM
//RESULT TABLE t_results
//PARAM v_start_date D OPTIONAL Enter Start Date
//PARAM v_end_date D OPTIONAL Enter End Date
//PARAM v_entity_list C MULTI OPTIONAL |entity1|entity2|
END

IF NOT ISDEFINED("v_start_date") v_start_date = `19000101`
IF NOT ISDEFINED("v_end_date") v_end_date = `99991231`
IF NOT ISDEFINED("v_entity_list") v_entity_list = "'all'"

EXTRACT FIELDS ALL TO t_results IF BETWEEN(transaction_date v_start_date v_end_date)
AND (MATCH(entity_field %v_entity_list%) OR v_entity_list = "'all'")
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Designing optional input parameters

If you use OPTIONAL with the PARAM tag, the variable associated with the input parameter may or may not be created when the analytic script runs:

- **variable automatically created** – if the user specifies an input value
- **variable not created** – if the user skips the optional parameter and does not specify an input value

Test for the existence of the parameter variable

If subsequent logic in the analytic script depends on being able to evaluate the contents of the parameter variable, including evaluating an empty or null state, you need to test for the existence of the parameter variable. If the parameter variable does not exist, you need to create it and initialize it to null.

Use the IF command with the FTYPE( ) function, or the ISDEFINED( ) function, to perform the test and create the variable if it does not exist:

```
IF FTYPE("var_name") = "U" var_name = ""
```

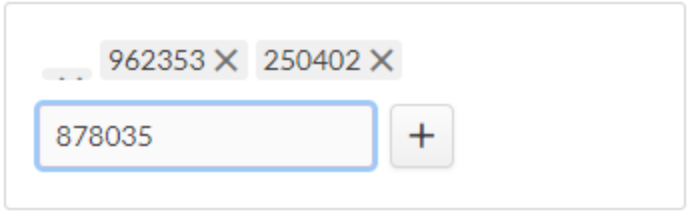
```
IF NOT ISDEFINED("var_name") var_name = ""
```

When to perform the test

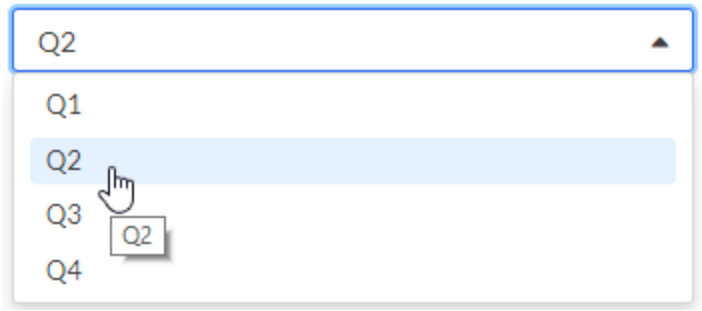
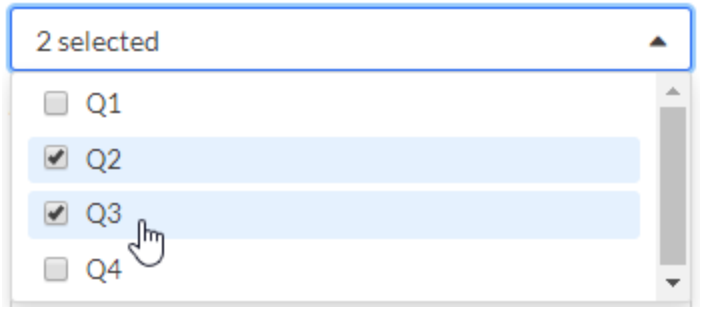
Perform the test after the analytic header and before any Analyticsscript logic that depends on the existence of the parameter variable.

Summary of the MULTI and VALUES options

The table below summarizes the effect of the MULTI and VALUES options on the user input control in the user interface.

User input control (Robots)	Parameter design	MULTI	VALUES
	A single input value manually entered in a field	✗	✗
	One or more input values manually entered in a field	✓	✗

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

User input control (Robots)	Parameter design	MULTI	VALUES
	A single input value selected from a drop-down list of values	✗	✓
	One or more input values selected from a checklist of values	✓	✓

Delimiting and qualifying character input values

For an analytic script to run successfully, character input values must be delimited by a separator if there is more than one value, and values must be qualified.

Avoid nested text qualifiers

When you create character input parameters, and when you instruct users of the analytic script how to enter character input values, you need to be careful to avoid creating redundant or nested text qualifiers (qualifiers within qualifiers). Redundant text qualifiers will cause the input parameter to malfunction.

Methods for inserting text qualifiers

There are four different methods available for inserting text qualifiers around character input values. Depending on the method, a separator is also inserted between the input values.

As you develop an analytic script, you may need to experiment with different methods to find what works best for the character values that users will input.

> **Note**
>
> One or more of the methods may not be applicable, depending on how you are using the MULTI and VALUES options.
>
> Each input parameter must use **only one** of these methods.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

1	Use SEPARATOR and QUALIFIER	Include the SEPARATOR and QUALIFIER options in the PARAM tag. For example: `//PARAM v_regions C MULTI SEPARATOR , QUALIFIER '` Not applicable if you use VALUES without MULTI. **Tip** Use this method whenever possible. It is the least labor-intensive and the least error-prone.
2	Manually specify separators and qualifiers	Require the user of the analytic script to manually specify separators and qualifiers in addition to the actual input values. For example: `'North America','Europe','Asia'` Not applicable if you use VALUES with or without MULTI.
3	Include qualifiers in the value_list	Include qualifiers with each value in the *value_list* specified with the VALUES option. For example: `VALUES |'Asia'|'Europe'|'Middle East'|'North America'|` Not applicable if you use MULTI without VALUES.
4	Enclose the parameter variable in qualifiers	In the syntax of the Analytics script, enclose the parameter variable in text qualifiers. For example: `IF MATCH(REGIONS, "%v_regions%")` Use this method only if you are using VALUES without MULTI.
	Note Analytic input parameters currently do not support the use of the double quotation mark (") as a text qualifier. You can use the single quotation mark (') instead with the QUALIFIER option, in the *value_list*, or when manually specifying qualifiers around input values. Double quotation marks can be used as text qualifiers in the body of an Analytics script.	

When to use the different methods

The table below summarizes when to use the different methods for inserting text qualifiers.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

	MULTI ✔ VALUES ✘	MULTI ✘ VALUES ✔	MULTI ✔ VALUES ✔
Method 1 Use the SEPARATOR and QUALIFIER options	If used, do not use Method 2	Not applicable	If used, do not use Method 3
Method 2 Manually specify separators and qualifiers	If used, do not use Method 1	Not applicable	Not applicable
Method 3 Include qualifiers in the *value_list*	Not applicable	If used, do not use Method 4	If used, do not use Method 1
Method 4 Enclose the parameter variable in qualifiers	Do not use	If used, do not use Method 3	Do not use

Specifying or selecting a non-Analytics input file for an analytic script

The table below summarizes the different methods for specifying or selecting a non-Analytics input file for an analytic script. The method you choose is partly dependent on which client application will be used to run the analytic script.

Method	Details	Robots	AX Client	AX Web Client	The Analysis App window
PARAM tag with type of 'F'	◦ **AX Web Client** - user selects the input file using a file upload utility The file name is automatically specified as the input value. The file is automatically uploaded to the appropriate **Related Files** subfolder on AX Server. ◦ **Analysis App window** - user selects the input file using the Windows file browser The file path and the file name are automatically specified as the input value. This method is the best option because it combines flexibility, ease of use, and precision.			✔	✔
PARAM tag with type of 'C'	The user manually specifies an input file path and file name as an input value.				✔

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Method	Details	Robots	AX Client	AX Web Client	The Analysis App window
	This method provides flexibility because the file path and the file name are not specified in advance. However, it is laborious and error prone because it requires the user to manually enter these values.				
FILE tag (For more information, see "FILE" on page 889)	◦ **Robots** - the input file must be located in the **Input/Output** tab in the robot ◦ **AX Client, AX Web Client** -the input file must be located in the appropriate **Related Files** subfolder on AX Server	✔	✔	✔	
Input file path and file name hard-coded in the analytic script	This method avoids use of the PARAM tag, however it is the least flexible. On every computer where the analytic script is run, the user must ensure that the input file has a file path and a file name identical to those specified in the analytic script.				✔

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PASSWORD

Creates a password input parameter for an analytic script. The parameter provides encrypted storage of a password for subsequent use in an ACLScript command.

The user is prompted to specify the required password value when they schedule or start the analytic script so that user intervention is not required as the analytic script is running.

Syntax

```
//PASSWORD identifier label
  <description>
```

Parameters

Name	Description
identifier	The numerical identifier associated with the password. The value must be from 1 to 10.
label	In client applications, the interface label that users see when prompted to enter the password. For example, SAP Password:
description optional	In client applications, descriptive text about the required password that users see. The description can help the user enter the correct password. The description can be multiline, but it cannot skip lines. The description must be entered on the line below the associated PASSWORD tag.

Examples

Create a password input parameter for a Direct Link SAP query

The analytic header specifies a password input parameter that prompts the user to enter an SAP password. The stored password is used in the subsequent RETRIEVE command in the body of the script.

```
COMMENT
//ANALYTIC SAP Password Example
//PASSWORD 1 SAP Password:
//DATA RSADMIN
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
END
SET SAFETY OFF
RETRIEVE RSADMIN PASSWORD 1
OPEN RSADMIN
SET SAFETY ON
```

> **Note**
>
> The password input parameter and the password parameter in the RETRIEVE command are linked by using the same numerical identifier:
>
> ```
> //PASSWORD 1 SAP Password:
> .
> .
> .
> RETRIEVE RSADMIN PASSWORD 1
> ```

Create a password input parameter for an export to Results

The analytic header specifies a password input parameter that prompts the user to enter a HighBond password. The stored password is used in the subsequent EXPORT command in the body of the script.

```
COMMENT
//ANALYTIC HighBond Password Example
//PASSWORD 3 HighBond Password:
END
SET SAFETY OFF
OPEN AR_Exceptions
EXPORT FIELDS No Due Date Ref Amount Type ACLGRC PASSWORD 3 TO "10926@us"
SET SAFETY ON
```

Remarks

Password storage and encryption

Password values are associated with individual users, and are encrypted at rest. Passwords remain secure throughout analytic script processing, and are encrypted in any temporary files created in the deployment environment.

Testing in Analytics

If you test an analytic script that has one or more PASSWORD tags in Analytics, Analytics automatically generates a PASSWORD command and prompts you to enter the appropriate password. This auto-generated

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

command saves you the labor of inserting PASSWORD commands in the script portion of an analytic script for the purposes of testing, and then removing them again before delivering the analytic script to users.

The auto-generated PASSWORD command is saved in the log, without the password value.

Password values are not saved when you run an analytic script in Analytics, so you must specify the password or passwords each time you run the analytic script, including running or stepping through the analytic script from the cursor position.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

DATA

Specifies that an Analytics table output by an analytic script is copied to a data subfolder (a storage location) in the deployment environment.

Typically, you store Analytics tables so that they can be used as input tables for subsequent analytic scripts.

> **Note**
>
> ACL Robotics with a cloud-based Robots Agent does not include a storage location for Analytics tables. The //DATA tag is ignored in analytic scripts run with a cloud-based agent.

Syntax

```
//DATA table_name <description>
```

Parameters

Name	Description

table_name	The name of the Analytics table to be stored. The value of *table_name* cannot contain any spaces.
	
	Note
	The *table_name* value must exactly match the name of the Analytics output table in the analytic script. You are not naming a table with *table_name*, you are matching a table name specified in the script.
	You can use wildcard characters in *table_name* to assist with matching a table name in the script.
	You cannot use a variable for *table_name*.
	
	You must specify the table name, not the source data file name.
	Correct:
	`//DATA Missing_Checks`
	Incorrect:
	`//DATA Missing_Checks.fil`
	
	Note
	If an existing Analytics table in the data subfolder has the same name as the value you specify, the existing table is overwritten.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description
	## Wildcard characters You can use wildcard characters in *table_name* if part of the table name may change. For example, if the table name depends on the month (invoices-jan, invoices-feb, and so on), specifying invoices-* ensures that the table is copied to the data subfolder regardless of the month suffix. You can specify a single wildcard character to copy all Analytics output tables in the analytic script to the data subfolder: `//DATA *` **Caution** Be careful when using wildcards characters. You may unintentionally overwrite existing data tables if the wildcard pattern that you specify matches unintended tables. As a best practice, make the value of *table_name* as specific as possible. Use wildcard characters only where they are required. ## Uploads to Robots For information about uploads to Robots, see "Uploads to the cloud-based Robots app" on page 914.
description optional	Descriptive text about the output table or other information. The description can be multiline, but it cannot skip lines. The description appears in the analytic header only and is not visible to end users in client applications.

Examples

Copying an Analytics table to the storage location

The following analytic header specifies that the Invoices table, which is output in the associated script, is copied to the storage location:

```
COMMENT
//ANALYTIC Import Table
//DATA Invoices
END
IMPORT DELIMITED TO Invoices "Invoices.fil" FROM "Invoices.csv" 0 SEPARATOR "," QUALIFIER
"" CONSECUTIVE STARTLINE 1 KEEPTITLE ALLCHAR ALLFIELDS
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Remarks

Storing output tables

Output tables are not automatically copied to the storage location. You must use a DATA tag for each table that you want to store. You can include multiple DATA tags in an analytic header if necessary.

When should I use the DATA tag?

Two situations require use of the DATA tag and storing Analytics tables:

- output tables are used as input for subsequent analytic scripts
- users can select input tables or fields when scheduling an analytic script, or running it ad hoc

> **Note**
>
> If an entire data analysis process is completed using a single analytic script, use of the DATA tag is not required.
>
> The DATA tag is not intended to be used for specifying result tables. Use the RESULT tag instead. For more information, see "RESULT" on page 915.

Output tables are used as input for subsequent analytic scripts

Depending on the deployment environment, and the structure of associated scripts, you may need to use the DATA tag to store an Analytics output table that you want to use in a subsequent analytic script.

During analytic script processing, Robots and AX Server use a temporary directory to store and access Analytics output tables, so you may not need to use the DATA tag.

The table below provides guidance.

Deployment environment	Use the **DATA** tag if…	Do not need the **DATA** tag if…
Robots (Enterprise Edition only)	◦ an Analytics table output in one robot task is required as input in another robot task	◦ an Analytics table is output and subsequently input during a sequence of analytic scripts run in a single robot task ◦ an entire data analysis process is completed using a single analytic script
AX Server	◦ an Analytics table output by one analytic script is required as input for another analytic script	◦ an entire data analysis process is completed using a single analytic script
Analysis App window	◦ an Analytics table output by one analytic script is required as input for another analytic script	◦ an entire data analysis process is completed using a single analytic script

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Users can select input tables or fields

The TABLE and FIELD analytic tags create input parameters that allow a user to select an Analytics table, and to select fields from the table, for use as input to an analytic script. However, a table must pre-exist in the storage location in order to be available to be selected.

If you are developing an analytic script that allows a user to choose one or more input tables and fields, a prior analytic script with the DATA tag must run and save the appropriate table or tables to the storage location.

Locate output tables in the Source tables section in Robots

You can optionally add the src_ prefix to an output table name to locate the output table in the **Source tables** section of the **Input/Output** tab in Robots.

```
//DATA src_Invoices
```

You must add the prefix to the table name in both the //DATA tag and in the accompanying script.

The **Source tables** section allows you to visually separate tables that provide input for subsequent scripts. If no output table names have the src_ prefix, the **Source tables** section does not appear in the **Input/Output** tab and all tables are located by default in the **Other tables** section.

Uploads to the cloud-based Robots app

With analytic scripts run in Robots installations, specifying DATA uploads the table layout only (field name, data type, field length) from the on-premise Robots Agent to the cloud-based Robots app in HighBond. Table data remains on your organization's network, within the Robots Agent directory.

All information is encrypted in transit.

Overwriting linked or shared tables in AX Server

If an output table overwrites a linked or shared table in AX Server, the table is changed to a standalone table.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

RESULT

Specifies the output results of an analytic script that you want to make available to end users in client applications.

Output results, even if they exist, are not automatically made available.

Syntax

```
//RESULT type name
  <description>
```

Parameters

Name	Description
type	The type of result item: ◦ TABLE - an Analytics table and associated data file (.fil) ◦ LOG - an analytic log file ◦ FILE - a non-Analytics file **Note** Specifying the LOG or FILE parameters causes data to be uploaded from an on-premise Robots Agent to the cloud-based Robots app in HighBond. For information about uploads to Robots, see "Uploads to the cloud-based Robots app" on page 918.
name	The name of the result item. The *name* value cannot contain any spaces. **Note** The *name* value must exactly match the name of the result item in the analytic script. You are not naming an item with *name*, you are matching a name specified in the script. You can use wildcard characters in *name* to assist with matching a name in the script. You cannot use a variable for *name*. ### Table name The *name* value specifies an Analytics table name. You must specify the table name, not the source data file name. Correct: `//RESULT TABLE Missing_Checks`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Name	Description

	Incorrect:
	`//RESULT TABLE Missing_Checks.fil`
	## Log name
	Optional. The *name* value specifies an analytic log file name. If you do not specify *name*, the default log name is used: *analytic_name*.log.
	> **Note**
	> If you specify a log name, SET LOG TO *log_name* must appear in the script.
	## File name
	The *name* value specifies a non-Analytics file name.
	You must specify the appropriate file extension for the type of non-Analytics file being output.
	Correct:
	`//RESULT FILE Missing_Checks.xlsx`
	Incorrect:
	`//RESULT FILE Missing_Checks`
	## Wildcard characters
	Use one or more wildcard characters in *name* to assist with matching a table, log, or file name in the script. Use a single asterisk (`*`) to substitute for zero or more consecutive characters.
	Patterns created by mixing wildcard and literal characters allow you to match all items of a particular type (for example, `*.xlsx`), or items where part of the name may change based on a variable definition in the script.

description optional	Descriptive text about the result or other information. The description can be multiline, but it cannot skip lines. The description appears in the analytic header only and is not visible to end users in client applications.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Examples

Basic examples

RESULT tag for an Analytics table:

```
//RESULT TABLE Missing_Checks
```

RESULT tag for an analytic log with the default name:

```
//RESULT LOG
```

RESULT tag for an analytic log with a specified name:

```
//RESULT LOG My_log_name
.
.
.
SET LOG TO My_log_name
```

RESULT tag for a specific Excel file:

```
//RESULT FILE Missing_Checks.xlsx
```

RESULT tag for all Excel files:

```
//RESULT FILE *.xlsx
```

Advanced examples

Table name with a month that varies

An output table name includes the month (invoices-jan, invoices-feb, and so on), so you specify `invoices-*` to ensure that the table is made available in the results regardless of the month suffix:

```
//RESULT TABLE invoices-*
```

Log name with a date that varies

A log file name includes a datestamp (prepare_invoice_table_31072019, and so on), so you specify `prepare_invoice_table_*` to ensure that the log file is made available in the results regardless of the datestamp:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
//RESULT LOG prepare_invoice_table_*
```

File name with a month that varies

An output file name includes the month (invoices-jan.xlsx, invoices-feb.xlsx, and so on), so you specify `invoices-*.xlsx` to ensure that the file is made available in the results regardless of the month suffix:

```
//RESULT FILE invoices-*.xlsx
```

File name with a month and a format that vary

An output file name includes the month and is output in different formats (invoices-jan.xlsx, invoices-jan.del, and so on), so you specify `invoices-*.*` to ensure that the files are made available in the results regardless of the month suffix or file type:

```
//RESULT FILE invoices-*.*
```

Remarks

Uploads to the cloud-based Robots app

For Robots installations that use an on-premise Robots Agent, specifying `RESULT LOG` or `RESULT FILE` in an analytic script causes analytic log files or non-Analytics files to be uploaded to the cloud-based Robots app in HighBond. All information is encrypted in transit, and in the cloud-based Robots app.

For more information about logs, see "How log files are output" below.

Specifying `RESULT TABLE` uploads the table layout only (field name, data type, field length). Result table data remains on your organization's network, within the Robots Agent directory.

How log files are output

How log files for analytic scripts are output depends on whether a script succeeded or failed, and on which environment the script is running in.

Analytic script	Robots Agent	AX Server	Analysis App window
Succeeded	◦ **RESULT LOG specified** log file uploaded to cloud-based Robots app ◦ **RESULT LOG not specified** no log file	◦ **RESULT LOG specified** log file output to AX Server (available in client applications) ◦ **RESULT LOG not specified**	◦ **RESULT LOG specified** log file output to Results tab ◦ **RESULT LOG not specified** no log file

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytic script	Robots Agent	AX Server	Analysis App window
		no log file	
Failed	◦ **RESULT LOG tag not considered** • log file automatically output to Robots Agent base data folder (configuration setting = "false") • log file automatically uploaded to cloud-based Robots app (configuration setting = "true" (default)) See configuration setting **UploadLogsWhenFailed** in Configuring a Robots Agent.	◦ **RESULT LOG tag not considered** log file automatically output to AX Server (available in client applications)	◦ **RESULT LOG tag not considered** log file automatically output to Results tab

Result file size limitation on AX Server

Result files are limited to a maximum of 2 GB for analytic scripts running on AX Server. If the file exceeds this size, results are not saved.

Result file storage and availability during script execution on AX Server

When you use the //RESULT FILE tag, the file that is created is available for download from AX Web Client and AX Client once your script completes. This file is stored in the AX database and is not available on the file system of AX Server when the script is not running.

During the execution of your script, the file is available temporarily on the file system of AX Server and you can work with it using external processes, such as those you invoke using the EXECUTE command. While your script is running, external processes can access the file from the analytic job subfolder.

> **Note**
>
> By default, analytic job subfolders are located inside `ACL\Data\jobs`. Once the script completes, the analytic job subfolder is removed and the file is stored in the database.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

PUBLISH

Specifies a file that contains metadata defining which Analytics tables to publish to AX Exception when an analytic script is finished processing.

Syntax

```
//PUBLISH filename
```

Parameters

Name	Description
filename	The name of the file containing publishing metadata for AX Exception.

Examples

The analytic header and the text file that specifies the publishing details for the analytic script

The FILE tag is required if the publish file is stored in the AX folder, so that the file is retrieved when the analytic script is processed.

```
COMMENT
//ANALYTIC Publish Results
//RESULT TABLE Results
//FILE ex_publish.txt
//PUBLISH ex_publish.txt
END
EXTRACT RECORD TO Results
```

The `ex_publish.txt` file uploaded to the **Related Files** subfolder in the collection contains the following line of text. The values must be quoted, and must use the following syntax: "table name","entity name","analytic name"

```
"Results","MyEntity","MyAnalytic"
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Appendix

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

System requirements

Before installing Analytics, ensure that your computer meets the minimum software and hardware requirements.

Software requirements

> **Note**
>
> Some software prerequisites are automatically installed if not already present on your computer. For a complete list of automatically installed prerequisites, see the online documentation.

Requirement	Additional information

Hardware requirements

> **Note**
>
> The best Analytics performance in a production environment may require greater resources than the minimum specification.

Component	Minimum	Recommendation
Processor	1.8 GHz	
Memory (RAM)	2 GB	◦ 64-bit operating systems: 8 GB or more, especially if sorting large files ◦ 32-bit operating systems: 4 GB, especially if sorting large files
Hard disk space (Analytics application files)	1.1 GB	
Hard disk space (software prerequisites)	8 GB	
Hard disk space (data storage)		100 GB or more In addition to the hard disk space required to install Analytics application files and prerequisites, significant additional space is required if a computer will be used to store data extracts, flat files, and results.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Installing ACL for Windows

Follow the ACL for Windows installation instructions to install or upgrade your copy of Analytics. Once the installation completes, you must activate your license.

> **Note**
>
> When you install or upgrade Analytics, any existing Analytics sample data files are over-written if they are in the Analytics working directory you specify during the installation or upgrade.
>
> If you have made changes to any of the sample projects or data files that you want to keep, save the files elsewhere before installing or upgrading, or rename the folder containing them. Do the same with the associated command log files if you want to preserve them.

Obtain the installer

Obtain your installer from your organization's primary contact with Galvanize. If you are the primary contact:

- **Internet access** – obtain the installer from the downloads page in Launchpad (http://www.high-bond.com/)
- **No Internet access** – ask your Galvanize representative to help you obtain the installer from Launchpad

Extract the installer

1. Double-click the ACL for Windows installation package (`ACLforWindows142.exe`).
2. If a security warning dialog box appears, verify the information listed, and click **Yes**.
3. Select the language you want to use for your installation and click **OK**.
4. In the **Setup Extraction Location** page, click **Extract**.

   After the files are extracted the installer starts automatically.

Install any prerequisites

If you are prompted to install prerequisites, click **Install**.

After the prerequisites are installed, the installer automatically proceeds.

Install ACL for Windows

Follow the onscreen instructions to perform the ACL for Windows installation.

In the **ACL Edition Selection** page, select the edition you want to install:

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- **Non-Unicode**
- **Unicode**

For more information about Unicode and non-Unicode editions, see "Unicode versus non-Unicode editions" on page 926.

Activate your license

If you have access to the Internet, you can activate the first time you start ACL for Windows and sign in to your organization. To activate your license offline, contact Galvanize Support.

Start Analytics

To start Analytics, do one of the following:

To create a new, empty Analytics project	Under **Create**, click **Analytic Project**
To open an existing Analytics project	Under **Open**, click **Analytic Project**
To open a recent or a sample Analytics project (.acl)	Under **Recent Analytics Files**, or **Sample Files**, click the name of a project

Install Python version 3.5.x (32-bit)

1. From the Python downloads page, download a version of Python to your computer or the server.

   > **Note**
   >
   > We recommend using Python 3.5 because it has been tested and verified to work correctly with Analytics or the Robots Agent.
   >
   > Any version of Python from 3.3.x onward should also work, however we do not offer the same guarantee of testing and support as we do with version 3.5.x.

2. On your computer or the server, double-click the installer.
3. In the installer, select **Add Python *versionNumber* to PATH**.
4. Click **Install** and follow the on-screen instructions.
5. Reboot the computer or the server before running any Python scripts called by an Analytics script.

Set the ACLPYTHONDLL and PYTHONPATH environment variables

1. In the C:\ drive of the operating system, create one or more folders to house your Python scripts.

   Example - `C:\python_scripts`

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

2. From the operating system, open the **System Properties** dialog box and click **Environment Variables**.
3. In the **System variables** section, click **New** and enter the following variables:

Variable name	Variable value
PYTHONPATH	The full path to the folder(s) you created to house the Python scripts. Separate multiple folder paths with a semi-colon. Example: `C:\python_scripts;C:\dev;C:\tmp`
ACLPYTHONDLL	The full path and filename of the Python DLL file in the Python installation folder that you want to use with Analytics or the Robots Agent. Example: `c:\python_install\python33.dll` If you are using Python 3.3.x, the following restrictions apply: ◦ Unicode characters are not supported in the path for European platforms ◦ Extended characters are not supported in the path for Asian platforms **Note** Python adds the DLL to the system folder (`c:\windows\system32\python33.dll`) rather than the installation folder. You must copy the DLL from the system folder to the installation folder, and use this as the variable value so that Analytics can access the DLL. You may also need to remove any read only settings from the installation folder. If you do not set this value, Analytics or the Robots Agent attempts to use the default supported version 3.5.x DLL `python35.dll`.

4. To save the variable, click **OK** and then in the **System Properties** dialog box, click **OK**.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Unicode versus non-Unicode editions

Unicode editions of Analytics products allow you to view and work with files that contain Unicode data. Unicode is an industry-standard method of character encoding that supports most of the world's languages.

Should I install the non-Unicode or the Unicode edition?

Analytics is available in non-Unicode and Unicode editions. Both editions are contained in the same installation package, and during the installation you specify which edition to install.

You should install the non-Unicode edition, unless you have a requirement to view or analyze Unicode data. Unicode data can only be opened in the Unicode edition of Analytics.

You are more likely to encounter Unicode data if you work in an environment with global information systems, or you analyze data that contains multiple languages.

When the Unicode edition is required

You need to install the Unicode edition to view or analyze data with:

- Asian characters
- a combination of non-Unicode, or traditional, character encodings

  For example, some combination of languages from at least two of these character encodings:

  - Latin 1 (English and Western European)
  - Latin 2 (Central European)
  - Cyrillic
  - Greek
  - Arabic

> **Note**
>
> If you want to use the Chinese, Japanese, or Polish Analytics user interface the only option is to install the Unicode edition. This requirement is related to the language of the user interface, not to the language of the data.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Converting analytic scripts to Unicode

If you are migrating from the non-Unicode edition of Analytics to the Unicode edition, existing regular scripts and analytic scripts are automatically converted to Unicode. However, you must verify that the logic of the scripts remains the same when applied to double-byte Unicode data.

What is Unicode?

Unicode is a standard for encoding text that uses two or more bytes to represent each character, and characters for all languages are contained in a single character set. The Unicode editions of Galvanize products allow you to view and work with files and databases that contain Unicode-encoded data in all modern languages.

> **Note**
>
> Analytics and the AX Engine support little-endian (LE) encoded Unicode data. These products cannot be used to analyze big-endian (BE) encoded data.

Migrating to Unicode Analytics Exchange

- encryption of Unicode scripts is not currently supported
- Analytics project files and log files are encoded as Unicode data (UTF-16 LE) and cannot be used with the non-Unicode edition of Analytics
- when you use Analytics to define print image and delimited files that contain ASCII or EBCDIC-encoded text, the fields in the Analytics table containing this data are assigned the Unicode data type by default

Required analytic scripts changes

Update any parameters that specify a value in bytes

Characters in the non-Unicode edition of Analytics are one byte in length. Characters in the Unicode edition, if they are Unicode data, are two bytes in length. When you specify field length or starting position in bytes in the non-Unicode edition of Analytics, the number of bytes is equal to the number of characters. This is not true for Unicode data in the Unicode edition of Analytics.

To convert analytic scripts for use in Unicode Analytics, you must adjust the numeric value of any parameters that specify field length or starting position in bytes. For example, for an IMPORT DELIMITED command that specifies a WID value of 7 in non-Unicode Analytics, you must double the WID value to 14 to produce the same result in Unicode Analytics.

In addition, for Unicode data, specify an odd-numbered starting byte position for fields, and an even number of bytes for field lengths. Specifying an even-numbered starting position, or an odd-numbered length, can cause characters to display incorrectly.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Recreate all instances of IMPORT PRINT and IMPORT DELIMITED

You need to recreate all instances of the IMPORT PRINT and IMPORT DELIMITED commands by importing the source data file using the Data Definition Wizard in the Unicode version of Analytics and reimporting the projects into AX Server. Using the Data Definition Wizard ensures that all syntax is valid.

Change all instances of the ZONED( ) and EBCDIC( ) functions

You need to change all instances of the ZONED() and EBCDIC() functions as follows so that the ASCII values returned by the functions are correctly converted to Unicode data:

- **Computed fields** – wrap the BINTOSTR() function around ZONED() or EBCDIC() instances
- **Static expressions** – wrap the BINTOSTR() function around ZONED() instances

```
BINTOSTR(ZONED(%result%, 5), 'A')
```

Change all instances of the OPEN FORMAT command

You need to modify all instances of the OPEN FORMAT command. You need to use the SKIP parameter to skip the first two bytes of the Unicode file you are opening. This is required because the first two bytes of UTF-16 encoded files are reserved as byte order marks and are separate from the text in the file.

Non-Unicode

```
OPEN "ascii_test.txt" FORMAT template_table CRLF
DEFINE FIELD full_rec ASCII 1 10
```

Unicode

```
OPEN "utf-16_test.txt" FORMAT template_table CRLF SKIP 2
DEFINE FIELD full_rec UNICODE 1 20
```

Verifying converted analytic scripts

Verify that the Unicode versions of the analytic scripts produce results that are identical to the results produced by the non-Unicode analytic scripts. The best way to do this is to use a Diff tool to compare the log files produced in the analysis. The Diff tool identifies any differences between the files.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

What if the results are not the same?

If you cannot produce the same results with the Unicode version of an analytic script as the non-Unicode version, you may be able to isolate the problem by comparing the log outputs of the scripts at each step of the analysis.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Checking for Unicode compatibility

When upgrading to a Unicode edition, you need to verify that any custom logic you have added to scripts will produce the same results when run against Unicode data. There are predictable areas where scripts may be affected when they are run against Unicode data.

Bit and Character functions

Each of the functions listed below returns values based on byte locations or byte counts. You need to check to ensure that these functions are still being used correctly when you move from the single-byte representation of characters used in the non-Unicode edition to the double-byte character encoding used for Unicode data:

- ASCII( )
- BIT( )
- BYTE( )
- CHR( )
- DIGIT( )
- HEX( )
- MASK( )
- SHIFT( )

Byte length does not equal character length

You need to check the way the following functions are used in your scripts to ensure that they do not assume a one-to-one correspondence between the number of characters in data and the number of bytes.

If you find any instances where the logic assumes a one-to-one correspondence between characters and bytes, you must adjust the logic to work correctly with Unicode data, which uses two bytes to represent each character. Numbers specified as string function parameters, such as 4 in `STRING(1000, 4)` refer to the number of characters, so standard usage of these functions will not cause problems.

Conversion Functions

- PACKED( )
- STRING( )
- UNSIGNED( )

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- VALUE( )
- ZONED( )

String functions

- AT( )
- BLANKS( )
- INSERT( )
- LAST( )
- LENGTH( )
- REPEAT( )
- SUBSTRING( )

Miscellaneous functions

- FILESIZE( )
- LEADING( )
- OFFSET( )
- RECLEN( )

Substituting Unicode-specific functions

Galvanize Unicode products support six Unicode-specific functions that support conversions between non-Unicode and Unicode data. The following functions are available in Galvanize Unicode products:

- **BINTOSTR()** – converts ZONED or EBCDIC data to its corresponding Unicode string. This ensures that values encoded as ZONED or EBCDIC data can be displayed correctly
- **DHEX( )** – returns the hexadecimal equivalent of a specified Unicode field value. This function is the inverse of HTOU( )
- **DBYTE( )** – returns the Unicode character interpretation of a double-byte character at a specified position in a record
- **DTOU( )** – converts a date value to the correct Unicode string display based on the specified locale setting
- **HTOU( )** – returns the Unicode equivalent of a specified hexadecimal string. This function is the inverse of DHEX( )
- **UTOD( )** – converts a locale-specific Unicode string to an Analytics date value

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Running R scripts on AX Server

Import external R scripts as related files along with an analysis app and then call the R scripts from your analytic scripts to leverage the statistical analysis capabilities of the R scripting language on the server. To prepare the AX Server environment to run R scripts, you must first install R and then add the `.r` extension to the file extension whitelist.

Prerequisites

To run R scripts on AX Server, you must:

1. Install a supported version of the R scripting language on your AX Server computer.
2. Add the `.r` extension to the file extension whitelist on AX Server.
3. In Analytics, create a project to work with and import into AX Server.

> **Note**
>
> For help completing these prerequisites, contact your Analytics Exchange administrator and see:
>
> - AX Server requirements
> - Whitelisting file extensions

Add R scripts to the Analytics project directory

After you create your Analytics project in Analytics, copy the R scripts you intend to use and paste them into the project folder so that you can test your script locally in Analytics before importing it to Analytics Exchange.

Example R files

The following example R files contain trivial scripts that concatenate two strings and return a single string joined by a space character. These examples are intended to show how R scripts run on AX Server, not how to analyze data with R.

analysis_a.r

```
conc<-function(x, y) {
  paste(x, y, sep=" ")
}
print(conc(value1, value2))
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

analysis_b.r

```
conc<-function(x, y) {
  paste(y, x, sep=" ")
}
print(conc(value1, value2))
```

Create an Analytics script

In your Analytics project, create a new script to use as the analytic script you run on AX Server. This script does the following:

1. Opens a temporary table called t_tmp with one record.

   You must open a table to execute the EXTRACT command in Analytics, here the t_tmp table is used only for this purpose.
2. Uses the EXTRACT command to run each R script and writes the results to a table.

Add the analytic header

Add the appropriate analytic header tags at the beginning of the script so that the Analytics script can run on AX Server after you import your analysis app. You must add a FILE tag for any R script you intend to run from the analytic script:

```
COMMENT
//ANALYTIC R integration test
  verify R integration on AX Server
//DATA t_tmp
//FILE analysis_a.r
//FILE analysis_b.r
//RESULT TABLE results
END
```

Add the script logic

```
SET SAFETY OFF
DEL ALL OK
CLOSE PRIMARY SECONDARY

OPEN t_tmp

COM **** execute R scripts and write results to table
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

```
EXTRACT FIELDS RSTRING("a<-source('./analysis_a.r');a[[1]]",50,"test","value") AS "value" TO
"results.fil"
EXTRACT FIELDS RSTRING("a<-source('./analysis_b.r');a[[1]]",50,"test","value") AS "value" TO
"results.fil" APPEND

CLOSE t_tmp
```

The complete analytic script

The complete analytic script that you run on AX Server looks as follows:

```
COMMENT
//ANALYTIC R integration test
 verify R integration on AX Server
//DATA t_tmp
//FILE analysis_a.r
//FILE analysis_b.r
//RESULT TABLE results
END

SET SAFETY OFF
DEL ALL OK
CLOSE PRIMARY SECONDARY

OPEN t_tmp

COM **** execute R scripts and write results to table
EXTRACT FIELDS RSTRING("a<-source('./analysis_a.r');a[[1]]",50,"test","value") AS "value" TO
"results.fil"
EXTRACT FIELDS RSTRING("a<-source('./analysis_b.r');a[[1]]",50,"test","value") AS "value" TO
"results.fil" APPEND

CLOSE t_tmp
```

Import the Analytics project and related R files

Once you have authored the analytic script:

1. In AX Client, create a collection and folder to house the Analytics project.
2. To import the project and R files:
   a. Right-click the folder you created and select **Import**.
   b. Navigate to your Analytics project on your local computer and select the `.acl` project file and the `.r` R scripts.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

> **Note**
>
> Make sure you select the R files in the project folder as well as the Analytics project using **Ctrl+click** so that they are imported into AX Server. You must also import the source data files for the t_tmp table.

c. Click **Open**.

Server explorer after import

- *collectionName*
  - *folderName*
    - **Analysis Apps**
      - *ACLProjectName*
        - *analyticScriptName*
    - **Data**
      - t_tmp
    - **Related Files**
      - analysis_a.r
      - analysis_b.r

Run the analytic script

From the **Server Explorer** of AX Client, right-click the analytic script and select **Run**. The R scripts are executed as part of the analytic script and you can access the results results table from AX Web Client.

Results

Server explorer after running the analytic script

- *collectionName*
  - *folderName*
    - **Analysis Apps**
      - *ACLProjectName*
        - *analyticScriptName*
    - **Data**
      - results
    - **Related Files**
      - analysis_a.r
      - analysis_b.r

Results table

- **value**
- test value
- value test

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Running Python scripts on AX Server

Have an Analytics Exchange administrator upload external Python scripts to the PYTHONPATH directory of AX Server and then call the Python scripts from your analytic scripts to leverage the object-oriented features of the Python programming language on the server. To prepare the AX Server environment to run Python scripts, you must first install Python, and then set the PYTHONPATH environment variable.

Prerequisites

To run Python scripts on AX Server, you must:

1. Install a supported version of the Python scripting language on your AX Server computer.
2. Set the PYTHONPATH environment variable on AX Server.
3. In Analytics, create a project to work with and import into AX Server.

> **Note**
>
> For help completing these prerequisites, contact your Analytics Exchange administrator and see:
>
> - AX Server requirements
> - Configuring Python for use with AX Server

Create a Python script

After you create your Analytics project in Analytics, create a Python script that you can call from an analytic script.

Then, give your Analytics Exchange administrator this script file to upload to the PYTHONPATH directory of the machine hosting AX Server before you call the Python script from an analytic script. When the analytic script runs on AX Server, the Python executable looks for the script in the PYTHONPATH directory, so it must be present.

Example Python file

The following example Python file contains a trivial script that uses a lambda expression to raise a number to the power of itself. This example is intended to show how Python scripts run on AX Server, not how to analyze data with Python.

Filename: lambda_example.py

```
# myFunc squares value1 and returns the value
myFunc = lambda value1: value1**2
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Create an Analytics script

In your Analytics project, create a new script to use as the analytic script you run on AX Server. This script does the following:

1. Opens a simple table called `py` with one record.

   You must open a table to execute the `GROUP` command in Analytics, here the `py` table is used only for this purpose.

2. Loops 10 times and in each loop, executes the Python script by passing the incrementing counter as an argument and extracting the output to a results table.

Add the analytic header

Add the appropriate analytic header tags at the beginning of the script so that the Analytics script can run on AX Server after you import your analysis app:

```
COMMENT
//ANALYTIC Python integration test
 verify Python integration on AX Server
//DATA py
//DATA results
//RESULT TABLE results
END
```

Add the script logic

```
SET SAFETY OFF
DEL ALL OK
CLOSE

OPEN py

GROUP
 ASSIGN v_max = 11
 ASSIGN v_counter = 1
 LOOP WHILE v_counter < v_max
  EXTRACT PYNUMERIC("lambda_example,myFunc",0,v_counter) AS "Results value" TO "res-
ults.fil"
  v_counter = v_counter + 1
 END
END
CLOSE py
```

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

The complete analytic script

The complete analytic script that you run on AX Server looks as follows:

```
COMMENT
//ANALYTIC Python integration test
 verify Python integration on AX Server
//DATA py
//DATA results
//RESULT TABLE results
END

SET SAFETY OFF
DEL ALL OK
CLOSE

OPEN py

GROUP
 ASSIGN v_max = 11
 ASSIGN v_counter = 1
 LOOP WHILE v_counter < v_max
  EXTRACT PYNUMERIC("lambda_example,myFunc",0,v_counter) AS "Results value" TO "res-
ults.fil"
  v_counter = v_counter + 1
 END
END
CLOSE py
```

Import the Analytics project

Once you have authored the analytic script:

1. In AX Client, create a collection and folder to house the Analytics project.
2. To import the project:
   a. Right-click the folder you created and select **Import**.
   b. Navigate to your Analytics project on your local computer, select the `.acl` project file, and then click **Open**.

> **Note**
>
> Make sure you import the source data files to import the `py` table with your Analytics project.

Server explorer after import

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

- *collectionName*
  - *folderName*
    - **Analysis Apps**
      - *ACLProjectName*
        - *analyticScriptName*
    - **Data**
      - py
    - **Related Files**

Run the analytic script

From the **Server Explorer** of AX Client, right-click the analytic script and select **Run**. The Python script is executed as part of the analytic script and you can access the `results` results table from AX Web Client.

> **Note**
>
> When the script runs, the Python executable looks for the script file in the `PYTHONPATH` directory of the machine hosting AX Server. If your Analytics Exchange administrator has not uploaded the file to this directory, the analytic script fails.

Results

Server explorer after running the analytic script

- *collectionName*
  - *folderName*
    - **Analysis Apps**
      - *ACLProjectName*
        - *analyticScriptName*
    - **Data**
      - py
      - results
    - **Related Files**

Results table

- **Results value**
- 1
- 4
- 9
- 16
- 25
- 36
- 49
- 64
- 81
- 100

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytic engine error codes

The following table lists the error codes that you may encounter when running analytic scripts.

Analytic engine startup errors

Error Code	Description
202	System error.
203	The evaluation period has expired.
204	The evaluation period has expired.
205	Activation failed.
206	Maximum number of sessions reached.
207	Memory initialization problem(s).
209	Unknown script error.
210	Database profile password is missing.
211	Server connection failure.
212	An unsupported command was encountered.
213	A dialog box was generated by the script.
256	The AX Engine failed to start.

Analytic engine error codes

Error Code	Description
1000	No preference file was specified. A new default preference file was created.
1001	There is a problem with the preference file. A new default preference file was created.
1002	The project has been upgraded from an earlier version. A copy was saved with a .old extension.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Error Code	Description
1003	The project file could not be processed. The last saved project was used.
1004	No project file specified.
1005	The specified project file does not exist.
1006	The specified project file is read-only.
1007	The specified project is currently being used by another application.
1008	The specified .old project file cannot be used. You must specify a project file with the .ACL extension.
1009	The specified project file is not an Analytics project file.
1011	The specified project file cannot be saved as an earlier version.
1012	Unable to open the log file for writing.
1013	No script was specified.
1014	The specified script does not exist.
1015	The Analytics license was not found or is invalid.
1016	A required library file (.dll) was not found.
1017	An unknown error occurred.

Command errors

The following table lists the error codes that are returned when an analytic script fails because of an invalid ACLScript command. The error code number returned identifies the command that failed.

Error Code	Command
1	SAMPLE
2	EXTRACT
3	LIST
4	TOTAL

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Error Code	Command
5	DEFINE
6	COMMENT
7	QUIT
8	STOP
9	BYE
10	USE
11	OPEN
12	SAVE
13	DISPLAY
14	ACTIVATE
15	CLOSE
16	HELP
17	COUNT
18	STATISTICS
19	HISTOGRAM
20	STRATIFY
21	SUMMARIZE
22	EXPLAIN
23	GROUP
24	ELSE
25	END
26	CANCEL
27	SUBMIT

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Error Code	Command
28	DELETE
29	RANDOM
30	SORT
31	FIND
32	DIRECTORY
33	TYPE
34	DUMP
35	INDEX
37	SET
40	DO
41	TOP
42	EXPORT
43	VERIFY
44	SEEK
45	JOIN
46	MERGE
47	SEQUENCE
48	CALCULATE
49	PRINT
50	LOCATE
51	RENAME
54	COPY
55	REPORT

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Error Code	Command
56	EJECT
58	LET
59	ACCUMULATE
63	ACCEPT
64	ASSIGN
65	AGE
66	CLASSIFY
67	PROFILE
68	DO REPORT
69	LOOP
70	PAUSE
71	SIZE
72	EVALUATE
73	DIALOG
74	IF
75	GAPS
76	DUPS
77	SQLOPEN
78	PASSWORD
79	IMPORT
80	REFRESH
81	NOTIFY
82	CONNECT

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Error Code	Command
83	RETRIEVE
84	FIELDSHIFT
85	BENFORD
86	CROSSTAB
87	(not used)
88	ESCAPE
89	NOTES
90	FUZZY DUPLICATE
91	EXECUTE
92	ACCESSDATA32
93	ACCESSDATA64
94	APPEND
95	RCOMMAND
96	CVSPREPARE
97	CVSSAMPLE
98	CVSEVALUATE
99	OUTLIER
100	FUZZYJOIN
101	CLUSTER
102	TRAIN
103	PREDICT

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytic job processing errors

Error Code	Error message
-10	The analytic results could not be saved because the destination results folder was deleted after the analytic started running.
-11	Job was stopped.
-12	Stopped due to server shut down.
-13	Failed to create results.
-16	Could not run due to server properties configuration error.
-17	Unable to create uniquely named results directory.
-19	Job was skipped.
-20	Could not prepare publish result tables.
-21	Could not publish results to *AX* Exception.
-22	Publish failed. Invalid table name.
-23	Publish failed. One or more of the table's column names are too long.
-24	Publish failed. Invalid values within data cells within an Analytics table.
-25	Publish failed. Not supported data types within table fields.
-26	Publish failed. Could not connect to *AX* Exception server.
-27	Job did not run. The user was removed or does not have permission.
-28	Job did not run. Unexpected error. See the server log and Analytics log for details.
-29	Could not copy data files. The analytic failed because the required data files could not be copied to the jobs directory.
-30	Job did not run. The analytic link is broken.
-31	Publish failed. The exception mapping file could not be located.
-32	Publish failed. Failed to parse the exception mapping file.
-34	Failed to store job results. Check that there is sufficient space on the drive storing the jobs folder and that no data files are locked.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Variables created by Analytics commands

When you execute certain commands, either by entering information in dialog boxes in Analytics or by running scripts, system variables are automatically created by Analytics. You can use these variables, and the values they contain, when processing subsequent Analytics commands.

The value in a system variable is replaced with an updated value if you execute the same command again.

"Analytics system variables" on the next page lists the system variables created by Analytics.

> **Note**
>
> System variables, and the values they contain, are retained for the duration of the current Analytics session only.

Displaying the current value of variables

You can use any of the following methods to display the current values of all system-defined and user-defined variables in an Analytics project:

- Select the **Variables** tab in the **Navigator**
- Enter DISPLAY VARIABLES in the command line
- Click **Display Variables** on the toolbar (requires that you first add the button to the toolbar)

The second and third methods also display the remaining memory available to store variables.

Incrementally numbered system variables

For system variable names that include *n* in "Analytics system variables" on the next page, *n* is always 1 if commands are executed outside a group – for example, **TOTAL1**.

If you use a group to execute multiple commands, any system variables that result are numbered based on the line number of the command that creates the variable. The first command in a group is considered to be on line number 2.

For example:

- If the Total command is the third command in a group, the results are contained in the variable **TOTAL4**.
- If a second Total command is the fifth command in the group, the results are contained in the variable **TOTAL6**.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Analytics system variables

The following table lists the system variables created by Analytics, the commands that generate them, and the values the variables contain.

> **Note**
>
> If you run the Statistics command on more than one field simultaneously, the system variables contain values for the first field you specify.

System variable	Command	Value
WRITE*n*	◦ Any command that outputs a table ◦ Verify ◦ Sequence	◦ The number of records in the output table ◦ The number of data validity errors (Verify) ◦ The number of sequence errors (Sequence)
OUTPUTFOLDER	◦ Any command that outputs an Analytics table	The path to the Analytics project folder in the **Navigator** that contains the output table. This a DOS-style path that uses the format **/folder-name/subfoldername**, in which the initial slash (/) indicates the root level in the **Overview** tab. **Tip** Use the SET FOLDER command to specify a different output folder or to create a new output folder.
COUNT*n*	◦ Count ◦ Statistics	The number of records tallied.
ACL_Ver_Major	◦ Display Version (Analytics version numbers are in the format *major.minor.patch*)	The major version of Analytics that is currently running.
ACL_Ver_Minor		The minor version of Analytics that is currently running.
ACL_Ver_Patch		The patch version of Analytics that is currently running.
ACL_Ver_Type		The edition of Analytics that is currently running: ◦ A value of '0' indicates the non-Unicode edition ◦ A value of '1' indicates the Unicode edition
MLE*n*	◦ Evaluate	**Monetary unit sampling** Most Likely Error amount (projected misstatement) **Record sampling** Upper error limit frequency rate (computed upper deviation rate)
UEL*n*		**Monetary unit sampling** Upper Error Limit amount (upper misstatement limit)

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

System variable	Command	Value
RETURN_CODE	◦ Execute	The code returned by an external application or process run using the Execute command. Return codes are numbers generated by the external application or process and sent back to Analytics to indicate the outcome of the external process. Analytics does not generate the return code. Typical return codes are integer values that map to specific notifications or error messages. For example, the return code "0" could mean "The operation completed successfully". The return code "2" could mean "The system cannot find the file specified". Specific return codes and their meanings vary depending on the external application or process. Lists of return codes, also called 'error codes' or 'exit codes', and their meanings, can often be found in the documentation for the associated external application. Lists of return codes can also be found on the Internet. The RETURN_CODE variable is created when the Execute command is used synchronously, but not when the command is used asynchronously.
GAPDUP*n*	◦ Gaps ◦ Duplicates ◦ Fuzzy Duplicates	The total number of gaps, duplicates, or fuzzy duplicate groups.
SAMPINT*n*	◦ Size	The required sample interval.
SAMPSIZE*n*		The required sample size.
ABS*n*	◦ Statistics	The absolute value of the first specified field.
AVERAGE*n*		The mean value of the first specified field.
HIGH*n*		The 5th highest value in the first specified field. The 5th highest is the default setting. The setting can be changed using the **# of High/Low** option in the **Statistics** dialog box. **Note** When Analytics identifies the highest value, duplicate values are not factored out. For example, if values in descending order are 100, 100, 99, 98, the 3rd highest value is 99, not 98.
LOW*n*		The 5th lowest value in the first specified field. The 5th lowest is the default setting. The setting can be changed using the **# of High/Low** option in the **Statistics** dialog box.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

System variable	Command	Value
		Note When Analytics identifies the lowest value, duplicate values are not factored out. For example, if values in ascending order are 1, 1, 2, 3, the 3rd lowest value is 2, not 3.
MAX*n*		The maximum value in the first specified field.
MEDIAN*n*		The median value in the first specified field.
MIN*n*		The minimum value in the first specified field.
MODE*n*		The most frequently occurring value in the first specified field.
Q25*n*		The first quartile value (lower quartile value) in the first specified field.
Q75*n*		The third quartile value (upper quartile value) in the first specified field.
RANGE*n*		The difference between the maximum and minimum values in the first specified field.
STDDEV*n*		The standard deviation of the first specified field.
TOTAL*n*	◦ Total ◦ Statistics	The sum total of the values in the first specified field.

Other system variables

The following variables are system-generated but not created by commands:

- **AXRunByUser** – available in scripts running on AX Server, this variable stores the username of the user running the analytic in the format "*domain**username*"
- **OUTPUTFOLDER** – the current Analytics project output folder

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Reserved keywords

Analytics reserves certain keywords for special purposes. You cannot name fields or variables with these reserved keyword values.

If you add a suffix to a reserved keyword, then you can use it as a field or variable name. For example, the name "Field" is not permitted, but "Field_1" or "Field_2" are permitted.

> **Note**
>
> In some cases, you are also prevented from using abbreviations of the reserved keywords, such as "Can" (CANCEL), "Form" (FORMAT), or "Rec" (RECORD).

Reserved Keyword	Purpose in Analytics
ALL	Refers to all previously defined fields.
AND	The logical AND operator.
AS	Assigns a display name to the output field or expression.
AXRunByUser	A system variable that stores the username of the user running an analytic script on AX Server in the format "*domain\username*".
CANCEL	Cancels the current command.
D	Specifies a descending sort order for the preceding expression or field name.
END	Concludes the input stream and acts like a null line.
EXPR	The prefix for the name of a default output field.
F	Refers to the *false* value of a logical expression.
FIELD/FIELDS	Part of the EXPORT, EXTRACT, JOIN, and SAMPLE commands.
FORMAT	An older name for an Analytics table layout. Cannot be used as a name for an Analytics table.
IF	Specifies a condition.
LINE	Used by the DEFINE COLUMN command to specify whether a field breaks over a specified number of lines.
NODUPS	Suppresses duplicate display values in the break field in an Analytics report.
NOT	The logical NOT operator.
NOZEROS	Displays or prints zero values in a numeric field or report as blank.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize

Reserved Keyword	Purpose in Analytics
ON	Precedes a field list.
OR	The logical OR operator.
OTHER	Indicates which fields or expressions to include, but not subtotal, in the output of the SUMMARIZE command.
PAGE	Used by the REPORT command to create page breaks.
PICTURE	Specifies a format for a numeric field.
PRIMARY	Specifies a certain type of join.
RECORD	Refers to the entire input record as it exists.
RECORD_LENGTH	Stores record-length values for use in record-processing operations.
SECONDARY	Specifies a certain type of join.
SUPPRESS	Suppresses the output of numeric totals.
T	Refers to the *true* value of a logical expression.
TAPE	Refers to an older method of accessing data with Analytics. Cannot be used as a name for an Analytics table.
TO	Designates an output file for any command.
WIDTH	Changes the default print width of a specified field or expression.

Published 7/15/2020 © 2019 ACL Services Ltd. dba Galvanize