root

Go Up

Table of Contents

BLAS

Go Up

IMPORTS

lib_eclblas |

DESCRIPTIONS

MODULE BLAS

	BLAS

Children

1. Types
2. ICellFunc : Function prototype for Apply2Cell
3. Apply2Cells : Iterate matrix and apply function to each cell
4. dasum : Absolute sum, the 1 norm of a vector
5. daxpy : alpha*X + Y
6. dgemm : alpha*op(A) op(B) + beta*C where op() is transpose
7. dgetf2 : Compute LU Factorization of matrix A
8. dpotf2 : DPOTF2 computes the Cholesky factorization of a real symmetric positive definite matrix A
9. dscal : Scale a vector alpha
10. dsyrk : Implements symmetric rank update C

11. dtrsm : Triangular matrix solver
12. extract_diag : Extract the diagonal of he matrix
13. extract_tri : Extract the upper or lower triangle
14. make_diag : Generate a diagonal matrix
15. make_vector : Make a vector of dimension m
16. trace : The trace of the input matrix

MODULE Types

BLAS \

	Types

Children

1. value_t
2. dimension_t
3. matrix_t
4. Triangle
5. Diagonal
6. Side

ATTRIBUTE value_t

BLAS \ Types \

	value_t

ATTRIBUTE dimension_t

BLAS \ Types \

	dimension_t

ATTRIBUTE matrix_t

BLAS \ Types \

	matrix_t

ATTRIBUTE Triangle

BLAS \ Types \

	Triangle

ATTRIBUTE Diagonal

BLAS \ Types \

	Diagonal

ATTRIBUTE Side

BLAS \ Types \

	Side

FUNCTION ICellFunc

BLAS \

Types.value_t	ICellFunc
(Types.value_t v, Types.dimension_t r, Types.dimension_t c)	

Function prototype for Apply2Cell.

PARAMETER **v** the value

PARAMETER **r** the row ordinal

PARAMETER **c** the column ordinal

RETURN the updated value

FUNCTION Apply2Cells

BLAS \

Types.matrix_t	Apply2Cells
(Types.dimension_t m, Types.dimension_t n, Types.matrix_t x, ICellFunc f)	

Iterate matrix and apply function to each cell

PARAMETER **m** number of rows

PARAMETER **n** number of columns

PARAMETER **x** matrix

PARAMETER **f** function to apply

RETURN updated matrix

FUNCTION dasum

BLAS \

Types.value_t	dasum
(Types.dimension_t m, Types.matrix_t x, Types.dimension_t incx, Types.dimension_t skipped=0)	

Absolute sum, the 1 norm of a vector.

PARAMETER **m** the number of entries

PARAMETER **x** the column major matrix holding the vector

PARAMETER **incx** the increment for x, 1 in the case of an actual vector

PARAMETER **skipped** default is zero, the number of entries stepped over to get to the first entry

RETURN the sum of the absolute values

FUNCTION daxpy

BLAS \

Types.matrix_t	daxpy
(Types.dimension_t N, Types.value_t alpha, Types.matrix_t X, Types.dimension_t incX, Types.matrix_t Y, Types.dimension_t incY, Types.dimension_t x_skipped=0, Types.dimension_t y_skipped=0)	

alpha*X + Y

PARAMETER **N** number of elements in vector

PARAMETER **alpha** the scalar multiplier

PARAMETER **X** the column major matrix holding the vector X

PARAMETER **incX** the increment or stride for the vector

PARAMETER **Y** the column major matrix holding the vector Y

PARAMETER **incY** the increment or stride of Y

PARAMETER **x_skipped** number of entries skipped to get to the first X

PARAMETER **y_skipped** number of entries skipped to get to the first Y

RETURN the updated matrix

FUNCTION dgemm

BLAS \

Types.matrix_t	dgemm
(BOOLEAN transposeA, BOOLEAN transposeB, Types.dimension_t M, Types.dimension_t N, Types.dimension_t K, Types.value_t alpha, Types.matrix_t A, Types.matrix_t B, Types.value_t beta=0.0, Types.matrix_t C=[])	

alpha*op(A) op(B) + beta*C where op() is transpose

PARAMETER **transposeA** true when transpose of A is used

PARAMETER **transposeB** true when transpose of B is used

PARAMETER **M** number of rows in product

PARAMETER **N** number of columns in product

PARAMETER **K** number of columns/rows for the multiplier/multiplicand

PARAMETER **alpha** scalar used on A

PARAMETER **A** matrix A

PARAMETER **B** matrix B

PARAMETER **beta** scalar for matrix C

PARAMETER **C** matrix C or empty

FUNCTION dgetf2

BLAS \

Types.matrix_t	dgetf2
(Types.dimension_t m, Types.dimension_t n, Types.matrix_t a)	

Compute LU Factorization of matrix A.

PARAMETER **m** number of rows of A

PARAMETER **n** number of columns of A

RETURN composite matrix of factors, lower triangle has an implied diagonal of ones. Upper triangle has the diagonal of the composite.

FUNCTION dpotf2

BLAS \

Types.matrix_t	dpotf2
(Types.Triangle tri, Types.dimension_t r, Types.matrix_t A, BOOLEAN clear=TRUE)	

DPOTF2 computes the Cholesky factorization of a real symmetric positive definite matrix A. The factorization has the form A = U**T * U , if UPLO = 'U', or A = L * L**T, if UPLO = 'L', where U is an upper triangular matrix and L is lower triangular. This is the unblocked version of the algorithm, calling Level 2 BLAS.

PARAMETER **tri** indicate whether upper or lower triangle is used

PARAMETER **r** number of rows/columns in the square matrix

PARAMETER **A** the square matrix

PARAMETER **clear** clears the unused triangle

RETURN the triangular matrix requested.

FUNCTION dscal

BLAS \

Types.matrix_t	dscal
(Types.dimension_t N, Types.value_t alpha, Types.matrix_t X, Types.dimension_t incX, Types.dimension_t skipped=0)	

Scale a vector alpha

PARAMETER **N** number of elements in the vector

PARAMETER **alpha** the scaling factor

PARAMETER **X** the column major matrix holding the vector

PARAMETER **incX** the stride to get to the next element in the vector

PARAMETER **skipped** the number of elements skipped to get to the first element

RETURN the updated matrix

FUNCTION dsyrk

BLAS \

Types.matrix_t	dsyrk
(Types.Triangle tri, BOOLEAN transposeA, Types.dimension_t N, Types.dimension_t K, Types.value_t alpha, Types.matrix_t A, Types.value_t beta, Types.matrix_t C, BOOLEAN clear=FALSE)	

Implements symmetric rank update C

PARAMETER **tri** update upper or lower triangle

PARAMETER **transposeA** Transpose the A matrix to be NxK

PARAMETER **N** number of rows

PARAMETER **K** number of columns in the update matrix or transpose

PARAMETER **alpha** the alpha scalar

PARAMETER **A** the update matrix, either NxK or KxN

PARAMETER **beta** the beta scalar

PARAMETER **C** the matrix to update

PARAMETER **clear** clear the triangle that is not updated. BLAS assumes that symmetric matrices have only one of the triangles and this option lets you make that true.

FUNCTION dtrsm

BLAS \

Types.matrix_t	**dtrsm**
`(Types.Side side, Types.Triangle tri, BOOLEAN transposeA, Types.Diagonal diag, Types.dimension_t M, Types.dimension_t N, Types.dimension_t lda, Types.value_t alpha, Types.matrix_t A, Types.matrix_t B)`	

Triangular matrix solver. op(A) X = alpha B or X op(A) = alpha B where op is Transpose, X and B is MxN

PARAMETER **side** side for A, Side.Ax is op(A) X = alpha B

PARAMETER **tri** Says whether A is Upper or Lower triangle

PARAMETER **transposeA** is op(A) the transpose of A

PARAMETER **diag** is the diagonal an implied unit diagonal or supplied

PARAMETER **M** number of rows

PARAMETER **N** number of columns

PARAMETER **lda** the leading dimension of the A matrix, either M or N

PARAMETER **alpha** the scalar multiplier for B

PARAMETER **A** a triangular matrix

PARAMETER **B** the matrix of values for the solve

RETURN the matrix of coefficients to get B.

FUNCTION extract_diag

BLAS \

Types.matrix_t	extract_diag
(Types.dimension_t m, Types.dimension_t n, Types.matrix_t x)	

Extract the diagonal of he matrix

PARAMETER **m** number of rows

PARAMETER **n** number of columns

PARAMETER **x** matrix from which to extract the diagonal

RETURN diagonal matrix

FUNCTION extract_tri

BLAS \

Types.matrix_t	extract_tri
(Types.dimension_t m, Types.dimension_t n, Types.Triangle tri, Types.Diagonal dt, Types.matrix_t a)	

Extract the upper or lower triangle. Diagonal can be actual or implied unit diagonal.

PARAMETER **m** number of rows

PARAMETER **n** number of columns

PARAMETER **tri** Upper or Lower specifier, Triangle.Lower or Triangle.Upper

PARAMETER **dt** Use Diagonal.NotUnitTri or Diagonal.UnitTri

PARAMETER **a** Matrix, usually a composite from factoring

RETURN the triangle

FUNCTION make_diag

BLAS \

Types.matrix_t	make_diag
(Types.dimension_t m, Types.value_t v=1.0, Types.matrix_t X=[])	

Generate a diagonal matrix.

PARAMETER **m** number of diagonal entries

PARAMETER **v** option value, defaults to 1

PARAMETER **X** optional input of diagonal values, multiplied by v.

RETURN a diagonal matrix

FUNCTION make_vector

BLAS \

Types.matrix_t	make_vector
(Types.dimension_t m, Types.value_t v=1.0)	

Make a vector of dimension m

PARAMETER **m** number of elements

PARAMETER **v** the values, defaults to 1

RETURN the vector

FUNCTION trace

BLAS \

Types.value_t	**trace**
(Types.dimension_t m, Types.dimension_t n, Types.matrix_t x)	

The trace of the input matrix

PARAMETER **m** number of rows

PARAMETER **n** number of columns

PARAMETER **x** the matrix

RETURN the trace (sum of the diagonal entries)

BundleBase

Go Up

DESCRIPTIONS

MODULE BundleBase

	BundleBase

Children

1. PropertyRecord
2. Name
3. Description
4. Authors
5. License
6. Copyright
7. DependsOn
8. Version
9. Properties
10. PlatformVersion

RECORD PropertyRecord

BundleBase \

	PropertyRecord

ATTRIBUTE Name

BundleBase \

STRING	**Name**

ATTRIBUTE Description

BundleBase \

UTF8	**Description**

ATTRIBUTE Authors

BundleBase \

SET OF UTF8	**Authors**

ATTRIBUTE License

BundleBase \

UTF8	**License**

ATTRIBUTE Copyright

BundleBase \

UTF8	**Copyright**

ATTRIBUTE DependsOn

BundleBase \

SET OF STRING	**DependsOn**

ATTRIBUTE Version

BundleBase \

STRING	**Version**

ATTRIBUTE Properties

BundleBase \

	Properties

ATTRIBUTE PlatformVersion

BundleBase \

STRING	**PlatformVersion**

Date

Go Up

IMPORTS

DESCRIPTIONS

MODULE Date

	Date

Children

1. Date_rec
2. Date_t
3. Days_t
4. Time_rec
5. Time_t
6. Seconds_t
7. DateTime_rec
8. Timestamp_t
9. Year : Extracts the year from a date type
10. Month : Extracts the month from a date type
11. Day : Extracts the day of the month from a date type
12. Hour : Extracts the hour from a time type
13. Minute : Extracts the minutes from a time type
14. Second : Extracts the seconds from a time type

15. DateFromParts : Combines year, month day to create a date type
16. TimeFromParts : Combines hour, minute second to create a time type
17. SecondsFromParts : Combines date and time components to create a seconds type
18. SecondsToParts : Converts the number of seconds since epoch to a structure containing date and time parts
19. TimestampToSeconds : Converts the number of microseconds since epoch to the number of seconds since epoch
20. IsLeapYear : Tests whether the year is a leap year in the Gregorian calendar
21. IsDateLeapYear : Tests whether a date is a leap year in the Gregorian calendar
22. FromGregorianYMD : Combines year, month, day in the Gregorian calendar to create the number days since 31st December 1BC
23. ToGregorianYMD : Converts the number days since 31st December 1BC to a date in the Gregorian calendar
24. FromGregorianDate : Converts a date in the Gregorian calendar to the number days since 31st December 1BC
25. ToGregorianDate : Converts the number days since 31st December 1BC to a date in the Gregorian calendar
26. DayOfYear : Returns a number representing the day of the year indicated by the given date
27. DayOfWeek : Returns a number representing the day of the week indicated by the given date
28. IsJulianLeapYear : Tests whether the year is a leap year in the Julian calendar
29. FromJulianYMD : Combines year, month, day in the Julian calendar to create the number days since 31st December 1BC
30. ToJulianYMD : Converts the number days since 31st December 1BC to a date in the Julian calendar
31. FromJulianDate : Converts a date in the Julian calendar to the number days since 31st December 1BC
32. ToJulianDate : Converts the number days since 31st December 1BC to a date in the Julian calendar
33. DaysSince1900 : Returns the number of days since 1st January 1900 (using the Gregorian Calendar)
34. ToDaysSince1900 : Returns the number of days since 1st January 1900 (using the Gregorian Calendar)
35. FromDaysSince1900 : Converts the number days since 1st January 1900 to a date in the Julian calendar
36. YearsBetween : Calculate the number of whole years between two dates
37. MonthsBetween : Calculate the number of whole months between two dates

38. DaysBetween : Calculate the number of days between two dates
39. DateFromDateRec : Combines the fields from a Date_rec to create a Date_t
40. DateFromRec : Combines the fields from a Date_rec to create a Date_t
41. TimeFromTimeRec : Combines the fields from a Time_rec to create a Time_t
42. DateFromDateTimeRec : Combines the date fields from a DateTime_rec to create a Date_t
43. TimeFromDateTimeRec : Combines the time fields from a DateTime_rec to create a Time_t
44. SecondsFromDateTimeRec : Combines the date and time fields from a DateTime_rec to create a Seconds_t
45. FromStringToDate : Converts a string to a Date_t using the relevant string format
46. FromString : Converts a string to a date using the relevant string format
47. FromStringToTime : Converts a string to a Time_t using the relevant string format
48. MatchDateString : Matches a string against a set of date string formats and returns a valid Date_t object from the first format that successfully parses the string
49. MatchTimeString : Matches a string against a set of time string formats and returns a valid Time_t object from the first format that successfully parses the string
50. DateToString : Formats a date as a string
51. TimeToString : Formats a time as a string
52. SecondsToString : Converts a Seconds_t value into a human-readable string using a format template
53. ToString : Formats a date as a string
54. ConvertDateFormat : Converts a date from one format to another
55. ConvertFormat : Converts a date from one format to another
56. ConvertTimeFormat : Converts a time from one format to another
57. ConvertDateFormatMultiple : Converts a date that matches one of a set of formats to another
58. ConvertFormatMultiple : Converts a date that matches one of a set of formats to another
59. ConvertTimeFormatMultiple : Converts a time that matches one of a set of formats to another
60. AdjustDate : Adjusts a date by incrementing or decrementing year, month and/or day values
61. AdjustDateBySeconds : Adjusts a date by adding or subtracting seconds
62. AdjustTime : Adjusts a time by incrementing or decrementing hour, minute and/or second values
63. AdjustTimeBySeconds : Adjusts a time by adding or subtracting seconds
64. AdjustSeconds : Adjusts a Seconds_t value by adding or subtracting years, months, days, hours, minutes and/or seconds

65. AdjustCalendar : Adjusts a date by incrementing or decrementing months and/or years
66. IsLocalDaylightSavingsInEffect : Returns a boolean indicating whether daylight savings time is currently in effect locally
67. LocalTimeZoneOffset : Returns the offset (in seconds) of the time represented from UTC, with positive values indicating locations east of the Prime Meridian
68. CurrentDate : Returns the current date
69. Today : Returns the current date in the local time zone
70. CurrentTime : Returns the current time of day
71. CurrentSeconds : Returns the current date and time as the number of seconds since epoch
72. CurrentTimestamp : Returns the current date and time as the number of microseconds since epoch
73. DatesForMonth : Returns the beginning and ending dates for the month surrounding the given date
74. DatesForWeek : Returns the beginning and ending dates for the week surrounding the given date (Sunday marks the beginning of a week)
75. IsValidDate : Tests whether a date is valid, both by range-checking the year and by validating each of the other individual components
76. IsValidGregorianDate : Tests whether a date is valid in the Gregorian calendar
77. IsValidTime : Tests whether a time is valid
78. CreateDate : A transform to create a Date_rec from the individual elements
79. CreateDateFromSeconds : A transform to create a Date_rec from a Seconds_t value
80. CreateTime : A transform to create a Time_rec from the individual elements
81. CreateTimeFromSeconds : A transform to create a Time_rec from a Seconds_t value
82. CreateDateTime : A transform to create a DateTime_rec from the individual elements
83. CreateDateTimeFromSeconds : A transform to create a DateTime_rec from a Seconds_t value

---

RECORD Date_rec

Date \

	Date_rec

ATTRIBUTE Date_t

Date \

	Date_t

ATTRIBUTE Days_t

Date \

	Days_t

RECORD Time_rec

Date \

	Time_rec

ATTRIBUTE Time_t

Date \

	Time_t

ATTRIBUTE Seconds_t

Date \

	Seconds_t

RECORD DateTime_rec

Date \

	DateTime_rec

ATTRIBUTE Timestamp_t

Date \

	Timestamp_t

FUNCTION Year

Date \

INTEGER2	**Year**
(Date_t date)	

Extracts the year from a date type.

PARAMETER **date** The date.

RETURN An integer representing the year.

FUNCTION Month

Date \

UNSIGNED1	**Month**
(Date_t date)	

Extracts the month from a date type.

PARAMETER **date** The date.

RETURN An integer representing the year.

FUNCTION Day

Date \

UNSIGNED1	**Day**
(Date_t date)	

Extracts the day of the month from a date type.

PARAMETER **date** The date.

RETURN An integer representing the year.

FUNCTION Hour

Date \

UNSIGNED1	**Hour**
(Time_t time)	

Extracts the hour from a time type.

PARAMETER **time** The time.

RETURN An integer representing the hour.

FUNCTION Minute

Date \

UNSIGNED1	**Minute**
(Time_t time)	

Extracts the minutes from a time type.

PARAMETER **time** The time.

RETURN An integer representing the minutes.

FUNCTION Second

Date \

UNSIGNED1	**Second**
(Time_t time)	

Extracts the seconds from a time type.

PARAMETER **time** The time.

RETURN An integer representing the seconds.

FUNCTION DateFromParts

Date \

Date_t	DateFromParts
(INTEGER2 year, UNSIGNED1 month, UNSIGNED1 day)	

Combines year, month day to create a date type.

PARAMETER **year** The year (0-9999).

PARAMETER **month** The month (1-12).

PARAMETER **day** The day (1..daysInMonth).

RETURN A date created by combining the fields.

FUNCTION TimeFromParts

Date \

Time_t	TimeFromParts
(UNSIGNED1 hour, UNSIGNED1 minute, UNSIGNED1 second)	

Combines hour, minute second to create a time type.

PARAMETER **hour** The hour (0-23).

PARAMETER **minute** The minute (0-59).

PARAMETER **second** The second (0-59).

RETURN A time created by combining the fields.

FUNCTION SecondsFromParts

Date \

Seconds_t	SecondsFromParts
(INTEGER2 year, UNSIGNED1 month, UNSIGNED1 day, UNSIGNED1 hour, UNSIGNED1 minute, UNSIGNED1 second, BOOLEAN is_local_time = FALSE)	

Combines date and time components to create a seconds type. The date must be represented within the Gregorian calendar after the year 1600.

PARAMETER **year** The year (1601-30827).

PARAMETER **month** The month (1-12).

PARAMETER **day** The day (1..daysInMonth).

PARAMETER **hour** The hour (0-23).

PARAMETER **minute** The minute (0-59).

PARAMETER **second** The second (0-59).

PARAMETER **is_local_time** TRUE if the datetime components are expressed in local time rather than UTC, FALSE if the components are expressed in UTC. Optional, defaults to FALSE.

RETURN A Seconds_t value created by combining the fields.

MODULE SecondsToParts

Date \

	SecondsToParts
(Seconds_t seconds)	

Converts the number of seconds since epoch to a structure containing date and time parts. The result must be representable within the Gregorian calendar after the year 1600.

PARAMETER **seconds** The number of seconds since epoch.

RETURN Module with exported attributes for year, month, day, hour, minute, second, day_of_week, date and time.

Children

1. Year
2. Month
3. Day
4. Hour
5. Minute
6. Second
7. day_of_week
8. date : Combines year, month day to create a date type
9. time : Combines hour, minute second to create a time type

ATTRIBUTE Year

Date \ SecondsToParts \

INTEGER2	**Year**

ATTRIBUTE Month

Date \ SecondsToParts \

UNSIGNED1	**Month**

ATTRIBUTE Day

Date \ SecondsToParts \

UNSIGNED1	**Day**

ATTRIBUTE Hour

Date \ SecondsToParts \

UNSIGNED1	**Hour**

ATTRIBUTE Minute

Date \ SecondsToParts \

UNSIGNED1	**Minute**

ATTRIBUTE Second

Date \ SecondsToParts \

UNSIGNED1	**Second**

ATTRIBUTE day_of_week

Date \ SecondsToParts \

UNSIGNED1	**day_of_week**

ATTRIBUTE date

Date \ SecondsToParts \

Date_t	**date**

Combines year, month day to create a date type.

PARAMETER **year** The year (0-9999).

PARAMETER **month** The month (1-12).

PARAMETER **day** The day (1..daysInMonth).

RETURN A date created by combining the fields.

ATTRIBUTE time

Date \ SecondsToParts \

Time_t	**time**

Combines hour, minute second to create a time type.

PARAMETER **hour** The hour (0-23).

PARAMETER **minute** The minute (0-59).

PARAMETER **second** The second (0-59).

RETURN A time created by combining the fields.

FUNCTION TimestampToSeconds

Date \

Seconds_t	**TimestampToSeconds**
(Timestamp_t timestamp)	

Converts the number of microseconds since epoch to the number of seconds since epoch.

PARAMETER **timestamp** The number of microseconds since epoch.

RETURN The number of seconds since epoch.

FUNCTION IsLeapYear

Date \

BOOLEAN	**IsLeapYear**
(INTEGER2 year)	

Tests whether the year is a leap year in the Gregorian calendar.

PARAMETER **year** The year (0-9999).

RETURN True if the year is a leap year.

FUNCTION IsDateLeapYear

Date \

BOOLEAN	**IsDateLeapYear**
(Date_t date)	

Tests whether a date is a leap year in the Gregorian calendar.

PARAMETER **date** The date.

RETURN True if the year is a leap year.

FUNCTION FromGregorianYMD

Date \

Days_t	FromGregorianYMD
(INTEGER2 year, UNSIGNED1 month, UNSIGNED1 day)	

Combines year, month, day in the Gregorian calendar to create the number days since 31st December 1BC.

PARAMETER **year** The year (-4713..9999).

PARAMETER **month** The month (1-12). A missing value (0) is treated as 1.

PARAMETER **day** The day (1..daysInMonth). A missing value (0) is treated as 1.

RETURN The number of elapsed days (1 Jan 1AD = 1)

MODULE ToGregorianYMD

Date \

	ToGregorianYMD
(Days_t days)	

Converts the number days since 31st December 1BC to a date in the Gregorian calendar.

PARAMETER **days** The number of elapsed days (1 Jan 1AD = 1)

RETURN Module containing Year, Month, Day in the Gregorian calendar

Children

1. year
2. month
3. day

ATTRIBUTE year

Date \ ToGregorianYMD \

	year

ATTRIBUTE month

Date \ ToGregorianYMD \

	month

ATTRIBUTE day

Date \ ToGregorianYMD \

	day

FUNCTION FromGregorianDate

Date \

`Days_t`	**FromGregorianDate**
`(Date_t date)`	

Converts a date in the Gregorian calendar to the number days since 31st December 1BC.

PARAMETER **date** The date (using the Gregorian calendar)

RETURN The number of elapsed days (1 Jan 1AD = 1)

FUNCTION ToGregorianDate

Date \

Date_t	ToGregorianDate
(Days_t days)	

Converts the number days since 31st December 1BC to a date in the Gregorian calendar.

PARAMETER **days** The number of elapsed days (1 Jan 1AD = 1)

RETURN A Date_t in the Gregorian calendar

FUNCTION DayOfYear

Date \

UNSIGNED2	DayOfYear
(Date_t date)	

Returns a number representing the day of the year indicated by the given date. The date must be in the Gregorian calendar after the year 1600.

PARAMETER **date** A Date_t value.

RETURN A number (1-366) representing the number of days since the beginning of the year.

FUNCTION DayOfWeek

Date \

UNSIGNED1	DayOfWeek
(Date_t date)	

Returns a number representing the day of the week indicated by the given date. The date must be in the Gregorian calendar after the year 1600.

PARAMETER **date** A Date_t value.

RETURN A number 1-7 representing the day of the week, where 1 = Sunday.

FUNCTION IsJulianLeapYear

Date \

BOOLEAN	**IsJulianLeapYear**
(INTEGER2 year)	

Tests whether the year is a leap year in the Julian calendar.

PARAMETER **year** The year (0-9999).

RETURN True if the year is a leap year.

FUNCTION FromJulianYMD

Date \

Days_t	**FromJulianYMD**
(INTEGER2 year, UNSIGNED1 month, UNSIGNED1 day)	

Combines year, month, day in the Julian calendar to create the number days since 31st December 1BC.

PARAMETER **year** The year (-4800..9999).

PARAMETER **month** The month (1-12).

PARAMETER **day** The day (1..daysInMonth).

RETURN The number of elapsed days (1 Jan 1AD = 1)

MODULE ToJulianYMD

Date \

	ToJulianYMD
(Days_t days)	

Converts the number days since 31st December 1BC to a date in the Julian calendar.

PARAMETER **days** The number of elapsed days (1 Jan 1AD = 1)

RETURN Module containing Year, Month, Day in the Julian calendar

Children

1. Day
2. Month
3. Year

ATTRIBUTE Day

Date \ ToJulianYMD \

UNSIGNED1	Day

ATTRIBUTE Month

Date \ ToJulianYMD \

UNSIGNED1	Month

ATTRIBUTE Year

Date \ ToJulianYMD \

INTEGER2	**Year**

FUNCTION FromJulianDate

Date \

Days_t	**FromJulianDate**
(Date_t date)	

Converts a date in the Julian calendar to the number days since 31st December 1BC.

PARAMETER **date** The date (using the Julian calendar)

RETURN The number of elapsed days (1 Jan 1AD = 1)

FUNCTION ToJulianDate

Date \

Date_t	**ToJulianDate**
(Days_t days)	

Converts the number days since 31st December 1BC to a date in the Julian calendar.

PARAMETER **days** The number of elapsed days (1 Jan 1AD = 1)

RETURN A Date_t in the Julian calendar

FUNCTION DaysSince1900

Date \

Days_t	DaysSince1900
(INTEGER2 year, UNSIGNED1 month, UNSIGNED1 day)	

Returns the number of days since 1st January 1900 (using the Gregorian Calendar)

PARAMETER **year** The year (-4713..9999).

PARAMETER **month** The month (1-12). A missing value (0) is treated as 1.

PARAMETER **day** The day (1..daysInMonth). A missing value (0) is treated as 1.

RETURN The number of elapsed days since 1st January 1900

FUNCTION ToDaysSince1900

Date \

Days_t	ToDaysSince1900
(Date_t date)	

Returns the number of days since 1st January 1900 (using the Gregorian Calendar)

PARAMETER **date** The date

RETURN The number of elapsed days since 1st January 1900

FUNCTION FromDaysSince1900

Date \

Date_t	FromDaysSince1900
(Days_t days)	

Converts the number days since 1st January 1900 to a date in the Julian calendar.

PARAMETER **days** The number of elapsed days since 1st Jan 1900

RETURN A Date_t in the Julian calendar

FUNCTION YearsBetween

Date \

INTEGER	**YearsBetween**
`(Date_t from, Date_t to)`	

Calculate the number of whole years between two dates.

PARAMETER **from** The first date

PARAMETER **to** The last date

RETURN The number of years between them.

FUNCTION MonthsBetween

Date \

INTEGER	**MonthsBetween**
`(Date_t from, Date_t to)`	

Calculate the number of whole months between two dates.

PARAMETER **from** The first date

PARAMETER **to** The last date

RETURN The number of months between them.

FUNCTION DaysBetween

Date \

INTEGER	**DaysBetween**
(Date_t from, Date_t to)	

Calculate the number of days between two dates.

PARAMETER **from** The first date

PARAMETER **to** The last date

RETURN The number of days between them.

FUNCTION DateFromDateRec

Date \

Date_t	**DateFromDateRec**
(Date_rec date)	

Combines the fields from a Date_rec to create a Date_t

PARAMETER **date** The row containing the date.

RETURN A Date_t representing the combined values.

FUNCTION DateFromRec

Date \

Date_t	**DateFromRec**
(Date_rec date)	

Combines the fields from a Date_rec to create a Date_t

PARAMETER **date** The row containing the date.

RETURN A Date_t representing the combined values.

FUNCTION TimeFromTimeRec

Date \

Time_t	**TimeFromTimeRec**
(Time_rec time)	

Combines the fields from a Time_rec to create a Time_t

PARAMETER **time** The row containing the time.

RETURN A Time_t representing the combined values.

FUNCTION DateFromDateTimeRec

Date \

Date_t	**DateFromDateTimeRec**
(DateTime_rec datetime)	

Combines the date fields from a DateTime_rec to create a Date_t

PARAMETER **datetime** The row containing the datetime.

RETURN A Date_t representing the combined values.

FUNCTION TimeFromDateTimeRec

Date \

Time_t	TimeFromDateTimeRec
(DateTime_rec datetime)	

Combines the time fields from a DateTime_rec to create a Time_t

PARAMETER **datetime** The row containing the datetime.

RETURN A Time_t representing the combined values.

FUNCTION SecondsFromDateTimeRec

Date \

Seconds_t	SecondsFromDateTimeRec
(DateTime_rec datetime, BOOLEAN is_local_time = FALSE)	

Combines the date and time fields from a DateTime_rec to create a Seconds_t

PARAMETER **datetime** The row containing the datetime.

PARAMETER **is_local_time** TRUE if the datetime components are expressed in local time rather than UTC, FALSE if the components are expressed in UTC. Optional, defaults to FALSE.

RETURN A Seconds_t representing the combined values.

FUNCTION FromStringToDate

Date \

Date_t	FromStringToDate
(STRING date_text, VARSTRING format)	

Converts a string to a Date_t using the relevant string format. The resulting date must be representable within the Gregorian calendar after the year 1600.

PARAMETER **date_text** The string to be converted.

PARAMETER **format** The format of the input string. (See documentation for strftime)

RETURN The date that was matched in the string. Returns 0 if failed to match or if the date components match but the result is an invalid date. Supported characters: %B Full month name %b or %h Abbreviated month name %d Day of month (two digits) %e Day of month (two digits, or a space followed by a single digit) %m Month (two digits) %t Whitespace %y year within century (00-99) %Y Full year (yyyy) %j Julian day (1-366) Common date formats American '%m/%d/%Y' mm/dd/yyyy Euro '%d/%m/%Y' dd/mm/yyyy Iso format '%Y-%m-%d' yyyy-mm-dd Iso basic 'Y%m%d' yyyymmdd '%d-%b-%Y' dd-mon-yyyy e.g., '21-Mar-1954'

FUNCTION FromString

Date \

Date_t	FromString
(STRING date_text, VARSTRING format)	

Converts a string to a date using the relevant string format.

PARAMETER **date_text** The string to be converted.

PARAMETER **format** The format of the input string. (See documentation for strftime)

RETURN The date that was matched in the string. Returns 0 if failed to match.

FUNCTION FromStringToTime

Date \

Time_t	FromStringToTime
(STRING time_text, VARSTRING format)	

Converts a string to a Time_t using the relevant string format.

PARAMETER **date_text** The string to be converted.

PARAMETER **format** The format of the input string. (See documentation for strftime)

RETURN The time that was matched in the string. Returns 0 if failed to match. Supported characters: %H Hour (two digits) %k (two digits, or a space followed by a single digit) %M Minute (two digits) %S Second (two digits) %t Whitespace

---

FUNCTION MatchDateString

Date \

Date_t	MatchDateString
(STRING date_text, SET OF VARSTRING formats)	

Matches a string against a set of date string formats and returns a valid Date_t object from the first format that successfully parses the string.

PARAMETER **date_text** The string to be converted.

PARAMETER **formats** A set of formats to check against the string. (See documentation for strftime)

RETURN The date that was matched in the string. Returns 0 if failed to match.

FUNCTION MatchTimeString

Date \

Time_t	MatchTimeString
(STRING time_text, SET OF VARSTRING formats)	

Matches a string against a set of time string formats and returns a valid Time_t object from the first format that successfully parses the string.

PARAMETER **time_text** The string to be converted.

PARAMETER **formats** A set of formats to check against the string. (See documentation for strftime)

RETURN The time that was matched in the string. Returns 0 if failed to match.

FUNCTION DateToString

Date \

STRING	DateToString
(Date_t date, VARSTRING format = '%Y-%m-%d')	

Formats a date as a string.

PARAMETER **date** The date to be converted.

PARAMETER **format** The format template to use for the conversion; see strftime() for appropriate values. The maximum length of the resulting string is 255 characters. Optional; defaults to '%Y-%m-%d' which is YYYY-MM-DD.

RETURN Blank if date cannot be formatted, or the date in the requested format.

FUNCTION TimeToString

Date \

STRING	TimeToString
(Time_t time, VARSTRING format = '%H:%M:%S')	

Formats a time as a string.

PARAMETER **time** The time to be converted.

PARAMETER **format** The format template to use for the conversion; see strftime() for appropriate values. The maximum length of the resulting string is 255 characters. Optional; defaults to '%H:%M:%S' which is HH:MM:SS.

RETURN Blank if the time cannot be formatted, or the time in the requested format.

FUNCTION SecondsToString

Date \

STRING	SecondsToString
(Seconds_t seconds, VARSTRING format = '%Y-%m-%dT%H:%M:%S')	

Converts a Seconds_t value into a human-readable string using a format template.

PARAMETER **seconds** The seconds since epoch.

PARAMETER **format** The format template to use for the conversion; see strftime() for appropriate values. The maximum length of the resulting string is 255 characters. Optional; defaults to '%Y-%m-%dT%H:%M:%S' which is YYYY-MM-DDTHH:MM:SS.

RETURN The converted seconds as a string.

FUNCTION ToString

Date \

STRING	ToString
(Date_t date, VARSTRING format)	

Formats a date as a string.

PARAMETER **date** The date to be converted.

PARAMETER **format** The format the date is output in. (See documentation for strftime)

RETURN Blank if date cannot be formatted, or the date in the requested format.

FUNCTION ConvertDateFormat

Date \

STRING	ConvertDateFormat
(STRING date_text, VARSTRING from_format='%m/%d/%Y', VARSTRING to_format='%Y%m%d')	

Converts a date from one format to another

PARAMETER **date_text** The string containing the date to be converted.

PARAMETER **from_format** The format the date is to be converted from.

PARAMETER **to_format** The format the date is to be converted to.

RETURN The converted string, or blank if it failed to match the format.

FUNCTION ConvertFormat

Date \

STRING	ConvertFormat
(STRING date_text, VARSTRING from_format='%m/%d/%Y', VARSTRING to_format='%Y%m%d')	

Converts a date from one format to another

PARAMETER **date_text** The string containing the date to be converted.

PARAMETER **from_format** The format the date is to be converted from.

PARAMETER **to_format** The format the date is to be converted to.

RETURN The converted string, or blank if it failed to match the format.

FUNCTION ConvertTimeFormat

Date \

STRING	ConvertTimeFormat
(STRING time_text, VARSTRING from_format='%H%M%S', VARSTRING to_format='%H:%M:%S')	

Converts a time from one format to another

PARAMETER **time_text** The string containing the time to be converted.

PARAMETER **from_format** The format the time is to be converted from.

PARAMETER **to_format** The format the time is to be converted to.

RETURN The converted string, or blank if it failed to match the format.

FUNCTION ConvertDateFormatMultiple

Date \

STRING	ConvertDateFormatMultiple
(STRING date_text, SET OF VARSTRING from_formats, VARSTRING to_format='%Y%m%d')	

Converts a date that matches one of a set of formats to another.

PARAMETER **date_text** The string containing the date to be converted.

PARAMETER **from_formats** The list of formats the date is to be converted from.

PARAMETER **to_format** The format the date is to be converted to.

RETURN The converted string, or blank if it failed to match the format.

FUNCTION ConvertFormatMultiple

Date \

STRING	ConvertFormatMultiple
(STRING date_text, SET OF VARSTRING from_formats, VARSTRING to_format='%Y%m%d')	

Converts a date that matches one of a set of formats to another.

PARAMETER **date_text** The string containing the date to be converted.

PARAMETER **from_formats** The list of formats the date is to be converted from.

PARAMETER **to_format** The format the date is to be converted to.

RETURN The converted string, or blank if it failed to match the format.

FUNCTION ConvertTimeFormatMultiple

Date \

STRING	ConvertTimeFormatMultiple
(STRING time_text, SET OF VARSTRING from_formats, VARSTRING to_format='%H:%m:%s')	

Converts a time that matches one of a set of formats to another.

PARAMETER **time_text** The string containing the time to be converted.

PARAMETER **from_formats** The list of formats the time is to be converted from.

PARAMETER **to_format** The format the time is to be converted to.

RETURN The converted string, or blank if it failed to match the format.

FUNCTION AdjustDate

Date \

Date_t	AdjustDate
(Date_t date, INTEGER2 year_delta = 0, INTEGER4 month_delta = 0, INTEGER4 day_delta = 0)	

Adjusts a date by incrementing or decrementing year, month and/or day values. The date must be in the Gregorian calendar after the year 1600. If the new calculated date is invalid then it will be normalized according to mktime() rules. Example: 20140130 + 1 month = 20140302.

PARAMETER **date** The date to adjust.

PARAMETER **year_delta** The requested change to the year value; optional, defaults to zero.

PARAMETER **month_delta** The requested change to the month value; optional, defaults to zero.

PARAMETER **day_delta** The requested change to the day of month value; optional, defaults to zero.

RETURN The adjusted Date_t value.

FUNCTION AdjustDateBySeconds

Date \

Date_t	AdjustDateBySeconds
(Date_t date, INTEGER4 seconds_delta)	

Adjusts a date by adding or subtracting seconds. The date must be in the Gregorian calendar after the year 1600. If the new calculated date is invalid then it will be normalized according to mktime() rules. Example: 20140130 + 172800 seconds = 20140201.

PARAMETER **date** The date to adjust.

PARAMETER **seconds_delta** The requested change to the date, in seconds.

RETURN The adjusted Date_t value.

FUNCTION AdjustTime

Date \

Time_t	AdjustTime
(Time_t time, INTEGER2 hour_delta = 0, INTEGER4 minute_delta = 0, INTEGER4 second_delta = 0)	

Adjusts a time by incrementing or decrementing hour, minute and/or second values. If the new calculated time is invalid then it will be normalized according to mktime() rules.

PARAMETER **time** The time to adjust.

PARAMETER **hour_delta** The requested change to the hour value; optional, defaults to zero.

PARAMETER **minute_delta** The requested change to the minute value; optional, defaults to zero.

PARAMETER **second_delta** The requested change to the second of month value; optional, defaults to zero.

RETURN The adjusted Time_t value.

FUNCTION AdjustTimeBySeconds

Date \

Time_t	AdjustTimeBySeconds
(Time_t time, INTEGER4 seconds_delta)	

Adjusts a time by adding or subtracting seconds. If the new calculated time is invalid then it will be normalized according to mktime() rules.

PARAMETER **time** The time to adjust.

PARAMETER **seconds_delta** The requested change to the time, in seconds.

RETURN The adjusted Time_t value.

FUNCTION AdjustSeconds

Date \

Seconds_t	AdjustSeconds
(Seconds_t seconds, INTEGER2 year_delta = 0, INTEGER4 month_delta = 0, INTEGER4 day_delta = 0, INTEGER4 hour_delta = 0, INTEGER4 minute_delta = 0, INTEGER4 second_delta = 0)	

Adjusts a Seconds_t value by adding or subtracting years, months, days, hours, minutes and/or seconds. This is performed by first converting the seconds into a full date/time structure, applying any delta values to individual date/time components, then converting the structure back to the number of seconds. This interim date must lie within Gregorian calendar after the year 1600. If the interim structure is found to have an invalid date/time then it will be normalized according to mktime() rules. Therefore, some delta values (such as "1 month") are actually relative to the value of the seconds argument.

PARAMETER **seconds** The number of seconds to adjust.

PARAMETER **year_delta** The requested change to the year value; optional, defaults to zero.

PARAMETER **month_delta** The requested change to the month value; optional, defaults to zero.

PARAMETER **day_delta** The requested change to the day of month value; optional, defaults to zero.

PARAMETER **hour_delta** The requested change to the hour value; optional, defaults to zero.

PARAMETER **minute_delta** The requested change to the minute value; optional, defaults to zero.

PARAMETER **second_delta** The requested change to the second of month value; optional, defaults to zero.

RETURN The adjusted Seconds_t value.

FUNCTION AdjustCalendar

Date \

Date_t	AdjustCalendar
(Date_t date, INTEGER2 year_delta = 0, INTEGER4 month_delta = 0, INTEGER4 day_delta = 0)	

Adjusts a date by incrementing or decrementing months and/or years. This routine uses the rule outlined in McGinn v. State, 46 Neb. 427, 65 N.W. 46 (1895): "The term calendar month, whether employed in

statutes or contracts, and not appearing to have been used in a different sense, denotes a period terminating with the day of the succeeding month numerically corresponding to the day of its beginning, less one. If there be no corresponding day of the succeeding month, it terminates with the last day thereof." The internet suggests similar legal positions exist in the Commonwealth and Germany. Note that day adjustments are performed after year and month adjustments using the preceding rules. As an example, Jan. 31, 2014 + 1 month will result in Feb. 28, 2014; Jan. 31, 2014 + 1 month + 1 day will result in Mar. 1, 2014.

PARAMETER **date** The date to adjust, in the Gregorian calendar after 1600.

PARAMETER **year_delta** The requested change to the year value; optional, defaults to zero.

PARAMETER **month_delta** The requested change to the month value; optional, defaults to zero.

PARAMETER **day_delta** The requested change to the day value; optional, defaults to zero.

RETURN The adjusted Date_t value.

FUNCTION IsLocalDaylightSavingsInEffect

Date \

BOOLEAN	**IsLocalDaylightSavingsInEffect**
()	

Returns a boolean indicating whether daylight savings time is currently in effect locally.

RETURN TRUE if daylight savings time is currently in effect, FALSE otherwise.

FUNCTION LocalTimeZoneOffset

Date \

INTEGER4	**LocalTimeZoneOffset**
()	

Returns the offset (in seconds) of the time represented from UTC, with positive values indicating locations east of the Prime Meridian. Given a UTC time in seconds since epoch, you can find the local time by adding the result of this function to the seconds.

RETURN The number of seconds offset from UTC.

FUNCTION CurrentDate

Date \

Date_t	**CurrentDate**
(BOOLEAN in_local_time = FALSE)	

Returns the current date.

PARAMETER **in_local_time** TRUE if the returned value should be local to the cluster computing the date, FALSE for UTC. Optional, defaults to FALSE.

RETURN A Date_t representing the current date.

FUNCTION Today

Date \

Date_t	**Today**
()	

Returns the current date in the local time zone.

RETURN A Date_t representing the current date.

FUNCTION CurrentTime

Date \

Time_t	CurrentTime
(BOOLEAN in_local_time = FALSE)	

Returns the current time of day

PARAMETER **in_local_time** TRUE if the returned value should be local to the cluster computing the time, FALSE for UTC. Optional, defaults to FALSE.

RETURN A Time_t representing the current time of day.

FUNCTION CurrentSeconds

Date \

Seconds_t	CurrentSeconds
(BOOLEAN in_local_time = FALSE)	

Returns the current date and time as the number of seconds since epoch.

PARAMETER **in_local_time** TRUE if the returned value should be local to the cluster computing the time, FALSE for UTC. Optional, defaults to FALSE.

RETURN A Seconds_t representing the current time in UTC or local time, depending on the argument.

FUNCTION CurrentTimestamp

Date \

Timestamp_t	CurrentTimestamp
(BOOLEAN in_local_time = FALSE)	

Returns the current date and time as the number of microseconds since epoch.

PARAMETER **in_local_time** TRUE if the returned value should be local to the cluster computing the time, FALSE for UTC. Optional, defaults to FALSE.

RETURN A Timestamp_t representing the current time in microseconds in UTC or local time, depending on the argument.

MODULE DatesForMonth

Date \

	DatesForMonth
(Date_t as_of_date = CurrentDate(FALSE))	

Returns the beginning and ending dates for the month surrounding the given date.

PARAMETER **as_of_date** The reference date from which the month will be calculated. This date must be a date within the Gregorian calendar. Optional, defaults to the current date in UTC.

RETURN Module with exported attributes for startDate and endDate.

Children

1. startDate
2. endDate

ATTRIBUTE startDate

Date \ DatesForMonth \

Date_t	startDate

ATTRIBUTE endDate

Date \ DatesForMonth \

Date_t	endDate

MODULE DatesForWeek

Date \

	DatesForWeek
(Date_t as_of_date = CurrentDate(FALSE))	

Returns the beginning and ending dates for the week surrounding the given date (Sunday marks the beginning of a week).

PARAMETER **as_of_date** The reference date from which the week will be calculated. This date must be a date within the Gregorian calendar. Optional, defaults to the current date in UTC.

RETURN Module with exported attributes for startDate and endDate.

Children

1. startDate
2. endDate

ATTRIBUTE startDate

Date \ DatesForWeek \

Date_t	startDate

ATTRIBUTE endDate

Date \ DatesForWeek \

Date_t	endDate

FUNCTION IsValidDate

Date \

BOOLEAN	IsValidDate
(Date_t date, INTEGER2 yearLowerBound = 1800, INTEGER2 yearUpperBound = 2100)	

Tests whether a date is valid, both by range-checking the year and by validating each of the other individual components.

PARAMETER **date** The date to validate.

PARAMETER **yearLowerBound** The minimum acceptable year. Optional; defaults to 1800.

PARAMETER **yearUpperBound** The maximum acceptable year. Optional; defaults to 2100.

RETURN TRUE if the date is valid, FALSE otherwise.

FUNCTION IsValidGregorianDate

Date \

BOOLEAN	IsValidGregorianDate
(Date_t date)	

Tests whether a date is valid in the Gregorian calendar. The year must be between 1601 and 30827.

PARAMETER **date** The Date_t to validate.

RETURN TRUE if the date is valid, FALSE otherwise.

FUNCTION IsValidTime

Date \

BOOLEAN	**IsValidTime**
(Time_t time)	

Tests whether a time is valid.

PARAMETER **time** The time to validate.

RETURN TRUE if the time is valid, FALSE otherwise.

TRANSFORM CreateDate

Date \

Date_rec	**CreateDate**
(INTEGER2 year, UNSIGNED1 month, UNSIGNED1 day)	

A transform to create a Date_rec from the individual elements

PARAMETER **year** The year

PARAMETER **month** The month (1-12).

PARAMETER **day** The day (1..daysInMonth).

RETURN A transform that creates a Date_rec containing the date.

TRANSFORM CreateDateFromSeconds

Date \

Date_rec	CreateDateFromSeconds
(Seconds_t seconds)	

A transform to create a Date_rec from a Seconds_t value.

PARAMETER **seconds** The number seconds since epoch.

RETURN A transform that creates a Date_rec containing the date.

TRANSFORM CreateTime

Date \

Time_rec	CreateTime
(UNSIGNED1 hour, UNSIGNED1 minute, UNSIGNED1 second)	

A transform to create a Time_rec from the individual elements

PARAMETER **hour** The hour (0-23).

PARAMETER **minute** The minute (0-59).

PARAMETER **second** The second (0-59).

RETURN A transform that creates a Time_rec containing the time of day.

TRANSFORM CreateTimeFromSeconds

Date \

Time_rec	CreateTimeFromSeconds
(Seconds_t seconds)	

A transform to create a Time_rec from a Seconds_t value.

PARAMETER **seconds** The number seconds since epoch.

RETURN A transform that creates a Time_rec containing the time of day.

TRANSFORM CreateDateTime

Date \

DateTime_rec	**CreateDateTime**
(INTEGER2 year, UNSIGNED1 month, UNSIGNED1 day, UNSIGNED1 hour, UNSIGNED1 minute, UNSIGNED1 second)	

A transform to create a DateTime_rec from the individual elements

PARAMETER **year** The year

PARAMETER **month** The month (1-12).

PARAMETER **day** The day (1..daysInMonth).

PARAMETER **hour** The hour (0-23).

PARAMETER **minute** The minute (0-59).

PARAMETER **second** The second (0-59).

RETURN A transform that creates a DateTime_rec containing date and time components.

TRANSFORM CreateDateTimeFromSeconds

Date \

DateTime_rec	**CreateDateTimeFromSeconds**
(Seconds_t seconds)	

A transform to create a DateTime_rec from a Seconds_t value.

PARAMETER **seconds** The number seconds since epoch.

RETURN A transform that creates a DateTime_rec containing date and time components.

File

Go Up

IMPORTS

lib_fileservices |

DESCRIPTIONS

MODULE File

	File

Children

1. FsFilenameRecord : A record containing information about filename
2. FsLogicalFileName : An alias for a logical filename that is stored in a row
3. FsLogicalFileNameRecord : A record containing a logical filename
4. FsLogicalFileInfoRecord : A record containing information about a logical file
5. FsLogicalSuperSubRecord : A record containing information about a superfile and its contents
6. FsFileRelationshipRecord : A record containing information about the relationship between two files
7. RECFMV_RECSIZE : Constant that indicates IBM RECFM V format file
8. RECFMVB_RECSIZE : Constant that indicates IBM RECFM VB format file
9. PREFIX_VARIABLE_RECSIZE : Constant that indicates a variable little endian 4 byte length prefixed file

10. PREFIX_VARIABLE_BIGENDIAN_RECSIZE : Constant that indicates a variable big endian 4 byte length prefixed file
11. FileExists : Returns whether the file exists
12. DeleteLogicalFile : Removes the logical file from the system, and deletes from the disk
13. SetReadOnly : Changes whether access to a file is read only or not
14. RenameLogicalFile : Changes the name of a logical file
15. ForeignLogicalFileName : Returns a logical filename that can be used to refer to a logical file in a local or remote dali
16. ExternalLogicalFileName : Returns an encoded logical filename that can be used to refer to a external file
17. GetFileDescription : Returns a string containing the description information associated with the specified filename
18. SetFileDescription : Sets the description associated with the specified filename
19. RemoteDirectory : Returns a dataset containing a list of files from the specified machineIP and directory
20. LogicalFileList : Returns a dataset of information about the logical files known to the system
21. CompareFiles : Compares two files, and returns a result indicating how well they match
22. VerifyFile : Checks the system datastore (Dali) information for the file against the physical parts on disk
23. AddFileRelationship : Defines the relationship between two files
24. FileRelationshipList : Returns a dataset of relationships
25. RemoveFileRelationship : Removes a relationship between two files
26. GetColumnMapping : Returns the field mappings for the file, in the same format specified for the SetColumnMapping function
27. SetColumnMapping : Defines how the data in the fields of the file mist be transformed between the actual data storage format and the input format used to query that data
28. EncodeRfsQuery : Returns a string that can be used in a DATASET declaration to read data from an RFS (Remote File Server) instance (e.g
29. RfsAction : Sends the query to the rfs server
30. MoveExternalFile : Moves the single physical file between two locations on the same remote machine
31. DeleteExternalFile : Removes a single physical file from a remote machine
32. CreateExternalDirectory : Creates the path on the location (if it does not already exist)
33. GetLogicalFileAttribute : Returns the value of the given attribute for the specified logicalfilename

34. ProtectLogicalFile : Toggles protection on and off for the specified logicalfilename
35. DfuPlusExec : The DfuPlusExec action executes the specified command line just as the DfuPLus.exe program would do
36. fSprayFixed : Sprays a file of fixed length records from a single machine and distributes it across the nodes of the destination group
37. SprayFixed : Same as fSprayFixed, but does not return the DFU Workunit ID
38. fSprayVariable
39. SprayVariable
40. fSprayDelimited : Sprays a file of fixed delimited records from a single machine and distributes it across the nodes of the destination group
41. SprayDelimited : Same as fSprayDelimited, but does not return the DFU Workunit ID
42. fSprayXml : Sprays an xml file from a single machine and distributes it across the nodes of the destination group
43. SprayXml : Same as fSprayXml, but does not return the DFU Workunit ID
44. fDespray : Copies a distributed file from multiple machines, and desprays it to a single file on a single machine
45. Despray : Same as fDespray, but does not return the DFU Workunit ID
46. fCopy : Copies a distributed file to another distributed file
47. Copy : Same as fCopy, but does not return the DFU Workunit ID
48. fReplicate : Ensures the specified file is replicated to its mirror copies
49. Replicate : Same as fReplicated, but does not return the DFU Workunit ID
50. fRemotePull : Copies a distributed file to a distributed file on remote system
51. RemotePull : Same as fRemotePull, but does not return the DFU Workunit ID
52. fMonitorLogicalFileName : Creates a file monitor job in the DFU Server
53. MonitorLogicalFileName : Same as fMonitorLogicalFileName, but does not return the DFU Workunit ID
54. fMonitorFile : Creates a file monitor job in the DFU Server
55. MonitorFile : Same as fMonitorFile, but does not return the DFU Workunit ID
56. WaitDfuWorkunit : Waits for the specified DFU workunit to finish
57. AbortDfuWorkunit : Aborts the specified DFU workunit
58. CreateSuperFile : Creates an empty superfile
59. SuperFileExists : Checks if the specified filename is present in the Distributed File Utility (DFU) and is a SuperFile

60. DeleteSuperFile : Deletes the superfile
61. GetSuperFileSubCount : Returns the number of sub-files contained within a superfile
62. GetSuperFileSubName : Returns the name of the Nth sub-file within a superfile
63. FindSuperFileSubName : Returns the position of a file within a superfile
64. StartSuperFileTransaction : Starts a superfile transaction
65. AddSuperFile : Adds a file to a superfile
66. RemoveSuperFile : Removes a sub-file from a superfile
67. ClearSuperFile : Removes all sub-files from a superfile
68. RemoveOwnedSubFiles : Removes all soley-owned sub-files from a superfile
69. DeleteOwnedSubFiles : Legacy version of RemoveOwnedSubFiles which was incorrectly named in a previous version
70. SwapSuperFile : Swap the contents of two superfiles
71. ReplaceSuperFile : Removes a sub-file from a superfile and replaces it with another
72. FinishSuperFileTransaction : Finishes a superfile transaction
73. SuperFileContents : Returns the list of sub-files contained within a superfile
74. LogicalFileSuperOwners : Returns the list of superfiles that a logical file is contained within
75. LogicalFileSuperSubList : Returns the list of all the superfiles in the system and their component sub-files
76. fPromoteSuperFileList : Moves the sub-files from the first entry in the list of superfiles to the next in the list, repeating the process through the list of superfiles
77. PromoteSuperFileList : Same as fPromoteSuperFileList, but does not return the DFU Workunit ID

RECORD FsFilenameRecord

File \

	FsFilenameRecord

A record containing information about filename. Includes name, size and when last modified. export FsFilenameRecord := RECORD string name; integer8 size; string19 modified; END;

ATTRIBUTE FsLogicalFileName

File \

	FsLogicalFileName

An alias for a logical filename that is stored in a row.

RECORD FsLogicalFileNameRecord

File \

	FsLogicalFileNameRecord

A record containing a logical filename. It contains the following fields:

FIELD **name** The logical name of the file;

RECORD FsLogicalFileInfoRecord

File \

	FsLogicalFileInfoRecord

A record containing information about a logical file.

FIELD **superfile** Is this a superfile?

FIELD **size** Number of bytes in the file (before compression)

FIELD **rowcount** Number of rows in the file.

RECORD FsLogicalSuperSubRecord

File \

	FsLogicalSuperSubRecord

A record containing information about a superfile and its contents.

FIELD **supername** The name of the superfile

FIELD **subname** The name of the sub-file

RECORD FsFileRelationshipRecord

File \

	FsFileRelationshipRecord

A record containing information about the relationship between two files.

FIELD **primaryfile** The logical filename of the primary file

FIELD **secondaryfile** The logical filename of the secondary file.

FIELD **primaryflds** The name of the primary key field for the primary file. The value "___fileposition___" indicates the secondary is an INDEX that must use FETCH to access non-keyed fields.

FIELD **secondaryflds** The name of the foreign key field relating to the primary file.

FIELD **kind** The type of relationship between the primary and secondary files. Containing either 'link' or 'view'.

FIELD **cardinality** The cardinality of the relationship. The format is <primary>:<secondary>. Valid values are "1" or "M".</secondary></primary>

FIELD **payload** Indicates whether the primary or secondary are payload INDEXes.

FIELD **description** The description of the relationship.

ATTRIBUTE RECFMV_RECSIZE

File \

	RECFMV_RECSIZE

Constant that indicates IBM RECFM V format file. Can be passed to SprayFixed for the record size.

ATTRIBUTE RECFMVB_RECSIZE

File \

	RECFMVB_RECSIZE

Constant that indicates IBM RECFM VB format file. Can be passed to SprayFixed for the record size.

ATTRIBUTE PREFIX_VARIABLE_RECSIZE

File \

INTEGER4	**PREFIX_VARIABLE_RECSIZE**

Constant that indicates a variable little endian 4 byte length prefixed file. Can be passed to SprayFixed for the record size.

ATTRIBUTE PREFIX_VARIABLE_BIGENDIAN_RECSIZE

File \

INTEGER4	**PREFIX_VARIABLE_BIGENDIAN_RECSIZE**

Constant that indicates a variable big endian 4 byte length prefixed file. Can be passed to SprayFixed for

the record size.

FUNCTION FileExists

File \

boolean	**FileExists**
(varstring lfn, boolean physical=FALSE)	

Returns whether the file exists.

PARAMETER **lfn** The logical name of the file.

PARAMETER **physical** Whether to also check for the physical existence on disk. Defaults to FALSE.

RETURN Whether the file exists.

FUNCTION DeleteLogicalFile

File \

	DeleteLogicalFile
(varstring lfn, boolean allowMissing=FALSE)	

Removes the logical file from the system, and deletes from the disk.

PARAMETER **lfn** The logical name of the file.

PARAMETER **allowMissing** Whether to suppress an error if the filename does not exist. Defaults to FALSE.

FUNCTION SetReadOnly

File \

	SetReadOnly
(varstring lfn, boolean ro=TRUE)	

Changes whether access to a file is read only or not.

PARAMETER **lfn** The logical name of the file.

PARAMETER **ro** Whether updates to the file are disallowed. Defaults to TRUE.

FUNCTION RenameLogicalFile

File \

	RenameLogicalFile
(varstring oldname, varstring newname)	

Changes the name of a logical file.

PARAMETER **oldname** The current name of the file to be renamed.

PARAMETER **newname** The new logical name of the file.

FUNCTION ForeignLogicalFileName

File \

varstring	ForeignLogicalFileName
(varstring name, varstring foreigndali=", boolean abspath=FALSE)	

Returns a logical filename that can be used to refer to a logical file in a local or remote dali.

PARAMETER **name** The logical name of the file.

PARAMETER **foreigndali** The IP address of the foreign dali used to resolve the file. If blank then the file is resolved locally. Defaults to blank.

PARAMETER **abspath** Should a tilde (~) be prepended to the resulting logical file name. Defaults to FALSE.

FUNCTION ExternalLogicalFileName

File \

varstring	ExternalLogicalFileName
(varstring location, varstring path, boolean abspath=TRUE)	

Returns an encoded logical filename that can be used to refer to a external file. Examples include directly reading from a landing zone. Upper case characters and other details are escaped.

PARAMETER **location** The IP address of the remote machine. '.' can be used for the local machine.

PARAMETER **path** The path/name of the file on the remote machine.

PARAMETER **abspath** Should a tilde (~) be prepended to the resulting logical file name. Defaults to TRUE.

RETURN The encoded logical filename.

FUNCTION GetFileDescription

File \

varstring	GetFileDescription
(varstring lfn)	

Returns a string containing the description information associated with the specified filename. This description is set either through ECL watch or by using the FileServices.SetFileDescription function.

PARAMETER **lfn** The logical name of the file.

FUNCTION SetFileDescription

File \

	SetFileDescription
(varstring lfn, varstring val)	

Sets the description associated with the specified filename.

PARAMETER **lfn** The logical name of the file.

PARAMETER **val** The description to be associated with the file.

FUNCTION RemoteDirectory

File \

dataset(FsFilenameRecord)	RemoteDirectory
(varstring machineIP, varstring dir, varstring mask='*', boolean recurse=FALSE)	

Returns a dataset containing a list of files from the specified machineIP and directory.

PARAMETER **machineIP** The IP address of the remote machine.

PARAMETER **directory** The path to the directory to read. This must be in the appropriate format for the operating system running on the remote machine.

PARAMETER **mask** The filemask specifying which files to include in the result. Defaults to '*' (all files).

PARAMETER **recurse** Whether to include files from subdirectories under the directory. Defaults to FALSE.

FUNCTION LogicalFileList

File \

dataset(FsLogicalFileInfoRecord)	**LogicalFileList**
(varstring namepattern='*', boolean includenormal=TRUE, boolean includesuper=FALSE, boolean unknownszero=FALSE, varstring foreigndali='')	

Returns a dataset of information about the logical files known to the system.

PARAMETER **namepattern** The mask of the files to list. Defaults to '*' (all files).

PARAMETER **includenormal** Whether to include 'normal' files. Defaults to TRUE.

PARAMETER **includesuper** Whether to include SuperFiles. Defaults to FALSE.

PARAMETER **unknownszero** Whether to set file sizes that are unknown to zero(0) instead of minus-one (-1). Defaults to FALSE.

PARAMETER **foreigndali** The IP address of the foreign dali used to resolve the file. If blank then the file is resolved locally. Defaults to blank.

FUNCTION CompareFiles

File \

INTEGER4	**CompareFiles**
(varstring lfn1, varstring lfn2, boolean logical_only=TRUE, boolean use_crcs=FALSE)	

Compares two files, and returns a result indicating how well they match.

PARAMETER **file1** The logical name of the first file.

PARAMETER **file2** The logical name of the second file.

PARAMETER **logical_only** Whether to only compare logical information in the system datastore (Dali), and ignore physical information on disk. [Default TRUE]

PARAMETER **use_crcs** Whether to compare physical CRCs of all the parts on disk. This may be slow on large files. Defaults to FALSE.

RETURN 0 if file1 and file2 match exactly 1 if file1 and file2 contents match, but file1 is newer than file2 -1 if file1 and file2 contents match, but file2 is newer than file1 2 if file1 and file2 contents do not match and file1 is newer than file2 -2 if file1 and file2 contents do not match and file2 is newer than file1

FUNCTION VerifyFile

File \

varstring	**VerifyFile**
`(varstring lfn, boolean usecrcs)`	

Checks the system datastore (Dali) information for the file against the physical parts on disk.

PARAMETER **lfn** The name of the file to check.

PARAMETER **use_crcs** Whether to compare physical CRCs of all the parts on disk. This may be slow on large files.

RETURN 'OK' - The file parts match the datastore information 'Could not find file: <filename>' - The logical filename was not found 'Could not find part file: <partname>' - The partname was not found 'Modified time differs for: <partname>' - The partname has a different timestamp 'File size differs for: <partname>' - The partname has a file size 'File CRC differs for: <partname>' - The partname has a different CRC</partname></partname></partname></partname></filename>

FUNCTION AddFileRelationship

File \

	AddFileRelationship
`(varstring primary, varstring secondary, varstring primaryflds, varstring secondaryflds, varstring kind='link', varstring cardinality, boolean payload, varstring description="")`	

Defines the relationship between two files. These may be DATASETs or INDEXes. Each record in the primary file should be uniquely defined by the primaryfields (ideally), preferably efficiently. This information is used by the roxie browser to link files together.

PARAMETER **primary** The logical filename of the primary file.

PARAMETER **secondary** The logical filename of the secondary file.

PARAMETER **primaryfields** The name of the primary key field for the primary file. The value "___fileposition___" indicates the secondary is an INDEX that must use FETCH to access non-keyed fields.

PARAMETER **secondaryfields** The name of the foreign key field relating to the primary file.

PARAMETER **relationship** The type of relationship between the primary and secondary files. Containing either 'link' or 'view'. Default is "link".

PARAMETER **cardinality** The cardinality of the relationship. The format is <primary>:<secondary>. Valid values are "1" or "M".</secondary></primary>

PARAMETER **payload** Indicates whether the primary or secondary are payload INDEXes.

PARAMETER **description** The description of the relationship.

FUNCTION FileRelationshipList

File \

dataset(FsFileRelationshipRecord)	**FileRelationshipList**
(varstring primary, varstring secondary, varstring primflds=", varstring secondaryflds=", varstring kind='link')	

Returns a dataset of relationships. The return records are structured in the FsFileRelationshipRecord format.

PARAMETER **primary** The logical filename of the primary file.

PARAMETER **secondary** The logical filename of the secondary file.

PARAMETER **primaryfields** The name of the primary key field for the primary file.

PARAMETER **secondaryfields** The name of the foreign key field relating to the primary file.

PARAMETER **relationship** The type of relationship between the primary and secondary files. Containing either 'link' or 'view'. Default is "link".

FUNCTION RemoveFileRelationship

File \

	RemoveFileRelationship
	`(varstring primary, varstring secondary, varstring primaryflds=", varstring secondaryflds=", varstring kind='link')`

Removes a relationship between two files.

PARAMETER **primary** The logical filename of the primary file.

PARAMETER **secondary** The logical filename of the secondary file.

PARAMETER **primaryfields** The name of the primary key field for the primary file.

PARAMETER **secondaryfields** The name of the foreign key field relating to the primary file.

PARAMETER **relationship** The type of relationship between the primary and secondary files. Containing either 'link' or 'view'. Default is "link".

FUNCTION GetColumnMapping

File \

varstring	GetColumnMapping
`(varstring lfn)`	

Returns the field mappings for the file, in the same format specified for the SetColumnMapping function.

PARAMETER **lfn** The logical filename of the primary file.

FUNCTION SetColumnMapping

File \

	SetColumnMapping
(varstring lfn, varstring mapping)	

Defines how the data in the fields of the file mist be transformed between the actual data storage format and the input format used to query that data. This is used by the user interface of the roxie browser.

PARAMETER **lfn** The logical filename of the primary file.

PARAMETER **mapping** A string containing a comma separated list of field mappings.

FUNCTION EncodeRfsQuery

File \

varstring	EncodeRfsQuery
(varstring server, varstring query)	

Returns a string that can be used in a DATASET declaration to read data from an RFS (Remote File Server) instance (e.g. rfsmysql) on another node.

PARAMETER **server** A string containing the ip:port address for the remote file server.

PARAMETER **query** The text of the query to send to the server

---

FUNCTION RfsAction

File \

	RfsAction
(varstring server, varstring query)	

Sends the query to the rfs server.

PARAMETER **server** A string containing the ip:port address for the remote file server.

PARAMETER **query** The text of the query to send to the server

---

FUNCTION MoveExternalFile

File \

	MoveExternalFile
(varstring location, varstring frompath, varstring topath)	

Moves the single physical file between two locations on the same remote machine. The dafileserv utility program must be running on the location machine.

PARAMETER **location** The IP address of the remote machine.

PARAMETER **frompath** The path/name of the file to move.

PARAMETER **topath** The path/name of the target file.

FUNCTION DeleteExternalFile

File \

	DeleteExternalFile
(varstring location, varstring path)	

Removes a single physical file from a remote machine. The dafileserv utility program must be running on the location machine.

PARAMETER **location** The IP address of the remote machine.

PARAMETER **path** The path/name of the file to remove.

FUNCTION CreateExternalDirectory

File \

	CreateExternalDirectory
(varstring location, varstring path)	

Creates the path on the location (if it does not already exist). The dafileserv utility program must be running on the location machine.

PARAMETER **location** The IP address of the remote machine.

PARAMETER **path** The path/name of the file to remove.

FUNCTION GetLogicalFileAttribute

File \

varstring	GetLogicalFileAttribute
(varstring lfn, varstring attrname)	

Returns the value of the given attribute for the specified logicalfilename.

PARAMETER **lfn** The name of the logical file.

PARAMETER **attrname** The name of the file attribute to return.

FUNCTION ProtectLogicalFile

File \

	ProtectLogicalFile
(varstring lfn, boolean value=TRUE)	

Toggles protection on and off for the specified logicalfilename.

PARAMETER **lfn** The name of the logical file.

PARAMETER **value** TRUE to enable protection, FALSE to disable.

FUNCTION DfuPlusExec

File \

	DfuPlusExec
(varstring cmdline)	

The DfuPlusExec action executes the specified command line just as the DfuPLus.exe program would do. This allows you to have all the functionality of the DfuPLus.exe program available within your ECL code. param cmdline The DFUPlus.exe command line to execute. The valid arguments are documented in the Client Tools manual, in the section describing the DfuPlus.exe program.

FUNCTION fSprayFixed

File \

varstring	fSprayFixed
(varstring sourceIP, varstring sourcePath, integer4 recordSize, varstring destinationGroup, varstring destinationLogicalName, integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1, boolean allowOverwrite=FALSE, boolean replicate=FALSE, boolean compress=FALSE, boolean failIfNoSourceFile=FALSE, integer4 expireDays=-1)	

Sprays a file of fixed length records from a single machine and distributes it across the nodes of the destination group.

PARAMETER **sourceIP** The IP address of the file.

PARAMETER **sourcePath** The path and name of the file.

PARAMETER **recordsize** The size (in bytes) of the records in the file.

PARAMETER **destinationGroup** The name of the group to distribute the file across.

PARAMETER **destinationLogicalName** The logical name of the file to create.

PARAMETER **timeOut** The time in ms to wait for the operation to complete. A value of 0 causes the call to return immediately. Defaults to no timeout (-1).

PARAMETER **espServerIpPort** The url of the ESP file copying service. Defaults to the value of ws_fs_server in the environment.

PARAMETER **maxConnections** The maximum number of target nodes to write to concurrently. Defaults to 1.

PARAMETER **allowOverwrite** Is it valid to overwrite an existing file of the same name? Defaults to FALSE

PARAMETER **replicate** Whether to replicate the new file. Defaults to FALSE.

PARAMETER **compress** Whether to compress the new file. Defaults to FALSE.

PARAMETER **failIfNoSourceFile** If TRUE it causes a missing source file to trigger a failure. Defaults to FALSE.

PARAMETER **expireDays** Number of days to auto-remove file. Default is -1, not expire.

RETURN The DFU workunit id for the job.

FUNCTION SprayFixed

File \

	SprayFixed
	`(varstring sourceIP, varstring sourcePath, integer4 recordSize, varstring destinationGroup, varstring destinationLogicalName, integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1, boolean allowOverwrite=FALSE, boolean replicate=FALSE, boolean compress=FALSE, boolean failIfNoSourceFile=FALSE, integer4 expireDays=-1)`

Same as fSprayFixed, but does not return the DFU Workunit ID.

SEE fSprayFixed

FUNCTION fSprayVariable

File \

varstring	fSprayVariable
(varstring sourceIP, varstring sourcePath, integer4 sourceMaxRecordSize=8192, varstring sourceCsvSeparate='\\,', varstring sourceCsvTerminate='\\n,\\r\\n', varstring sourceCsvQuote='\"', varstring destinationGroup, varstring destinationLogicalName, integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1, boolean allowOverwrite=FALSE, boolean replicate=FALSE, boolean compress=FALSE, varstring sourceCsvEscape=", boolean failIfNoSourceFile=FALSE, boolean recordStructurePresent=FALSE, boolean quotedTerminator=TRUE, varstring encoding='ascii', integer4 expireDays=-1)	

FUNCTION SprayVariable

File \

	SprayVariable
(varstring sourceIP, varstring sourcePath, integer4 sourceMaxRecordSize=8192, varstring sourceCsvSeparate='\\,', varstring sourceCsvTerminate='\\n,\\r\\n', varstring sourceCsvQuote='\"', varstring destinationGroup, varstring destinationLogicalName, integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1, boolean allowOverwrite=FALSE, boolean replicate=FALSE, boolean compress=FALSE, varstring sourceCsvEscape=", boolean failIfNoSourceFile=FALSE, boolean recordStructurePresent=FALSE, boolean quotedTerminator=TRUE, varstring encoding='ascii', integer4 expireDays=-1)	

FUNCTION fSprayDelimited

File \

varstring	fSprayDelimited

```
(varstring sourceIP, varstring sourcePath, integer4
sourceMaxRecordSize=8192, varstring sourceCsvSeparate='\\,', varstring
sourceCsvTerminate='\\n,\\r\\n', varstring sourceCsvQuote='\"', varstring
destinationGroup, varstring destinationLogicalName, integer4 timeOut=-1,
varstring espServerIpPort=GETENV('ws_fs_server'), integer4
maxConnections=-1, boolean allowOverwrite=FALSE, boolean replicate=FALSE,
boolean compress=FALSE, varstring sourceCsvEscape=", boolean
failIfNoSourceFile=FALSE, boolean recordStructurePresent=FALSE, boolean
quotedTerminator=TRUE, varstring encoding='ascii', integer4 expireDays=-1)
```

Sprays a file of fixed delimited records from a single machine and distributes it across the nodes of the destination group.

PARAMETER **sourceIP** The IP address of the file.

PARAMETER **sourcePath** The path and name of the file.

PARAMETER **sourceCsvSeparate** The character sequence which separates fields in the file.

PARAMETER **sourceCsvTerminate** The character sequence which separates records in the file.

PARAMETER **sourceCsvQuote** A string which can be used to delimit fields in the file.

PARAMETER **sourceMaxRecordSize** The maximum size (in bytes) of the records in the file.

PARAMETER **destinationGroup** The name of the group to distribute the file across.

PARAMETER **destinationLogicalName** The logical name of the file to create.

PARAMETER **timeOut** The time in ms to wait for the operation to complete. A value of 0 causes the call to return immediately. Defaults to no timeout (-1).

PARAMETER **espServerIpPort** The url of the ESP file copying service. Defaults to the value of ws_fs_server in the environment.

PARAMETER **maxConnections** The maximum number of target nodes to write to concurrently. Defaults to 1.

PARAMETER **allowOverwrite** Is it valid to overwrite an existing file of the same name? Defaults to FALSE

PARAMETER **replicate** Whether to replicate the new file. Defaults to FALSE.

PARAMETER **compress** Whether to compress the new file. Defaults to FALSE.

PARAMETER **sourceCsvEscape** A character that is used to escape quote characters. Defaults to none.

PARAMETER **failIfNoSourceFile** If TRUE it causes a missing source file to trigger a failure. Defaults to FALSE.

PARAMETER **recordStructurePresent** If TRUE derives the record structure from the header of the file.

PARAMETER **quotedTerminator** Can the terminator character be included in a quoted field. Defaults to TRUE. If FALSE it allows quicker partitioning of the file (avoiding a complete file scan).

PARAMETER **expireDays** Number of days to auto-remove file. Default is -1, not expire.

RETURN The DFU workunit id for the job.

FUNCTION SprayDelimited

File \

	SprayDelimited

```
(varstring sourceIP, varstring sourcePath, integer4 sourceMaxRecordSize=8192,
varstring sourceCsvSeparate='\\,', varstring sourceCsvTerminate='\\n,\\r\\n',
varstring sourceCsvQuote='\"', varstring destinationGroup, varstring
destinationLogicalName, integer4 timeOut=-1, varstring
espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1, boolean
allowOverwrite=FALSE, boolean replicate=FALSE, boolean compress=FALSE, varstring
sourceCsvEscape=", boolean failIfNoSourceFile=FALSE, boolean
recordStructurePresent=FALSE, boolean quotedTerminator=TRUE, const varstring
encoding='ascii', integer4 expireDays=-1)
```

Same as fSprayDelimited, but does not return the DFU Workunit ID.

SEE fSprayDelimited

FUNCTION fSprayXml

File \

varstring	fSprayXml

```
(varstring sourceIP, varstring sourcePath, integer4
sourceMaxRecordSize=8192, varstring sourceRowTag, varstring
sourceEncoding='utf8', varstring destinationGroup, varstring
destinationLogicalName, integer4 timeOut=-1, varstring
espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1,
boolean allowOverwrite=FALSE, boolean replicate=FALSE, boolean
compress=FALSE, boolean failIfNoSourceFile=FALSE, integer4 expireDays=-1)
```

Sprays an xml file from a single machine and distributes it across the nodes of the destination group.

PARAMETER **sourceIP** The IP address of the file.

PARAMETER **sourcePath** The path and name of the file.

PARAMETER **sourceMaxRecordSize** The maximum size (in bytes) of the records in the file.

PARAMETER **sourceRowTag** The xml tag that is used to delimit records in the source file. (This tag cannot recursivly nest.)

PARAMETER **sourceEncoding** The unicode encoding of the file. (utf8,utf8n,utf16be,utf16le,utf32be,utf32le)

PARAMETER **destinationGroup** The name of the group to distribute the file across.

PARAMETER **destinationLogicalName** The logical name of the file to create.

PARAMETER **timeOut** The time in ms to wait for the operation to complete. A value of 0 causes the call to return immediately. Defaults to no timeout (-1).

PARAMETER **espServerIpPort** The url of the ESP file copying service. Defaults to the value of ws_fs_server in the environment.

PARAMETER **maxConnections** The maximum number of target nodes to write to concurrently. Defaults to 1.

PARAMETER **allowOverwrite** Is it valid to overwrite an existing file of the same name? Defaults to FALSE

PARAMETER **replicate** Whether to replicate the new file. Defaults to FALSE.

PARAMETER **compress** Whether to compress the new file. Defaults to FALSE.

PARAMETER **failIfNoSourceFile** If TRUE it causes a missing source file to trigger a failure. Defaults to FALSE.

PARAMETER **expireDays** Number of days to auto-remove file. Default is -1, not expire.

RETURN The DFU workunit id for the job.

FUNCTION SprayXml

File \

	SprayXml
	(varstring sourceIP, varstring sourcePath, integer4 sourceMaxRecordSize=8192, varstring sourceRowTag, varstring sourceEncoding='utf8', varstring destinationGroup, varstring destinationLogicalName, integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1, boolean allowOverwrite=FALSE, boolean replicate=FALSE, boolean compress=FALSE, boolean failIfNoSourceFile=FALSE, integer4 expireDays=-1)

Same as fSprayXml, but does not return the DFU Workunit ID.

SEE fSprayXml

FUNCTION fDespray

File \

varstring	fDespray
	(varstring logicalName, varstring destinationIP, varstring destinationPath, integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1, boolean allowOverwrite=FALSE)

Copies a distributed file from multiple machines, and desprays it to a single file on a single machine.

PARAMETER **logicalName** The name of the file to despray.

PARAMETER **destinationIP** The IP of the target machine.

PARAMETER **destinationPath** The path of the file to create on the destination machine.

PARAMETER **timeOut** The time in ms to wait for the operation to complete. A value of 0 causes the call to return immediately. Defaults to no timeout (-1).

PARAMETER **espServerIpPort** The url of the ESP file copying service. Defaults to the value of ws_fs_server in the environment.

PARAMETER **maxConnections** The maximum number of target nodes to write to concurrently. Defaults to 1.

PARAMETER **allowOverwrite** Is it valid to overwrite an existing file of the same name? Defaults to FALSE

RETURN The DFU workunit id for the job.

FUNCTION Despray

File \

	Despray
	`(varstring logicalName, varstring destinationIP, varstring destinationPath, integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1, boolean allowOverwrite=FALSE)`

Same as fDespray, but does not return the DFU Workunit ID.

SEE fDespray

FUNCTION fCopy

File \

varstring	fCopy
	`(varstring sourceLogicalName, varstring destinationGroup, varstring destinationLogicalName, varstring sourceDali=", integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1, boolean allowOverwrite=FALSE, boolean replicate=FALSE, boolean asSuperfile=FALSE, boolean compress=FALSE, boolean forcePush=FALSE, integer4 transferBufferSize=0, boolean preserveCompression=TRUE)`

Copies a distributed file to another distributed file.

PARAMETER **sourceLogicalName** The name of the file to despray.

PARAMETER **destinationGroup** The name of the group to distribute the file across.

PARAMETER **destinationLogicalName** The logical name of the file to create.

PARAMETER **sourceDali** The dali that contains the source file (blank implies same dali). Defaults to same dali.

PARAMETER **timeOut** The time in ms to wait for the operation to complete. A value of 0 causes the call to return immediately. Defaults to no timeout (-1).

PARAMETER **espServerIpPort** The url of the ESP file copying service. Defaults to the value of ws_fs_server in the environment.

PARAMETER **maxConnections** The maximum number of target nodes to write to concurrently. Defaults to 1.

PARAMETER **allowOverwrite** Is it valid to overwrite an existing file of the same name? Defaults to FALSE

PARAMETER **replicate** Should the copied file also be replicated on the destination? Defaults to FALSE

PARAMETER **asSuperfile** Should the file be copied as a superfile? If TRUE and source is a superfile, then the operation creates a superfile on the target, creating sub-files as needed and only overwriting existing sub-files whose content has changed. If FALSE, a single file is created. Defaults to FALSE.

PARAMETER **compress** Whether to compress the new file. Defaults to FALSE.

PARAMETER **forcePush** Should the copy process be executed on the source nodes (push) or on the destination nodes (pull)? Default is to pull.

PARAMETER **transferBufferSize** Overrides the size (in bytes) of the internal buffer used to copy the file. Default is 64k.

RETURN The DFU workunit id for the job.

FUNCTION Copy

File \

	Copy
	`(varstring sourceLogicalName, varstring destinationGroup, varstring destinationLogicalName, varstring sourceDali=", integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'), integer4 maxConnections=-1, boolean allowOverwrite=FALSE, boolean replicate=FALSE, boolean asSuperfile=FALSE, boolean compress=FALSE, boolean forcePush=FALSE, integer4 transferBufferSize=0, boolean preserveCompression=TRUE)`

Same as fCopy, but does not return the DFU Workunit ID.

SEE fCopy

FUNCTION fReplicate

File \

varstring	**fReplicate**
`(varstring logicalName, integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'))`	

Ensures the specified file is replicated to its mirror copies.

PARAMETER **logicalName** The name of the file to replicate.

PARAMETER **timeOut** The time in ms to wait for the operation to complete. A value of 0 causes the call to return immediately. Defaults to no timeout (-1).

PARAMETER **espServerIpPort** The url of the ESP file copying service. Defaults to the value of ws_fs_server in the environment.

RETURN The DFU workunit id for the job.

FUNCTION Replicate

File \

	Replicate
`(varstring logicalName, integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'))`	

Same as fReplicated, but does not return the DFU Workunit ID.

SEE fReplicate

FUNCTION fRemotePull

File \

varstring	fRemotePull
(varstring remoteEspFsURL, varstring sourceLogicalName, varstring destinationGroup, varstring destinationLogicalName, integer4 timeOut=-1, integer4 maxConnections=-1, boolean allowOverwrite=FALSE, boolean replicate=FALSE, boolean asSuperfile=FALSE, boolean forcePush=FALSE, integer4 transferBufferSize=0, boolean wrap=FALSE, boolean compress=FALSE)	

Copies a distributed file to a distributed file on remote system. Similar to fCopy, except the copy executes remotely. Since the DFU workunit executes on the remote DFU server, the user name authentication must be the same on both systems, and the user must have rights to copy files on both systems.

PARAMETER **remoteEspFsURL** The url of the remote ESP file copying service.

PARAMETER **sourceLogicalName** The name of the file to despray.

PARAMETER **destinationGroup** The name of the group to distribute the file across.

PARAMETER **destinationLogicalName** The logical name of the file to create.

PARAMETER **timeOut** The time in ms to wait for the operation to complete. A value of 0 causes the call to return immediately. Defaults to no timeout (-1).

PARAMETER **maxConnections** The maximum number of target nodes to write to concurrently. Defaults to 1.

PARAMETER **allowOverwrite** Is it valid to overwrite an existing file of the same name? Defaults to FALSE

PARAMETER **replicate** Should the copied file also be replicated on the destination? Defaults to FALSE

PARAMETER **asSuperfile** Should the file be copied as a superfile? If TRUE and source is a superfile, then the operation creates a superfile on the target, creating sub-files as needed and only overwriting existing sub-files whose content has changed. If FALSE a single file is created. Defaults to FALSE.

PARAMETER **compress** Whether to compress the new file. Defaults to FALSE.

PARAMETER **forcePush** Should the copy process should be executed on the source nodes (push) or on the destination nodes (pull)? Default is to pull.

PARAMETER **transferBufferSize** Overrides the size (in bytes) of the internal buffer used to copy the file. Default is 64k.

PARAMETER **wrap** Should the fileparts be wrapped when copying to a smaller sized cluster? The default is FALSE.

RETURN The DFU workunit id for the job.

FUNCTION RemotePull

File \

	RemotePull
	`(varstring remoteEspFsURL, varstring sourceLogicalName, varstring destinationGroup, varstring destinationLogicalName, integer4 timeOut=-1, integer4 maxConnections=-1, boolean allowOverwrite=FALSE, boolean replicate=FALSE, boolean asSuperfile=FALSE, boolean forcePush=FALSE, integer4 transferBufferSize=0, boolean wrap=FALSE, boolean compress=FALSE)`

Same as fRemotePull, but does not return the DFU Workunit ID.

SEE fRemotePull

FUNCTION fMonitorLogicalFileName

File \

varstring	fMonitorLogicalFileName
`(varstring eventToFire, varstring name, integer4 shotCount=1, varstring espServerIpPort=GETENV('ws_fs_server'))`	

Creates a file monitor job in the DFU Server. If an appropriately named file arrives in this interval it will fire the event with the name of the triggering object as the event subtype (see the EVENT function).

PARAMETER **eventToFire** The user-defined name of the event to fire when the filename appears. This value is used as the first parameter to the EVENT function.

PARAMETER **name** The name of the logical file to monitor. This may contain wildcard characters (* and ?)

PARAMETER **shotCount** The number of times to generate the event before the monitoring job completes. A value of -1 indicates the monitoring job continues until manually aborted. The default is 1.

PARAMETER **espServerIpPort** The url of the ESP file copying service. Defaults to the value of ws_fs_server in the environment.

RETURN The DFU workunit id for the job.

FUNCTION MonitorLogicalFileName

File \

	MonitorLogicalFileName
	`(varstring eventToFire, varstring name, integer4 shotCount=1, varstring espServerIpPort=GETENV('ws_fs_server'))`

Same as fMonitorLogicalFileName, but does not return the DFU Workunit ID.

SEE fMonitorLogicalFileName

FUNCTION fMonitorFile

File \

varstring	fMonitorFile
	`(varstring eventToFire, varstring ip, varstring filename, boolean subDirs=FALSE, integer4 shotCount=1, varstring espServerIpPort=GETENV('ws_fs_server'))`

Creates a file monitor job in the DFU Server. If an appropriately named file arrives in this interval it will fire the event with the name of the triggering object as the event subtype (see the EVENT function).

PARAMETER **eventToFire** The user-defined name of the event to fire when the filename appears. This value is used as the first parameter to the EVENT function.

PARAMETER **ip** The the IP address for the file to monitor. This may be omitted if the filename parameter contains a complete URL.

PARAMETER **filename** The full path of the file(s) to monitor. This may contain wildcard characters (* and ?)

PARAMETER **subDirs** Whether to include files in sub-directories (when the filename contains wildcards). Defaults to FALSE.

PARAMETER **shotCount** The number of times to generate the event before the monitoring job completes. A value of -1 indicates the monitoring job continues until manually aborted. The default is 1.

PARAMETER **espServerIpPort** The url of the ESP file copying service. Defaults to the value of ws_fs_server in the environment.

RETURN The DFU workunit id for the job.

FUNCTION MonitorFile

File \

	MonitorFile
	`(varstring eventToFire, varstring ip, varstring filename, boolean subdirs=FALSE, integer4 shotCount=1, varstring espServerIpPort=GETENV('ws_fs_server'))`

Same as fMonitorFile, but does not return the DFU Workunit ID.

SEE fMonitorFile

FUNCTION WaitDfuWorkunit

File \

varstring	WaitDfuWorkunit
	`(varstring wuid, integer4 timeOut=-1, varstring espServerIpPort=GETENV('ws_fs_server'))`

Waits for the specified DFU workunit to finish.

PARAMETER **wuid** The dfu wfid to wait for.

PARAMETER **timeOut** The time in ms to wait for the operation to complete. A value of 0 causes the call to return immediately. Defaults to no timeout (-1).

PARAMETER **espServerIpPort** The url of the ESP file copying service. Defaults to the value of ws_fs_server in the environment.

RETURN A string containing the final status string of the DFU workunit.

FUNCTION AbortDfuWorkunit

File \

	AbortDfuWorkunit
	`(varstring wuid, varstring espServerIpPort=GETENV('ws_fs_server'))`

Aborts the specified DFU workunit.

PARAMETER **wuid** The dfu wfid to abort.

PARAMETER **espServerIpPort** The url of the ESP file copying service. Defaults to the value of ws_fs_server in the environment.

FUNCTION CreateSuperFile

File \

	CreateSuperFile
	`(varstring superName, boolean sequentialParts=FALSE, boolean allowExist=FALSE)`

Creates an empty superfile. This function is not included in a superfile transaction.

PARAMETER **superName** The logical name of the superfile.

PARAMETER **sequentialParts** Whether the sub-files must be sequentially ordered. Default to FALSE.

PARAMETER **allowExist** Indicating whether to post an error if the superfile already exists. If TRUE, no error is posted. Defaults to FALSE.

FUNCTION SuperFileExists

File \

boolean	SuperFileExists
(varstring superName)	

Checks if the specified filename is present in the Distributed File Utility (DFU) and is a SuperFile.

PARAMETER **superName** The logical name of the superfile.

RETURN Whether the file exists.

SEE FileExists

FUNCTION DeleteSuperFile

File \

	DeleteSuperFile
(varstring superName, boolean deletesub=FALSE)	

Deletes the superfile.

PARAMETER **superName** The logical name of the superfile.

SEE FileExists

FUNCTION GetSuperFileSubCount

File \

unsigned4	GetSuperFileSubCount
(varstring superName)	

Returns the number of sub-files contained within a superfile.

PARAMETER **superName** The logical name of the superfile.

RETURN The number of sub-files within the superfile.

FUNCTION GetSuperFileSubName

File \

varstring	GetSuperFileSubName
(varstring superName, unsigned4 fileNum, boolean absPath=FALSE)	

Returns the name of the Nth sub-file within a superfile.

PARAMETER **superName** The logical name of the superfile.

PARAMETER **fileNum** The 1-based position of the sub-file to return the name of.

PARAMETER **absPath** Whether to prepend '~' to the name of the resulting logical file name.

RETURN The logical name of the selected sub-file.

FUNCTION FindSuperFileSubName

File \

unsigned4	FindSuperFileSubName
(varstring superName, varstring subName)	

Returns the position of a file within a superfile.

PARAMETER **superName** The logical name of the superfile.

PARAMETER **subName** The logical name of the sub-file.

RETURN The 1-based position of the sub-file within the superfile.

FUNCTION StartSuperFileTransaction

File \

	StartSuperFileTransaction
	`()`

Starts a superfile transaction. All superfile operations within the transaction will either be executed atomically or rolled back when the transaction is finished.

FUNCTION AddSuperFile

File \

	AddSuperFile
	`(varstring superName, varstring subName, unsigned4 atPos=0, boolean addContents=FALSE, boolean strict=FALSE)`

Adds a file to a superfile.

PARAMETER **superName** The logical name of the superfile.

PARAMETER **subName** The name of the logical file to add.

PARAMETER **atPos** The position to add the sub-file, or 0 to append. Defaults to 0.

PARAMETER **addContents** Controls whether adding a superfile adds the superfile, or its contents. Defaults to FALSE (do not expand).

PARAMETER **strict** Check addContents only if subName is a superfile, and ensure superfiles exist.

FUNCTION RemoveSuperFile

File \

	RemoveSuperFile
	`(varstring superName, varstring subName, boolean del=FALSE, boolean removeContents=FALSE)`

Removes a sub-file from a superfile.

PARAMETER **superName** The logical name of the superfile.

PARAMETER **subName** The name of the sub-file to remove.

PARAMETER **del** Indicates whether the sub-file should also be removed from the disk. Defaults to FALSE.

PARAMETER **removeContents** Controls whether the contents of a sub-file which is a superfile should be recursively removed. Defaults to FALSE.

FUNCTION ClearSuperFile

File \

	ClearSuperFile
	`(varstring superName, boolean del=FALSE)`

Removes all sub-files from a superfile.

PARAMETER **superName** The logical name of the superfile.

PARAMETER **del** Indicates whether the sub-files should also be removed from the disk. Defaults to FALSE.

FUNCTION RemoveOwnedSubFiles

File \

	RemoveOwnedSubFiles
	`(varstring superName, boolean del=FALSE)`

Removes all soley-owned sub-files from a superfile. If a sub-file is also contained within another superfile then it is retained.

PARAMETER **superName** The logical name of the superfile.

FUNCTION DeleteOwnedSubFiles

File \

	DeleteOwnedSubFiles
	`(varstring superName)`

Legacy version of RemoveOwnedSubFiles which was incorrectly named in a previous version.

SEE RemoveOwnedSubFIles

FUNCTION SwapSuperFile

File \

	SwapSuperFile
	`(varstring superName1, varstring superName2)`

Swap the contents of two superfiles.

PARAMETER **superName1** The logical name of the first superfile.

PARAMETER **superName2** The logical name of the second superfile.

FUNCTION ReplaceSuperFile

File \

	ReplaceSuperFile
	`(varstring superName, varstring oldSubFile, varstring newSubFile)`

Removes a sub-file from a superfile and replaces it with another.

PARAMETER **superName** The logical name of the superfile.

PARAMETER **oldSubFile** The logical name of the sub-file to remove.

PARAMETER **newSubFile** The logical name of the sub-file to replace within the superfile.

FUNCTION FinishSuperFileTransaction

File \

	FinishSuperFileTransaction
(boolean rollback=FALSE)	

Finishes a superfile transaction. This executes all the operations since the matching StartSuperFileTransaction(). If there are any errors, then all of the operations are rolled back.

FUNCTION SuperFileContents

File \

dataset(FsLogicalFileNameRecord)	SuperFileContents
(varstring superName, boolean recurse=FALSE)	

Returns the list of sub-files contained within a superfile.

PARAMETER **superName** The logical name of the superfile.

PARAMETER **recurse** Should the contents of child-superfiles be expanded. Default is FALSE.

RETURN A dataset containing the names of the sub-files.

FUNCTION LogicalFileSuperOwners

File \

dataset(FsLogicalFileNameRecord)	**LogicalFileSuperOwners**
(varstring name)	

Returns the list of superfiles that a logical file is contained within.

PARAMETER **name** The name of the logical file.

RETURN A dataset containing the names of the superfiles.

FUNCTION LogicalFileSuperSubList

File \

dataset(FsLogicalSuperSubRecord)	**LogicalFileSuperSubList**
()	

Returns the list of all the superfiles in the system and their component sub-files.

RETURN A dataset containing pairs of superName,subName for each component file.

FUNCTION fPromoteSuperFileList

File \

varstring	**fPromoteSuperFileList**
(set of varstring superNames, varstring addHead=", boolean delTail=FALSE, boolean createOnlyOne=FALSE, boolean reverse=FALSE)	

Moves the sub-files from the first entry in the list of superfiles to the next in the list, repeating the process through the list of superfiles.

PARAMETER **superNames** A set of the names of the superfiles to act on. Any that do not exist will be created. The contents of each superfile will be moved to the next in the list.

PARAMETER **addHead** A string containing a comma-delimited list of logical file names to add to the first superfile after the promotion process is complete. Defaults to ”.

PARAMETER **delTail** Indicates whether to physically delete the contents moved out of the last superfile. The default is FALSE.

PARAMETER **createOnlyOne** Specifies whether to only create a single superfile (truncate the list at the first non-existent superfile). The default is FALSE.

PARAMETER **reverse** Reverse the order of processing the superfiles list, effectively ’demoting’ instead of ’promoting’ the sub-files. The default is FALSE.

RETURN A string containing a comma separated list of the previous sub-file contents of the emptied superfile.

FUNCTION PromoteSuperFileList

File \

	PromoteSuperFileList
	`(set of varstring superNames, varstring addHead=", boolean delTail=FALSE, boolean createOnlyOne=FALSE, boolean reverse=FALSE)`

Same as fPromoteSuperFileList, but does not return the DFU Workunit ID.

SEE fPromoteSuperFileList

---

math

Go Up

DESCRIPTIONS

MODULE Math

	Math

Children

1. Infinity : Return a real ”infinity” value
2. NaN : Return a non-signalling NaN (Not a Number)value
3. isInfinite : Return whether a real value is infinite (positive or negative)
4. isNaN : Return whether a real value is a NaN (not a number) value
5. isFinite : Return whether a real value is a valid value (neither infinite not NaN)
6. FMod : Returns the floating-point remainder of numer/denom (rounded towards zero)
7. FMatch : Returns whether two floating point values are the same, within margin of error epsilon

---

ATTRIBUTE Infinity

Math \

REAL8	**Infinity**

Return a real ”infinity” value.

ATTRIBUTE NaN

Math \

REAL8	**NaN**

Return a non-signalling NaN (Not a Number)value.

FUNCTION isInfinite

Math \

BOOLEAN	**isInfinite**
(REAL8 val)	

Return whether a real value is infinite (positive or negative).

PARAMETER **val** The value to test.

FUNCTION isNaN

Math \

BOOLEAN	**isNaN**
(REAL8 val)	

Return whether a real value is a NaN (not a number) value.

PARAMETER **val** The value to test.

FUNCTION isFinite

Math \

BOOLEAN	**isFinite**
(REAL8 val)	

Return whether a real value is a valid value (neither infinite not NaN).

PARAMETER **val** The value to test.

FUNCTION FMod

Math \

REAL8	**FMod**
(REAL8 numer, REAL8 denom)	

Returns the floating-point remainder of numer/denom (rounded towards zero). If denom is zero, the result depends on the -fdivideByZero flag: 'zero' or unset: return zero. 'nan': return a non-signalling NaN value 'fail': throw an exception

PARAMETER **numer** The numerator.

PARAMETER **denom** The numerator.

FUNCTION FMatch

Math \

BOOLEAN	**FMatch**
(REAL8 a, REAL8 b, REAL8 epsilon=0.0)	

Returns whether two floating point values are the same, within margin of error epsilon.

PARAMETER **a** The first value.

PARAMETER **b** The second value.

PARAMETER **epsilon** The allowable margin of error.

Metaphone

Go Up

IMPORTS

lib_metaphone |

DESCRIPTIONS

MODULE Metaphone

	Metaphone

Children

1. primary : Returns the primary metaphone value
2. secondary : Returns the secondary metaphone value
3. double : Returns the double metaphone value (primary and secondary concatenated

---

FUNCTION primary

Metaphone \

String	**primary**
(STRING src)	

Returns the primary metaphone value

PARAMETER **src** The string whose metphone is to be calculated.

SEE http://en.wikipedia.org/wiki/Metaphone#Double_Metaphone

FUNCTION secondary

Metaphone \

String	secondary
(STRING src)	

Returns the secondary metaphone value

PARAMETER **src** The string whose metphone is to be calculated.

SEE http://en.wikipedia.org/wiki/Metaphone#Double_Metaphone

FUNCTION double

Metaphone \

String	double
(STRING src)	

Returns the double metaphone value (primary and secondary concatenated

PARAMETER **src** The string whose metphone is to be calculated.

SEE http://en.wikipedia.org/wiki/Metaphone#Double_Metaphone

---

str

Go Up

IMPORTS

lib_stringlib |

DESCRIPTIONS

MODULE Str

	Str

Children

1. CompareIgnoreCase : Compares the two strings case insensitively
2. EqualIgnoreCase : Tests whether the two strings are identical ignoring differences in case
3. Find : Returns the character position of the nth match of the search string with the first string
4. FindCount : Returns the number of occurences of the second string within the first string
5. WildMatch : Tests if the search string matches the pattern
6. Contains : Tests if the search string contains each of the characters in the pattern
7. FilterOut : Returns the first string with all characters within the second string removed
8. Filter : Returns the first string with all characters not within the second string removed
9. SubstituteIncluded : Returns the source string with the replacement character substituted for all characters included in the filter string
10. SubstituteExcluded : Returns the source string with the replacement character substituted for all characters not included in the filter string

11. Translate : Returns the source string with the all characters that match characters in the search string replaced with the character at the corresponding position in the replacement string
12. ToLowerCase : Returns the argument string with all upper case characters converted to lower case
13. ToUpperCase : Return the argument string with all lower case characters converted to upper case
14. ToCapitalCase : Returns the argument string with the first letter of each word in upper case and all other letters left as-is
15. ToTitleCase : Returns the argument string with the first letter of each word in upper case and all other letters lower case
16. Reverse : Returns the argument string with all characters in reverse order
17. FindReplace : Returns the source string with the replacement string substituted for all instances of the search string
18. Extract : Returns the nth element from a comma separated string
19. CleanSpaces : Returns the source string with all instances of multiple adjacent space characters (2 or more spaces together) reduced to a single space character
20. StartsWith : Returns true if the prefix string matches the leading characters in the source string
21. EndsWith : Returns true if the suffix string matches the trailing characters in the source string
22. RemoveSuffix : Removes the suffix from the search string, if present, and returns the result
23. ExtractMultiple : Returns a string containing a list of elements from a comma separated string
24. CountWords : Returns the number of words that the string contains
25. SplitWords : Returns the list of words extracted from the string
26. CombineWords : Returns the list of words extracted from the string
27. EditDistance : Returns the minimum edit distance between the two strings
28. EditDistanceWithinRadius : Returns true if the minimum edit distance between the two strings is with a specific range
29. WordCount : Returns the number of words in the string
30. GetNthWord : Returns the n-th word from the string
31. ExcludeFirstWord : Returns everything except the first word from the string
32. ExcludeLastWord : Returns everything except the last word from the string
33. ExcludeNthWord : Returns everything except the nth word from the string
34. FindWord : Tests if the search string contains the supplied word as a whole word
35. Repeat
36. ToHexPairs

37. FromHexPairs
38. EncodeBase64
39. DecodeBase64

FUNCTION CompareIgnoreCase

Str \

INTEGER4	**CompareIgnoreCase**
(STRING src1, STRING src2)	

Compares the two strings case insensitively. Returns a negative integer, zero, or a positive integer according to whether the first string is less than, equal to, or greater than the second.

PARAMETER **src1** The first string to be compared.

PARAMETER **src2** The second string to be compared.

SEE Str.EqualIgnoreCase

FUNCTION EqualIgnoreCase

Str \

BOOLEAN	**EqualIgnoreCase**
(STRING src1, STRING src2)	

Tests whether the two strings are identical ignoring differences in case.

PARAMETER **src1** The first string to be compared.

PARAMETER **src2** The second string to be compared.

SEE Str.CompareIgnoreCase

FUNCTION Find

Str \

UNSIGNED4	Find
(STRING src, STRING sought, UNSIGNED4 instance = 1)	

Returns the character position of the nth match of the search string with the first string. If no match is found the attribute returns 0. If an instance is omitted the position of the first instance is returned.

PARAMETER **src** The string that is searched

PARAMETER **sought** The string being sought.

PARAMETER **instance** Which match instance are we interested in?

FUNCTION FindCount

Str \

UNSIGNED4	FindCount
(STRING src, STRING sought)	

Returns the number of occurences of the second string within the first string.

PARAMETER **src** The string that is searched

PARAMETER **sought** The string being sought.

FUNCTION WildMatch

Str \

BOOLEAN	WildMatch
(STRING src, STRING _pattern, BOOLEAN ignore_case)	

Tests if the search string matches the pattern. The pattern can contain wildcards '?' (single character) and '*' (multiple character).

PARAMETER **src** The string that is being tested.

PARAMETER **pattern** The pattern to match against.

PARAMETER **ignore_case** Whether to ignore differences in case between characters

---

FUNCTION Contains

Str \

BOOLEAN	**Contains**
(STRING src, STRING _pattern, BOOLEAN ignore_case)	

Tests if the search string contains each of the characters in the pattern. If the pattern contains duplicate characters those characters will match once for each occurence in the pattern.

PARAMETER **src** The string that is being tested.

PARAMETER **pattern** The pattern to match against.

PARAMETER **ignore_case** Whether to ignore differences in case between characters

---

FUNCTION FilterOut

Str \

STRING	**FilterOut**
(STRING src, STRING filter)	

Returns the first string with all characters within the second string removed.

PARAMETER **src** The string that is being tested.

PARAMETER **filter** The string containing the set of characters to be excluded.

SEE Str.Filter

FUNCTION Filter

Str \

STRING	**Filter**
`(STRING src, STRING filter)`	

Returns the first string with all characters not within the second string removed.

PARAMETER **src** The string that is being tested.

PARAMETER **filter** The string containing the set of characters to be included.

SEE Str.FilterOut

FUNCTION SubstituteIncluded

Str \

STRING	**SubstituteIncluded**
`(STRING src, STRING filter, STRING1 replace_char)`	

Returns the source string with the replacement character substituted for all characters included in the filter string. MORE: Should this be a general string substitution?

PARAMETER **src** The string that is being tested.

PARAMETER **filter** The string containing the set of characters to be included.

PARAMETER **replace_char** The character to be substituted into the result.

SEE Std.Str.Translate, Std.Str.SubstituteExcluded

FUNCTION SubstituteExcluded

Str \

STRING	SubstituteExcluded
(STRING src, STRING filter, STRING1 replace_char)	

Returns the source string with the replacement character substituted for all characters not included in the filter string. MORE: Should this be a general string substitution?

PARAMETER **src** The string that is being tested.

PARAMETER **filter** The string containing the set of characters to be included.

PARAMETER **replace_char** The character to be substituted into the result.

SEE Std.Str.SubstituteIncluded

FUNCTION Translate

Str \

STRING	Translate
(STRING src, STRING search, STRING replacement)	

Returns the source string with the all characters that match characters in the search string replaced with the character at the corresponding position in the replacement string.

PARAMETER **src** The string that is being tested.

PARAMETER **search** The string containing the set of characters to be included.

PARAMETER **replacement** The string containing the characters to act as replacements.

SEE Std.Str.SubstituteIncluded

FUNCTION ToLowerCase

Str \

STRING	ToLowerCase
(STRING src)	

Returns the argument string with all upper case characters converted to lower case.

PARAMETER **src** The string that is being converted.

FUNCTION ToUpperCase

Str \

STRING	ToUpperCase
(STRING src)	

Return the argument string with all lower case characters converted to upper case.

PARAMETER **src** The string that is being converted.

FUNCTION ToCapitalCase

Str \

STRING	ToCapitalCase
(STRING src)	

Returns the argument string with the first letter of each word in upper case and all other letters left as-is. A contiguous sequence of alphanumeric characters is treated as a word.

PARAMETER **src** The string that is being converted.

FUNCTION ToTitleCase

Str \

STRING	**ToTitleCase**
(STRING src)	

Returns the argument string with the first letter of each word in upper case and all other letters lower case. A contiguous sequence of alphanumeric characters is treated as a word.

PARAMETER **src** The string that is being converted.

FUNCTION Reverse

Str \

STRING	**Reverse**
(STRING src)	

Returns the argument string with all characters in reverse order. Note the argument is not TRIMMED before it is reversed.

PARAMETER **src** The string that is being reversed.

FUNCTION FindReplace

Str \

STRING	**FindReplace**
(STRING src, STRING sought, STRING replacement)	

Returns the source string with the replacement string substituted for all instances of the search string.

PARAMETER **src** The string that is being transformed.

PARAMETER **sought** The string to be replaced.

PARAMETER **replacement** The string to be substituted into the result.

FUNCTION Extract

Str \

STRING	Extract
(STRING src, UNSIGNED4 instance)	

Returns the nth element from a comma separated string.

PARAMETER **src** The string containing the comma separated list.

PARAMETER **instance** Which item to select from the list.

FUNCTION CleanSpaces

Str \

STRING	CleanSpaces
(STRING src)	

Returns the source string with all instances of multiple adjacent space characters (2 or more spaces together) reduced to a single space character. Leading and trailing spaces are removed, and tab characters are converted to spaces.

PARAMETER **src** The string to be cleaned.

FUNCTION StartsWith

Str \

BOOLEAN	StartsWith
(STRING src, STRING prefix)	

Returns true if the prefix string matches the leading characters in the source string. Trailing spaces are stripped from the prefix before matching. // x.myString.StartsWith('x') as an alternative syntax would be even better

PARAMETER **src** The string being searched in.

PARAMETER **prefix** The prefix to search for.

FUNCTION EndsWith

Str \

BOOLEAN	EndsWith
(STRING src, STRING suffix)	

Returns true if the suffix string matches the trailing characters in the source string. Trailing spaces are stripped from both strings before matching.

PARAMETER **src** The string being searched in.

PARAMETER **suffix** The prefix to search for.

FUNCTION RemoveSuffix

Str \

STRING	RemoveSuffix
(STRING src, STRING suffix)	

Removes the suffix from the search string, if present, and returns the result. Trailing spaces are stripped from both strings before matching.

PARAMETER **src** The string being searched in.

PARAMETER **suffix** The prefix to search for.

FUNCTION ExtractMultiple

Str \

STRING	ExtractMultiple
(STRING src, UNSIGNED8 mask)	

Returns a string containing a list of elements from a comma separated string.

PARAMETER **src** The string containing the comma separated list.

PARAMETER **mask** A bitmask of which elements should be included. Bit 0 is item1, bit1 item 2 etc.

FUNCTION CountWords

Str \

UNSIGNED4	CountWords
(STRING src, STRING separator, BOOLEAN allow_blank = FALSE)	

Returns the number of words that the string contains. Words are separated by one or more separator strings. No spaces are stripped from either string before matching.

PARAMETER **src** The string being searched in.

PARAMETER **separator** The string used to separate words

PARAMETER **allow_blank** Indicates if empty/blank string items are included in the results.

FUNCTION SplitWords

Str \

SET OF STRING	SplitWords
(STRING src, STRING separator, BOOLEAN allow_blank = FALSE)	

Returns the list of words extracted from the string. Words are separated by one or more separator strings. No spaces are stripped from either string before matching.

PARAMETER **src** The string being searched in.

PARAMETER **separator** The string used to separate words

PARAMETER **allow_blank** Indicates if empty/blank string items are included in the results.

FUNCTION CombineWords

Str \

STRING	CombineWords
(SET OF STRING words, STRING separator)	

Returns the list of words extracted from the string. Words are separated by one or more separator strings. No spaces are stripped from either string before matching.

PARAMETER **words** The set of strings to be combined.

PARAMETER **separator** The string used to separate words.

FUNCTION EditDistance

Str \

UNSIGNED4	EditDistance
(STRING _left, STRING _right)	

Returns the minimum edit distance between the two strings. An insert change or delete counts as a single edit. The two strings are trimmed before comparing.

PARAMETER **__left** The first string to be compared.

PARAMETER **__right** The second string to be compared.

RETURN The minimum edit distance between the two strings.

FUNCTION EditDistanceWithinRadius

Str \

BOOLEAN	**EditDistanceWithinRadius**
`(STRING _left, STRING _right, UNSIGNED4 radius)`	

Returns true if the minimum edit distance between the two strings is with a specific range. The two strings are trimmed before comparing.

PARAMETER **__left** The first string to be compared.

PARAMETER **__right** The second string to be compared.

PARAMETER **radius** The maximum edit distance that is accepable.

RETURN Whether or not the two strings are within the given specified edit distance.

FUNCTION WordCount

Str \

UNSIGNED4	**WordCount**
`(STRING text)`	

Returns the number of words in the string. Words are separated by one or more spaces.

PARAMETER **text** The string to be broken into words.

RETURN The number of words in the string.

FUNCTION GetNthWord

Str \

STRING	GetNthWord
(STRING text, UNSIGNED4 n)	

Returns the n-th word from the string. Words are separated by one or more spaces.

PARAMETER **text** The string to be broken into words.

PARAMETER **n** Which word should be returned from the function.

RETURN The number of words in the string.

FUNCTION ExcludeFirstWord

Str \

	ExcludeFirstWord
(STRING text)	

Returns everything except the first word from the string. Words are separated by one or more whitespace characters. Whitespace before and after the first word is also removed.

PARAMETER **text** The string to be broken into words.

RETURN The string excluding the first word.

FUNCTION ExcludeLastWord

Str \

	ExcludeLastWord
(STRING text)	

Returns everything except the last word from the string. Words are separated by one or more whitespace characters. Whitespace after a word is removed with the word and leading whitespace is removed with the first word.

PARAMETER **text** The string to be broken into words.

RETURN The string excluding the last word.

FUNCTION ExcludeNthWord

Str \

	ExcludeNthWord
(STRING text, UNSIGNED2 n)	

Returns everything except the nth word from the string. Words are separated by one or more whitespace characters. Whitespace after a word is removed with the word and leading whitespace is removed with the first word.

PARAMETER **text** The string to be broken into words.

PARAMETER **n** Which word should be returned from the function.

RETURN The string excluding the nth word.

FUNCTION FindWord

Str \

BOOLEAN	FindWord
(STRING src, STRING word, BOOLEAN ignore_case=FALSE)	

Tests if the search string contains the supplied word as a whole word.

PARAMETER **src** The string that is being tested.

PARAMETER **word** The word to be searched for.

PARAMETER **ignore_case** Whether to ignore differences in case between characters.

FUNCTION Repeat

Str \

STRING	Repeat
(STRING text, UNSIGNED4 n)	

FUNCTION ToHexPairs

Str \

STRING	ToHexPairs
(DATA value)	

FUNCTION FromHexPairs

Str \

DATA	FromHexPairs
(STRING hex_pairs)	

FUNCTION EncodeBase64

Str \

STRING	EncodeBase64
(DATA value)	

FUNCTION DecodeBase64

Str \

DATA	DecodeBase64
(STRING value)	

Uni

Go Up

IMPORTS

lib_unicodelib |

DESCRIPTIONS

MODULE Uni

	Uni

Children

1. FilterOut : Returns the first string with all characters within the second string removed
2. Filter : Returns the first string with all characters not within the second string removed
3. SubstituteIncluded : Returns the source string with the replacement character substituted for all characters included in the filter string
4. SubstituteExcluded : Returns the source string with the replacement character substituted for all characters not included in the filter string
5. Find : Returns the character position of the nth match of the search string with the first string
6. FindWord : Tests if the search string contains the supplied word as a whole word
7. LocaleFind : Returns the character position of the nth match of the search string with the first string
8. LocaleFindAtStrength : Returns the character position of the nth match of the search string with the first string

9. Extract : Returns the nth element from a comma separated string
10. ToLowerCase : Returns the argument string with all upper case characters converted to lower case
11. ToUpperCase : Return the argument string with all lower case characters converted to upper case
12. ToTitleCase : Returns the upper case variant of the string using the rules for a particular locale
13. LocaleToLowerCase : Returns the lower case variant of the string using the rules for a particular locale
14. LocaleToUpperCase : Returns the upper case variant of the string using the rules for a particular locale
15. LocaleToTitleCase : Returns the upper case variant of the string using the rules for a particular locale
16. CompareIgnoreCase : Compares the two strings case insensitively
17. CompareAtStrength : Compares the two strings case insensitively
18. LocaleCompareIgnoreCase : Compares the two strings case insensitively
19. LocaleCompareAtStrength : Compares the two strings case insensitively
20. Reverse : Returns the argument string with all characters in reverse order
21. FindReplace : Returns the source string with the replacement string substituted for all instances of the search string
22. LocaleFindReplace : Returns the source string with the replacement string substituted for all instances of the search string
23. LocaleFindAtStrengthReplace : Returns the source string with the replacement string substituted for all instances of the search string
24. CleanAccents : Returns the source string with all accented characters replaced with unaccented
25. CleanSpaces : Returns the source string with all instances of multiple adjacent space characters (2 or more spaces together) reduced to a single space character
26. WildMatch : Tests if the search string matches the pattern
27. Contains : Tests if the search string contains each of the characters in the pattern
28. EditDistance : Returns the minimum edit distance between the two strings
29. EditDistanceWithinRadius : Returns true if the minimum edit distance between the two strings is with a specific range
30. WordCount : Returns the number of words in the string
31. GetNthWord : Returns the n-th word from the string

FUNCTION FilterOut

Uni \

unicode	**FilterOut**
(unicode src, unicode filter)	

Returns the first string with all characters within the second string removed.

PARAMETER **src** The string that is being tested.

PARAMETER **filter** The string containing the set of characters to be excluded.

SEE Std.Uni.Filter

FUNCTION Filter

Uni \

unicode	**Filter**
(unicode src, unicode filter)	

Returns the first string with all characters not within the second string removed.

PARAMETER **src** The string that is being tested.

PARAMETER **filter** The string containing the set of characters to be included.

SEE Std.Uni.FilterOut

FUNCTION SubstituteIncluded

Uni \

unicode	**SubstituteIncluded**
(unicode src, unicode filter, unicode replace_char)	

Returns the source string with the replacement character substituted for all characters included in the filter string. MORE: Should this be a general string substitution?

PARAMETER **src** The string that is being tested.

PARAMETER **filter** The string containing the set of characters to be included.

PARAMETER **replace_char** The character to be substituted into the result.

SEE Std.Uni.SubstituteOut

FUNCTION SubstituteExcluded

Uni \

unicode	**SubstituteExcluded**
`(unicode src, unicode filter, unicode replace_char)`	

Returns the source string with the replacement character substituted for all characters not included in the filter string. MORE: Should this be a general string substitution?

PARAMETER **src** The string that is being tested.

PARAMETER **filter** The string containing the set of characters to be included.

PARAMETER **replace_char** The character to be substituted into the result.

SEE Std.Uni.SubstituteIncluded

FUNCTION Find

Uni \

UNSIGNED4	**Find**
`(unicode src, unicode sought, unsigned4 instance)`	

Returns the character position of the nth match of the search string with the first string. If no match is found the attribute returns 0. If an instance is omitted the position of the first instance is returned.

PARAMETER **src** The string that is searched

PARAMETER **sought** The string being sought.

PARAMETER **instance** Which match instance are we interested in?

FUNCTION FindWord

Uni \

BOOLEAN	FindWord
(UNICODE src, UNICODE word, BOOLEAN ignore_case=FALSE)	

Tests if the search string contains the supplied word as a whole word.

PARAMETER **src** The string that is being tested.

PARAMETER **word** The word to be searched for.

PARAMETER **ignore_case** Whether to ignore differences in case between characters.

FUNCTION LocaleFind

Uni \

UNSIGNED4	LocaleFind
(unicode src, unicode sought, unsigned4 instance, varstring locale_name)	

Returns the character position of the nth match of the search string with the first string. If no match is found the attribute returns 0. If an instance is omitted the position of the first instance is returned.

PARAMETER **src** The string that is searched

PARAMETER **sought** The string being sought.

PARAMETER **instance** Which match instance are we interested in?

PARAMETER **locale_name** The locale to use for the comparison

---

FUNCTION LocaleFindAtStrength

Uni \

UNSIGNED4	LocaleFindAtStrength
(unicode src, unicode tofind, unsigned4 instance, varstring locale_name, integer1 strength)	

Returns the character position of the nth match of the search string with the first string. If no match is found the attribute returns 0. If an instance is omitted the position of the first instance is returned.

PARAMETER **src** The string that is searched

PARAMETER **sought** The string being sought.

PARAMETER **instance** Which match instance are we interested in?

PARAMETER **locale_name** The locale to use for the comparison

PARAMETER **strength** The strength of the comparison 1 ignores accents and case, differentiating only between letters 2 ignores case but differentiates between accents. 3 differentiates between accents and case but ignores e.g. differences between Hiragana and Katakana 4 differentiates between accents and case and e.g. Hiragana/Katakana, but ignores e.g. Hebrew cantellation marks 5 differentiates between all strings whose canonically decomposed forms (NFDNormalization Form D) are non-identical

---

FUNCTION Extract

Uni \

unicode	Extract
(unicode src, unsigned4 instance)	

Returns the nth element from a comma separated string.

PARAMETER **src** The string containing the comma separated list.

PARAMETER **instance** Which item to select from the list.

FUNCTION ToLowerCase

Uni \

unicode	**ToLowerCase**
(unicode src)	

Returns the argument string with all upper case characters converted to lower case.

PARAMETER **src** The string that is being converted.

FUNCTION ToUpperCase

Uni \

unicode	**ToUpperCase**
(unicode src)	

Return the argument string with all lower case characters converted to upper case.

PARAMETER **src** The string that is being converted.

FUNCTION ToTitleCase

Uni \

unicode	**ToTitleCase**
(unicode src)	

Returns the upper case variant of the string using the rules for a particular locale.

PARAMETER **src** The string that is being converted.

PARAMETER **locale_name** The locale to use for the comparison

---

FUNCTION LocaleToLowerCase

Uni \

unicode	**LocaleToLowerCase**
`(unicode src, varstring locale_name)`	

Returns the lower case variant of the string using the rules for a particular locale.

PARAMETER **src** The string that is being converted.

PARAMETER **locale_name** The locale to use for the comparison

---

FUNCTION LocaleToUpperCase

Uni \

unicode	**LocaleToUpperCase**
`(unicode src, varstring locale_name)`	

Returns the upper case variant of the string using the rules for a particular locale.

PARAMETER **src** The string that is being converted.

PARAMETER **locale_name** The locale to use for the comparison

---

FUNCTION LocaleToTitleCase

Uni \

unicode	**LocaleToTitleCase**
`(unicode src, varstring locale_name)`	

Returns the upper case variant of the string using the rules for a particular locale.

PARAMETER **src** The string that is being converted.

PARAMETER **locale_name** The locale to use for the comparison

---

FUNCTION CompareIgnoreCase

Uni \

integer4	**CompareIgnoreCase**
`(unicode src1, unicode src2)`	

Compares the two strings case insensitively. Equivalent to comparing at strength 2.

PARAMETER **src1** The first string to be compared.

PARAMETER **src2** The second string to be compared.

SEE Std.Uni.CompareAtStrength

FUNCTION CompareAtStrength

Uni \

integer4	**CompareAtStrength**
`(unicode src1, unicode src2, integer1 strength)`	

Compares the two strings case insensitively. Equivalent to comparing at strength 2.

PARAMETER **src1** The first string to be compared.

PARAMETER **src2** The second string to be compared.

PARAMETER **strength** The strength of the comparison 1 ignores accents and case, differentiating only between letters 2 ignores case but differentiates between accents. 3 differentiates between accents and case but ignores e.g. differences between Hiragana and Katakana 4 differentiates between accents and case and e.g. Hiragana/Katakana, but ignores e.g. Hebrew cantellation marks 5 differentiates between all strings whose canonically decomposed forms (NFDNormalization Form D) are non-identical

SEE Std.Uni.CompareAtStrength

FUNCTION LocaleCompareIgnoreCase

Uni \

integer4	**LocaleCompareIgnoreCase**
`(unicode src1, unicode src2, varstring locale_name)`	

Compares the two strings case insensitively. Equivalent to comparing at strength 2.

PARAMETER **src1** The first string to be compared.

PARAMETER **src2** The second string to be compared.

PARAMETER **locale_name** The locale to use for the comparison

SEE Std.Uni.CompareAtStrength

FUNCTION LocaleCompareAtStrength

Uni \

integer4	**LocaleCompareAtStrength**
`(unicode src1, unicode src2, varstring locale_name, integer1 strength)`	

Compares the two strings case insensitively. Equivalent to comparing at strength 2.

PARAMETER **src1** The first string to be compared.

PARAMETER **src2** The second string to be compared.

PARAMETER **locale_name** The locale to use for the comparison

PARAMETER **strength** The strength of the comparison 1 ignores accents and case, differentiating only between letters 2 ignores case but differentiates between accents. 3 differentiates between accents and case but ignores e.g. differences between Hiragana and Katakana 4 differentiates between accents and case and e.g. Hiragana/Katakana, but ignores e.g. Hebrew cantellation marks 5 differentiates between all strings whose canonically decomposed forms (NFDNormalization Form D) are non-identical

---

FUNCTION Reverse

Uni \

unicode	**Reverse**
`(unicode src)`	

Returns the argument string with all characters in reverse order. Note the argument is not TRIMMED before it is reversed.

PARAMETER **src** The string that is being reversed.

FUNCTION FindReplace

Uni \

unicode	**FindReplace**
`(unicode src, unicode sought, unicode replacement)`	

Returns the source string with the replacement string substituted for all instances of the search string.

PARAMETER **src** The string that is being transformed.

PARAMETER **sought** The string to be replaced.

PARAMETER **replacement** The string to be substituted into the result.

FUNCTION LocaleFindReplace

Uni \

unicode	LocaleFindReplace
(unicode src, unicode sought, unicode replacement, varstring locale_name)	

Returns the source string with the replacement string substituted for all instances of the search string.

PARAMETER **src** The string that is being transformed.

PARAMETER **sought** The string to be replaced.

PARAMETER **replacement** The string to be substituted into the result.

PARAMETER **locale_name** The locale to use for the comparison

---

FUNCTION LocaleFindAtStrengthReplace

Uni \

unicode	LocaleFindAtStrengthReplace
(unicode src, unicode sought, unicode replacement, varstring locale_name, integer1 strength)	

Returns the source string with the replacement string substituted for all instances of the search string.

PARAMETER **src** The string that is being transformed.

PARAMETER **sought** The string to be replaced.

PARAMETER **replacement** The string to be substituted into the result.

PARAMETER **locale_name** The locale to use for the comparison

PARAMETER **strength** The strength of the comparison

FUNCTION CleanAccents

Uni \

unicode	**CleanAccents**
(unicode src)	

Returns the source string with all accented characters replaced with unaccented.

PARAMETER **src** The string that is being transformed.

FUNCTION CleanSpaces

Uni \

unicode	**CleanSpaces**
(unicode src)	

Returns the source string with all instances of multiple adjacent space characters (2 or more spaces together) reduced to a single space character. Leading and trailing spaces are removed, and tab characters are converted to spaces.

PARAMETER **src** The string to be cleaned.

FUNCTION WildMatch

Uni \

boolean	**WildMatch**
(unicode src, unicode _pattern, boolean _noCase)	

Tests if the search string matches the pattern. The pattern can contain wildcards '?' (single character) and '*' (multiple character).

PARAMETER **src** The string that is being tested.

PARAMETER **pattern** The pattern to match against.

PARAMETER **ignore_case** Whether to ignore differences in case between characters

---

FUNCTION Contains

Uni \

BOOLEAN	Contains
(unicode src, unicode _pattern, boolean _noCase)	

Tests if the search string contains each of the characters in the pattern. If the pattern contains duplicate characters those characters will match once for each occurence in the pattern.

PARAMETER **src** The string that is being tested.

PARAMETER **pattern** The pattern to match against.

PARAMETER **ignore_case** Whether to ignore differences in case between characters

---

FUNCTION EditDistance

Uni \

UNSIGNED4	EditDistance
(unicode _left, unicode _right, varstring localename = ")	

Returns the minimum edit distance between the two strings. An insert change or delete counts as a single edit. The two strings are trimmed before comparing.

PARAMETER **_left** The first string to be compared.

PARAMETER **_right** The second string to be compared.

PARAMETER **localname** The locale to use for the comparison. Defaults to ".

RETURN The minimum edit distance between the two strings.

FUNCTION EditDistanceWithinRadius

Uni \

BOOLEAN	**EditDistanceWithinRadius**
(unicode _left, unicode _right, unsigned4 radius, varstring localename = '')	

Returns true if the minimum edit distance between the two strings is with a specific range. The two strings are trimmed before comparing.

PARAMETER **__left** The first string to be compared.

PARAMETER **__right** The second string to be compared.

PARAMETER **radius** The maximum edit distance that is accepable.

PARAMETER **localname** The locale to use for the comparison. Defaults to ''.

RETURN Whether or not the two strings are within the given specified edit distance.

FUNCTION WordCount

Uni \

unsigned4	**WordCount**
(unicode text, varstring localename = '')	

Returns the number of words in the string. Word boundaries are marked by the unicode break semantics.

PARAMETER **text** The string to be broken into words.

PARAMETER **localname** The locale to use for the break semantics. Defaults to ''.

RETURN The number of words in the string.

FUNCTION GetNthWord

Uni \

unicode	**GetNthWord**
(unicode text, unsigned4 n, varstring localename = '')	

Returns the n-th word from the string. Word boundaries are marked by the unicode break semantics.

PARAMETER **text** The string to be broken into words.

PARAMETER **n** Which word should be returned from the function.

PARAMETER **localname** The locale to use for the break semantics. Defaults to ''.

RETURN The number of words in the string.

---

system

Go Up

Table of Contents