### Logical Files

- CompareFiles ......................................................... 8
- DeleteLogicalFile .................................................. 9
- LogicalFileList .................................................... 10
- GetNoCommonDefault ............................................... 12
- FileExists .......................................................... 13
- ForeignLogicalFileName ........................................... 14
- GetFileDescription ................................................ 15
- GetLogicalFileAttribute ......................................... 16
- ProtectLogicalFile .................................................. 17
- RenameLogicalFile ................................................ 18
- SetFileDescription ................................................ 19
- SetReadOnly .......................................................... 20
- VerifyFile ............................................................. 21

### SuperFiles

- CreateSuperFile .................................................... 22
- SuperFileExists ..................................................... 23
- DeleteSuperFile ..................................................... 24
- GetSuperFileSubCount .............................................. 25
- GetSuperFileSubName ............................................... 26
- LogicalFileSuperOwners ......................................... 27
- LogicalFileSuperSubList ......................................... 28
- SuperFileContents .................................................. 29
- FindSuperFileSubName ............................................. 30
- StartSuperFileTransaction ....................................... 31
- AddSuperFile ......................................................... 32
- RemoveSuperFile .................................................... 33
- ClearSuperFile ....................................................... 34
- RemoveOwnedSubFiles ............................................. 35
- SwapSuperFile ........................................................ 36
- ReplaceSuperFile .................................................... 37
- PromoteSuperFileList .............................................. 38
- FinishSuperFileTransaction ...................................... 39

### External Files

- ExternalLogicalFileName .......................................... 40
- MoveExternalFile .................................................... 41
- DeleteExternalFile .................................................. 42
- CreateExternalDirectory .......................................... 43
- RemoteDirectory ..................................................... 44

### File Browsing

- SetColumnMapping ................................................ 45
- GetColumnMapping ................................................ 46
- AddFileRelationship ............................................... 47
- FileRelationshipList ............................................... 48
- RemoveFileRelationship .......................................... 49

### File Movement

- DfuPlusExec ........................................................ 50
- AbortDfuWorkunit .................................................. 51
- Copy ................................................................. 52
- DeSpray ............................................................. 53
- RemotePull .......................................................... 54
- Replicate ............................................................ 55
- SprayFixed .......................................................... 56
- SprayDelimited / SprayVariable .................................. 57

© 2024 HPCC Systems®. All rights reserved
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SprayXML</td>
<td>68</td>
</tr>
<tr>
<td>SprayJson</td>
<td>70</td>
</tr>
<tr>
<td>WaitDfuWorkunit</td>
<td>72</td>
</tr>
<tr>
<td>SetExpireDays</td>
<td>73</td>
</tr>
<tr>
<td>GetExpireDays</td>
<td>74</td>
</tr>
<tr>
<td>ClearExpireDays</td>
<td>75</td>
</tr>
<tr>
<td><strong>String Handling</strong></td>
<td></td>
</tr>
<tr>
<td>CleanAccents</td>
<td>76</td>
</tr>
<tr>
<td>CleanSpaces</td>
<td>77</td>
</tr>
<tr>
<td>CompareAtStrength</td>
<td>78</td>
</tr>
<tr>
<td>CompareIgnoreCase</td>
<td>79</td>
</tr>
<tr>
<td>Contains</td>
<td>80</td>
</tr>
<tr>
<td>CountWords</td>
<td>81</td>
</tr>
<tr>
<td>DecodeBase64</td>
<td>82</td>
</tr>
<tr>
<td>EditDistance</td>
<td>83</td>
</tr>
<tr>
<td>EditDistanceWithinRadius</td>
<td>84</td>
</tr>
<tr>
<td>EncodeBase64</td>
<td>85</td>
</tr>
<tr>
<td>EndsWith</td>
<td>86</td>
</tr>
<tr>
<td>EquallgnoreCase</td>
<td>87</td>
</tr>
<tr>
<td>ExcludeFirstWord</td>
<td>88</td>
</tr>
<tr>
<td>ExcludeLastWord</td>
<td>89</td>
</tr>
<tr>
<td>ExcludeNthWord</td>
<td>90</td>
</tr>
<tr>
<td>Extract</td>
<td>91</td>
</tr>
<tr>
<td>ExtractMultiple</td>
<td>92</td>
</tr>
<tr>
<td>Filter</td>
<td>93</td>
</tr>
<tr>
<td>FilterOut</td>
<td>94</td>
</tr>
<tr>
<td>Find</td>
<td>95</td>
</tr>
<tr>
<td>FindCount</td>
<td>96</td>
</tr>
<tr>
<td>FindAtStrength</td>
<td>97</td>
</tr>
<tr>
<td>FindAtStrengthReplace</td>
<td>98</td>
</tr>
<tr>
<td>FindReplace</td>
<td>99</td>
</tr>
<tr>
<td>FindWord</td>
<td>100</td>
</tr>
<tr>
<td>FromHexPairs</td>
<td>101</td>
</tr>
<tr>
<td>GetNthWord</td>
<td>102</td>
</tr>
<tr>
<td>RemoveSuffix</td>
<td>103</td>
</tr>
<tr>
<td>Repeat</td>
<td>104</td>
</tr>
<tr>
<td>Reverse</td>
<td>105</td>
</tr>
<tr>
<td>SplitWords</td>
<td>106</td>
</tr>
<tr>
<td>SubstituteExcluded</td>
<td>107</td>
</tr>
<tr>
<td>SubstituteIncluded</td>
<td>108</td>
</tr>
<tr>
<td>StartsWith</td>
<td>109</td>
</tr>
<tr>
<td>ToHexPairs</td>
<td>110</td>
</tr>
<tr>
<td>ToLowerCase</td>
<td>111</td>
</tr>
<tr>
<td>ToLowerCase</td>
<td>112</td>
</tr>
<tr>
<td>ToTitleCase</td>
<td>113</td>
</tr>
<tr>
<td>ToUpperCase</td>
<td>114</td>
</tr>
<tr>
<td>Translate</td>
<td>115</td>
</tr>
<tr>
<td>Version</td>
<td>116</td>
</tr>
<tr>
<td>WildMatch</td>
<td>117</td>
</tr>
<tr>
<td>WordCount</td>
<td>118</td>
</tr>
<tr>
<td><strong>Metaphone Support</strong></td>
<td></td>
</tr>
<tr>
<td>Primary</td>
<td>119</td>
</tr>
<tr>
<td>Secondary</td>
<td>120</td>
</tr>
<tr>
<td>Double</td>
<td>121</td>
</tr>
<tr>
<td><strong>Cryptography Support</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>122</td>
</tr>
<tr>
<td></td>
<td>123</td>
</tr>
</tbody>
</table>
Auditing

Cluster Handling

Job Handling

File Monitoring

Logging

dbglog

addWorkunitInformation

addWorkunitWarning

addWorkunitError

getGlobalId

getLocalId

generateGloballyUniqueID

getElapsedMs

Auditing

Audit

Utilities

GetHostName

ResolveHostName

GetUniqueInteger

GetEspUrl

PlatformVersionCheck

Debugging

GetParseTree

GetXMLParseTree

Sleep

msTick

Email

SendEmail

SendEmailAttachData
## Workunit Services
- WorkunitExists ................................................. 244
- WorkunitList ..................................................... 245
- SetWorkunitAppValue .............................................. 247
- WUIDonDate ....................................................... 248
- WUIDaysAgo ....................................................... 249
- WorkunitTimeStamps .............................................. 250
- WorkunitMessages ............................................... 251
- WorkunitFilesRead .............................................. 252
- WorkunitFilesWritten ......................................... 253
- WorkunitTimings ................................................ 254

## BLAS Support

### Types

- ICellFunc ....................................................... 257
- Apply2Cells ..................................................... 258
- dasum ............................................................. 259
- daxpy ............................................................ 260
- dgemm ............................................................ 261
- dgetf2 ............................................................ 262
- dpotf2 ............................................................ 263
- dscal ............................................................. 264
- dsyrk .............................................................. 265
- dtrsm ............................................................. 266
- extract_diag ..................................................... 267
- extract_tri ....................................................... 268
- make_diag ......................................................... 269
- make_vector ...................................................... 270
- trace .............................................................. 271

### Math Support

- Infinity .......................................................... 272
- NaN ................................................................. 273
- isInfinity ........................................................ 274
- isnaN .............................................................. 275
- isFinite .......................................................... 276
- FMod .............................................................. 277
- FMatch ............................................................ 278
**CompareFiles**

STD.File.CompareFiles(  
  file1, file2 [ , logicalonly ] [ , usecrcs ] )

**file1**
A null-terminated string containing the logical name of the first file.

**file2**
A null-terminated string containing the logical name of the second file.

**logicalonly**
Optional. A boolean TRUE/FALSE flag that, when TRUE, does not compare physical information from disk but only the logical information in the system datastore (Dali). If omitted, the default is TRUE.

**usecrcs**
Optional. A boolean TRUE/FALSE flag indicating that, when TRUE, compares physical CRCs of all the parts on disk. This may be slow on large files. If omitted, the default is FALSE.

**Return:**
CompareFiles returns an INTEGER4 value.

The **CompareFiles** function compares *file1* against *file2* and returns the following values:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><em>file1</em> and <em>file2</em> match exactly</td>
</tr>
<tr>
<td>1</td>
<td><em>file1</em> and <em>file2</em> contents match, but <em>file1</em> is newer than <em>file2</em></td>
</tr>
<tr>
<td>-1</td>
<td><em>file1</em> and <em>file2</em> contents match, but <em>file2</em> is newer than <em>file1</em></td>
</tr>
<tr>
<td>2</td>
<td><em>file1</em> and <em>file2</em> contents do not match and <em>file1</em> is newer than <em>file2</em></td>
</tr>
<tr>
<td>-2</td>
<td><em>file1</em> and <em>file2</em> contents do not match and <em>file2</em> is newer than <em>file1</em></td>
</tr>
</tbody>
</table>

**Example:**

```plaintext
A := STD.File.CompareFiles('Fred1', 'Fred2');
```
DeleteLogicalFile

STD.File.DeleteLogicalFile( filename [, ifexists ] )

<table>
<thead>
<tr>
<th>filename</th>
<th>A null-terminated string containing the logical name of the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifexists</td>
<td>Optional. A boolean value indicating whether to post an error if the filename does not exist. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>

The **DeleteLogicalFile** function removes the named file from disk.

Example:

```plaintext
A := STD.File.DeleteLogicalFile('Fred');
```
LogicalFileList

STD.File.LogicalFileList( [ pattern ] [ , includenormal ] [ , includesuper ] [ , unknownszero ] [ , foreigndali ] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pattern</td>
<td>Optional. A null-terminated string containing the mask of the files to list. If omitted, the default is &quot;*&quot; (all files).</td>
</tr>
<tr>
<td>includenormal</td>
<td>Optional. A boolean flag indicating whether to include &quot;normal&quot; files. If omitted, the default is TRUE.</td>
</tr>
<tr>
<td>includesuper</td>
<td>Optional. A boolean flag indicating whether to include SuperFiles. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>unknownszero</td>
<td>Optional. A boolean flag indicating to set file sizes that are unknown to zero (0) instead of minus-one (-1). If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>foreigndali</td>
<td>Optional. The IP address of the foreign dali used to resolve the file. If blank then the file is resolved locally. If omitted, the default is blank.</td>
</tr>
</tbody>
</table>

Return: LogicalFileList returns a dataset in the following format:

```
EXPORT FsLogicalFileNameRecord := RECORD
  STRING name;
END;

EXPORT FsLogicalFileInfoRecord := RECORD(FsLogicalFileNameRecord)
  BOOLEAN superfile;
  UNSIGNED8 size;
  UNSIGNED8 rowcount;
  STRING19 modified;
  STRING owner;
  STRING cluster;
END;
```

The LogicalFileList function returns a list of the logical files in the environment files as a dataset in the format listed above.

Example:

```
OUTPUT(STD.File.LogicalFileList());
  //returns all normal files

OUTPUT(STD.File.LogicalFileList(FALSE,TRUE));
  //returns all SuperFiles
```
GetNoCommonDefault

STD.File.GetNoCommonDefault()

Return: GetNoCommonDefault returns a BOOLEAN value

The `GetNoCommonDefault` function returns the boolean value of the 'noCommon' property in the system configuration if it is defined. Otherwise it returns 'true' as the default.

Example:

```
IMPORT STD;
A := STD.File.GetNoCommonDefault();
```

See Also: SprayFixed, SprayXML, SprayJSON, SprayDelimited
FileExists

STD.File.FileExists( filename [, physicalcheck ] )

<table>
<thead>
<tr>
<th>filename</th>
<th>A null-terminated string containing the logical name of the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>physicalcheck</td>
<td>Optional. A boolean TRUE/FALSE to indicate whether to check for the physical existence the filename on disk. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>

Return: FileExists returns a BOOLEAN value.

The FileExists function returns TRUE if the specified filename is present in the Distributed File Utility (DFU). If physicalcheck is set to TRUE, then the file's physical presence on disk is also checked.

Example:

A := STD.File.FileExists('~CLASS::RT::IN::People');

See Also: SuperFileExists
ForeignLogicalFileName


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>A null-terminated string containing the logical name of the file.</td>
</tr>
<tr>
<td>foreigndali</td>
<td>A null-terminated string containing the IP address of the foreign Dali. If omitted, the filename is presumed to be a foreign logical file name, which is converted to a local logical file name.</td>
</tr>
<tr>
<td>absolutepath</td>
<td>Optional. A boolean TRUE/FALSE to indicate whether to prepend a tilde (~) to the resulting foreign logical file name. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>omitClusterPrefix</td>
<td>Optional. A boolean TRUE/FALSE to indicate whether the target cluster's prefix should automatically be added if 'filename' is a relative logical file name. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>

Return: ForeignLogicalFileName returns a VARSTRING (null-terminated) value.

The ForeignLogicalFileName function returns either a foreign logical file name (if the foreigndali parameter is present) or a local logical file name.

Example:

```haskell
sf := '~thor_data400::BASE::Business_Header';
ff := STD.File.ForeignLogicalFileName(sf,'10.150.29.161',true);
//results in: ~foreign::10.150.29.161::thor_data400::base::business_header
lf := STD.File.ForeignLogicalFileName(ff,'',true);
//results in: ~thor_data400::base::business_header
```
GetFileDescription

STD.File.GetFileDescription(filename)

<table>
<thead>
<tr>
<th>filename</th>
<th>A null-terminated string containing the logical name of the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>GetFileDescription returns a VARSTRING (null-terminated) value.</td>
</tr>
</tbody>
</table>

The GetFileDescription function returns a string containing the description information stored by the DFU about the specified filename. This description is set either through ECL watch or by using the STD.File.SetFileDescription function.

Example:

A := STD.File.GetFileDescription('Fred');
GetLogicalFileAttribute

STD.File.GetLogicalFileAttribute(logicalfilename, attrname)

<table>
<thead>
<tr>
<th>logicalfilename</th>
<th>A null-terminated string containing the name of the logical file as it is known by the DFU.</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrname</td>
<td>A null-terminated string containing the name of the file attribute to return. Possible values are recordSize, recordCount, size, clusterName, directory, owner, description, ECL, partmask, numparts, name, modified, format, job, checkSum, kind, csvSeparate, csvTerminate, csvEscape, headerLength, footerLength, rowTag, workunit, accessed, expireDays, maxRecordSize, csvQuote, blockCompressed, compressedSize, fileCrc, formatCrc, or protected. The value is case-sensitive.</td>
</tr>
</tbody>
</table>

Return: GetLogicalFileAttribute returns returns a VARSTRING (null-terminated) value.

The GetLogicalFileAttribute function returns the value of the attrname for the specified logicalfilename.

Example:

```
IMPORT STD;

file := '~ certification::full_test_distributed';

OUTPUT(STD.File.GetLogicalFileAttribute(file,'recordSize'),NAMED('recordSize'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'recordCount'),NAMED('recordCount'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'size'),NAMED('size'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'clusterName'),NAMED('clusterName'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'directory'),NAMED('directory'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'numparts'),NAMED('numparts'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'owner'),NAMED('owner'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'description'),NAMED('description'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'ECL'),NAMED('ECL'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'partmask'),NAMED('partmask'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'name'),NAMED('name'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'modified'),NAMED('modified'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'protected'),NAMED('protected'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'format'),NAMED('format'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'job'),NAMED('job'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'checkSum'),NAMED('checkSum'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'kind'),NAMED('kind'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'csvSeparate'),NAMED('csvSeparate'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'csvTerminate'),NAMED('csvTerminate'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'csvEscape'),NAMED('csvEscape'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'headerLength'),NAMED('headerLength'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'footerLength'),NAMED('footerLength'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'rowTag'),NAMED('rowTag'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'workunit'),NAMED('workunit'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'accessed'),NAMED('accessed'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'expireDays'),NAMED('expireDays'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'maxRecordSize'),NAMED('maxRecordSize'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'csvQuote'),NAMED('csvQuote'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'blockCompressed'),NAMED('blockCompressed'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'compressedSize'),NAMED('compressedSize'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'fileCrc'),NAMED('fileCrc'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'formatCrc'),NAMED('formatCrc'));
```
ProtectLogicalFile

STD.File.ProtectLogicalFile( logicalfilename [, value ])

<table>
<thead>
<tr>
<th>logicalfilename</th>
<th>A null-terminated string containing the name of the logical file as it is known by the DFU.</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Optional. A boolean flag indicating whether to protect or un-protect the file. If omitted, the default is TRUE.</td>
</tr>
</tbody>
</table>

The **ProtectLogicalFile** function toggles protection on and off for the specified `logicalfilename`.

Example:

```plaintext
IMPORT STD;
file := '~class::bmf::join::halfkeyed';

STD.File.ProtectLogicalFile(file);       //protect
STD.File.ProtectLogicalFile(file, FALSE); //unprotect
```
RenameLogicalFile

STD.File.RenameLogicalFile( filename, newname, [allowOverwrite] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>A null-terminated string containing the current logical name of the file.</td>
</tr>
<tr>
<td>newname</td>
<td>A null-terminated string containing the new logical name for the file.</td>
</tr>
<tr>
<td>allowOverwrite</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to allow the renamed file to overwrite an existing file of the same name. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>

The `RenameLogicalFile` function changes the logical `filename` to the `newname`.

Example:

```python
A := STD.File.RenameLogicalFile('Fred', 'Freddie');
```
SetFileDescription

**STD.File.SetFileDescription** ( *filename*, *value* )

<table>
<thead>
<tr>
<th><em>filename</em></th>
<th>A null-terminated string containing the logical name of the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>value</em></td>
<td>A null-terminated string containing the description to place on the file.</td>
</tr>
</tbody>
</table>

The **SetFileDescription** function changes the description information stored by the DFU about the specified *filename* to the specified *value*. This description is seen either through ECL watch or by using the STD.File.GetFileDescription function.

Example:

```ecl
A := STD.File.SetFileDescription('Fred','All the Freds in the world');
```
SetReadOnly

STD.File.SetReadOnly( filename , flag )

<table>
<thead>
<tr>
<th>filename</th>
<th>A null-terminated string containing the logical name of the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>flag</td>
<td>A boolean value indicating which way to set the read-only attribute of the filename.</td>
</tr>
</tbody>
</table>

The **SetReadOnly** function toggles the read-only attribute of the filename. If the flag is TRUE, read-only is set on.

Example:

```plaintext
A := STD.File.SetReadOnly('Fred',TRUE);
//set read only flag on
```
VerifyFile

STD.File.VerifyFile(  file, usecrcs  )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file</td>
<td>A null-terminated string containing the logical name of the file.</td>
</tr>
<tr>
<td>usecrcs</td>
<td>A boolean TRUE/FALSE flag indicating that, when TRUE, compares physical CRCs of all the parts on disk. This may be slow on large files.</td>
</tr>
</tbody>
</table>

Return: VerifyFile returns returns a VARSTRING value.

The VerifyFile function checks the system datastore (Dali) information for the file against the physical parts on disk and returns the following values:

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>The file parts match the datastore information</td>
</tr>
<tr>
<td>Could not find file:</td>
<td>The logical filename was not found</td>
</tr>
<tr>
<td>Could not find part file:</td>
<td>The partname was not found</td>
</tr>
<tr>
<td>Modified time differs for:</td>
<td>The partname has a different timestamp</td>
</tr>
<tr>
<td>File size differs for:</td>
<td>The partname has a file size</td>
</tr>
<tr>
<td>File CRC differs for:</td>
<td>The partname has a different CRC</td>
</tr>
</tbody>
</table>

Example:

```
A := STD.File.VerifyFile('Fred1', TRUE);
```
CreateSuperFile

STD.File.CreateSuperFile( superfile [, sequentialparts ] [, allowExist ] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>superfile</td>
<td>A null-terminated string containing the logical name of the superfile.</td>
</tr>
<tr>
<td>sequentialparts</td>
<td>Optional. A boolean value indicating whether the sub-files must be sequentially ordered. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>allowExist</td>
<td>Optional. A boolean value indicating whether to post an error if the superfile already exists. If TRUE, no error is posted. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>

Return: Null.

The CreateSuperFile function creates an empty superfile. This function is not included in a superfile transaction.

The sequentialparts parameter set to TRUE governs the unusual case where the logical numbering of sub-files must be sequential (for example, where all sub-files are already globally sorted). With sequentialparts FALSE (the default) the subfile parts are interleaved so the parts are found locally.

For example, if on a 4-way cluster there are 3 files (A, B, and C) then the parts are as follows:

A._1_of_4, B._1_of_4, and C._1_of_4 are on node 1

A._2_of_4, B._2_of_4, and C._2_of_4 are on node 2

A._3_of_4, B._3_of_4, and C._3_of_4 are on node 3

A._4_of_4, B._4_of_4, and C._4_of_4 are on node 4

Reading the superfile created with sequentialparts FALSE on Thor will read the parts in the order:

[A1,B1,C1,] [A2,B2,C2,] [A3,B3,C3,] [A4,B4,C4]

so the reads will all be local (i.e., A1,B1,C1 on node 1 etc). Setting sequentialparts to TRUE will read the parts in subfile order, like this:

[A1,A2,A3,] [A4,B1,B2,] [B3,B4,C1,] [C2,C3,C4]

so that the global order of A,B,C,D is maintained. However, the parts cannot all be read locally (e.g., A2 and A3 will be read on part 1). Because of this it is much less efficient to set sequentialparts true, and as it is unusual anyway to have files that are partitioned in order, it becomes a very unusual option to set.

Example:

```c
STD.File.CreateSuperFile('~CLASS::RT::IN::SF1',,1);
//This is the same but uses named parameter
STD.File.CreateSuperFile('~CLASS::RT::IN::SF1',allowExist := 1);
```
SuperFileExists

STD.File.SuperFileExists( filename )

<table>
<thead>
<tr>
<th>filename</th>
<th>A null-terminated string containing the logical name of the superfile.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>SuperFileExists returns a BOOLEAN value.</td>
</tr>
</tbody>
</table>

The **SuperFileExists** function returns TRUE if the specified *filename* is present in the Distributed File Utility (DFU) and is a SuperFile. It returns FALSE if the *filename* does exist but it is not a SuperFile (in other words, it is a normal DATASET. Use the STD.File.FileExists function to detect their presence or absence).

This function is not included in a superfile transaction.

Example:

```plaintext
A := STD.File.SuperFileExists('~CLASS::RT::IN::SF1');
```

See Also: FileExists
DeleteSuperFile

STD.File.DeleteSuperFile( superName [, deletesub ] )

<table>
<thead>
<tr>
<th>superName</th>
<th>A null-terminated string containing the logical name of the superfile.</th>
</tr>
</thead>
<tbody>
<tr>
<td>deletesub</td>
<td>A boolean value indicating whether to delete the subfiles. If omitted, the default is FALSE. This option should not be used if the superfile contains any foreign file or foreign superfile.</td>
</tr>
</tbody>
</table>

Return: Null.

The DeleteSuperFile function deletes the superName superfile.

This function is not included in a superfile transaction.

Example:

`STD.File.DeleteSuperFile('~CLASS::RT::IN::SF1');`
GetSuperFileSubCount

STD.File.GetSuperFileSubCount( superfile )

<table>
<thead>
<tr>
<th>superfile</th>
<th>A null-terminated string containing the logical name of the superfile.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>GetSuperFileSubCount returns an INTEGER4 value.</td>
</tr>
</tbody>
</table>

The **GetSuperFileSubCount** function returns the number of sub-files comprising the *superfile*.

This function is not included in a superfile transaction.

Example:

```plaintext
A := STD.File.GetSuperFileSubCount('~CLASS::RT::IN::SF1');
```
## GetSuperFileSubName

\[ \text{STD.File.GetSuperFileSubName( superfile, subfile [, absolutepath ] )} \]

<table>
<thead>
<tr>
<th>superfile</th>
<th>A null-terminated string containing the logical name of the superfile.</th>
</tr>
</thead>
<tbody>
<tr>
<td>subfile</td>
<td>An integer in the range of one (1) to the total number of sub-files in the superfile specifying the ordinal position of the sub-file whose name to return.</td>
</tr>
<tr>
<td>absolutepath</td>
<td>Optional. A boolean TRUE/FALSE to indicate whether to prepend a tilde (~) to the resulting foreign logical file name. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>

Return: GetSuperFileSubName returns a VARSTRING value.

The **GetSuperFileSubName** function returns the logical name of the specified `subfile` in the `superfile`.

This function is not included in a superfile transaction.

**Example:**

```plaintext
A := STD.File.GetSuperFileSubName('~CLASS::RT::IN::SF1', 1); //get name of first sub-file

//this example gets the name of the first sub-file in
// a foreign superfile
sf := '~thor_data400::BASE::Business_Header';

sub := STD.File.GetSuperFileSubName( STD.File.ForeignLogicalFileName (sf, '10.150.29.161', TRUE), 1, TRUE);

OUTPUT(STD.File.ForeignLogicalFileName(sub,''));
```
LogicalFileSuperOwners

STD.File.LogicalFileSuperOwners( filename )

| filename | A null-terminated string containing the logical name of the file. |
| Return: | LogicalFileSuperOwners returns a dataset in the following format: |

```
EXPORT FsLogicalFileNameRecord := RECORD
  STRING name;
END;
```

The **LogicalFileSuperOwners** function returns a list of the logical filenames of all the SuperFiles that contain the *filename* as a sub-file.

This function is not included in a superfile transaction.

Example:

```
OUTPUT(STD.File.LogicalFileSuperowners('~CLASS::RT::SF::Daily1'));
  //returns all SuperFiles that "own" the Daily1 file
```
LogicalFileSuperSubList

STD.File.LogicalFileSuperSubList()

Return: LogicalFileSuperSubList returns a dataset in the following format:

```EXPORT FsLogicalSuperSubRecord := RECORD
  STRING supername{MAXLENGTH(255)};
  STRING subname{MAXLENGTH(255)};
END;```

The **LogicalFileSuperSubList** function returns a list of the logical filenames of all the SuperFiles and their component sub-files.

This function is not included in a superfile transaction.

Example:

```OUTPUT(STD.File.LogicalFileSuperSubList());
//returns all SuperFiles and their sub-files```
SuperFileContents

STD.File.SuperFileContents( filename [, recurse ])

| filename | A null-terminated string containing the logical name of the SuperFile. |
| recurse  | A boolean flag indicating whether to expand nested SuperFiles within the filename so that only logical files are returned. If omitted, the default is FALSE. |

Return: SuperFileContents returns a dataset in the following format:

```
EXPORT FsLogicalFileNameRecord := RECORD
  STRING name;
END;
```

The `SuperFileContents` function returns a list of the logical filenames of all the sub-files in the `filename`.

This function is not included in a superfile transaction.

Example:

```
OUTPUT(STD.File.SuperFileContents('~CLASS::RT::SF::Daily'));
//returns all files in the SuperFile
```
FindSuperFileSubName

STD.File.FindSuperFileSubName( superfile, subfile )

<table>
<thead>
<tr>
<th>superfile</th>
<th>A null-terminated string containing the logical name of the superfile.</th>
</tr>
</thead>
<tbody>
<tr>
<td>subfile</td>
<td>A null-terminated string containing the logical name of the sub-file.</td>
</tr>
<tr>
<td>Return:</td>
<td>FindSuperFileSubName returns an INTEGER4 value.</td>
</tr>
</tbody>
</table>

The FindSuperFileSubName function returns the ordinal position of the specified subfile in the superfile.

This function is not included in a superfile transaction.

Example:

A := STD.File.FindSuperFileSubName('~CLASS::SF1', '~CLASS::Sue');  //get position of
//sub-file '~CLASS::Sue'
The `StartSuperFileTransaction` function begins a transaction frame for superfile maintenance. The transaction frame is terminated by calling the `FinishSuperFileTransaction` function. Within the transaction frame, multiple superfiles may be maintained by using SuperFile Maintenance functions to add, remove, clear, swap, and replace sub-files.

You must use the SEQUENTIAL action to ensure ordered execution of the function calls within the transaction frame. This way, the SuperFile Maintenance functions are called in the order that they are listed between the transaction frame’s start and finish functions, but they are only committed once (i.e., actually executed) at the finish of the transaction function.

The first SuperFile Maintenance function called within the transaction frame initiates a “read” lock on the superfile until the commit. At commit, the superfile is “write” locked for the transaction to actually execute, and all locks are released at the end of the commit. It is important to note that any calls to functions other than SuperFile Maintenance functions within the transaction frame are not part of the transaction frame (even though they are executed in the order written). The “read” lock is only generated when the first SuperFile Maintenance function is called. While the superfile is “read” locked, no concurrent “write” locks can modify the superfile.

During the timeframe of the “write” lock at commit (usually a small time window), no concurrent “read” locks are allowed. Therefore, the SuperFile Maintenance functions must be called within a transaction frame to avoid the possibility of another process may try to modify the superfile during sub-file maintenance. As a result, maintenance work can be accomplished without causing problems with any query that might use the SuperFile.

The `FinishSuperFileTransaction` function does an automatic rollback of the transaction if any error or failure occurs during the transaction frame. If no error occurs, then the commit or rollback of the transaction is controlled by the `rollback` parameter to the `FinishSuperFileTransaction` function.

Example:

```plaintext
IMPORT STD;
WeeklyRollup:='~Training::Examples::WeeklyRollup';
WeeklySF  :='~Training::Examples::Weekly';
DailySF    :='~Training::Examples::Daily';

DailyDS := DATASET(DailySF,{string Myfield},THOR);

SEQUENTIAL(STD.File.StartSuperFileTransaction(),
            STD.File.ClearSuperFile(DailySF),
            OUTPUT(DailyDS,WeeklyRollup),
            STD.File.AddSuperFile(WeeklySF,WeeklyRollup),
            STD.File.FinishSuperFileTransaction());
//executes the OUTPUT after a "read" lock on the superfile DailySF
//has been initiated by the ClearSuperFile Maintenance function,
//which in turn executes only at the FinishTransaction
```
AddSuperFile

STD.File.AddSuperFile( superfile, subfile [ , atpos ] [ , addcontents ] [ ,strict ])

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>superfile</td>
<td>A null-terminated string containing the logical name of the superfile.</td>
</tr>
<tr>
<td>subfile</td>
<td>A null-terminated string containing the logical name of the sub-file. This may be another superfile.</td>
</tr>
<tr>
<td>atpos</td>
<td>An integer specifying the position of the subfile in the superfile. If omitted, the default is zero (0), which places the subfile at the end of the superfile.</td>
</tr>
<tr>
<td>addcontents</td>
<td>A boolean flag that, if set to TRUE, specifies the subfile is also a superfile and the contents of that superfile are added to the superfile rather than its reference. If omitted, the default is to add by reference (addcontents := FALSE).</td>
</tr>
<tr>
<td>strict</td>
<td>A boolean flag specifying, in the case of a subfile that is itself a superfile, whether to check for the existence of the superfile and raise an error if it does not. Also, if addcontents is set to TRUE, it will ensure the subfile that is itself a superfile is not empty. If omitted, the default is false.</td>
</tr>
</tbody>
</table>

Return: Null.

The AddSuperFile function adds the subfile to the list of files comprising the superfile. All subfiles in the superfile must have exactly the same structure type and format.

This function may be included in a superfile transaction, but is not required to be.

Example:

```hpcc
IMPORT STD;
SEQUENTIAL(
    STD.File.StartSuperFileTransaction(),
    STD.File.AddSuperFile('MySuperFile1','MySubFile1'),
    STD.File.AddSuperFile('MySuperFile1','MySubFile2'),
    STD.File.AddSuperFile('MySuperFile2','MySuperFile1'),
    STD.File.AddSuperFile('MySuperFile3','MySuperFile1',addcontents := true),
    STD.File.FinishSuperFileTransaction()
);
// MySuperFile1 contains { MySubFile1, MySubFile2 }
// MySuperFile2 contains { MySuperFile1 }
// MySuperFile3 contains { MySubFile1, MySubFile2 }
```
## RemoveSuperFile

The **RemoveSuperFile** function removes the *subfile* from the list of files comprising the *superfile*.

This function may be included in a superfile transaction.

### Example:

```plaintext
SEQUENTIAL(
    STD.File.StartSuperFileTransaction(),
    STD.File.RemoveSuperFile('MySuperFile','MySubFile'),
    STD.File.FinishSuperFileTransaction()
);
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>superfile</em></td>
<td>A null-terminated string containing the logical name of the superfile.</td>
</tr>
<tr>
<td><em>subfile</em></td>
<td>A null-terminated string containing the logical name of the sub-file. This may be another superfile or a foreign file or superfile.</td>
</tr>
<tr>
<td><em>del</em></td>
<td>A boolean flag specifying whether to delete the <em>subfile</em> from disk or just remove it from the <em>superfile</em> list of files. If omitted, the default is to just remove it from the <em>superfile</em> list of files. <strong>This option should not be used if the subfile is a foreign file or foreign superfile.</strong></td>
</tr>
<tr>
<td><em>removecontents</em></td>
<td>A boolean flag specifying whether the contents of a <em>subfile</em> that is itself a superfile are recursively removed.</td>
</tr>
</tbody>
</table>

### Return:

Null.
ClearSuperFile

STD.File.ClearSuperFile( superfile, [ , delete ] )

<table>
<thead>
<tr>
<th>superfile</th>
<th>A null-terminated string containing the logical name of the superfile.</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete</td>
<td>A boolean flag specifying whether to delete the sub-files from disk or just remove them from the superfile list of files. If omitted, the default is to just remove them from the superfile list of files.</td>
</tr>
</tbody>
</table>

Return: Null.

The ClearSuperFile function removes all sub-files from the list of files comprising the superfile.

This function may be included in a superfile transaction.

Example:

```plaintext
SEQUENTIAL(
    STD.File.StartSuperFileTransaction(),
    STD.File.ClearSuperFile('MySuperFile'),
    STD.File.FinishSuperFileTransaction()
);```

© 2024 HPCC Systems®. All rights reserved
RemoveOwnedSubFiles

STD.File.RemoveOwnedSubFiles(  superfile [ , delete ])

<table>
<thead>
<tr>
<th>superfile</th>
<th>A null-terminated string containing the logical name of the superfile.</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete</td>
<td>A boolean flag specifying to delete the sub-files from disk when TRUE or just remove them from the superfile list of files. If omitted, the default is to just remove them from the superfile list of files.</td>
</tr>
</tbody>
</table>

Return: Null.

The **RemoveOwnedSubFiles** function removes all owned sub-files from the specified superfile. These are only removed if they are solely owned by the superfile. If a subfile is co-owned, (i.e., a member of any other superfile), then the removal is ignored.

This function may be included in a superfile transaction, unless the delete Flag is TRUE.

Example:

```plaintext
SEQUENTIAL(
    STD.File.StartSuperFileTransaction(),
    STD.File.RemoveOwnedSubFiles('MySuperFile'),
    STD.File.FinishSuperFileTransaction()
);
```
SwapSuperFile

STD.File.SwapSuperFile(  superfile1, superfile2  )

| superfile1  | A null-terminated string containing the logical name of the superfile. |
| superfile2  | A null-terminated string containing the logical name of the superfile. |
| Return:     | Null. |

The SwapSuperFile function moves all sub-files from superfile1 to superfile2 and vice versa.

This function may be included in a superfile transaction.

Example:

```plaintext
SEQUENTIAL(
  STD.File.StartSuperFileTransaction(),
  STD.File.SwapSuperFile('MySuperFile','YourSuperFile'),
  STD.File.FinishSuperFileTransaction()
);
```
ReplaceSuperFile

STD.File.ReplaceSuperFile(  superfile, subfile1 , subfile2  )

<table>
<thead>
<tr>
<th>superfile</th>
<th>A null-terminated string containing the logical name of the superfile.</th>
</tr>
</thead>
<tbody>
<tr>
<td>subfile1</td>
<td>A null-terminated string containing the logical name of the sub-file. This may be another superfile.</td>
</tr>
<tr>
<td>subfile2</td>
<td>A null-terminated string containing the logical name of the sub-file. This may be another superfile.</td>
</tr>
</tbody>
</table>

Return: Null.

The ReplaceSuperFile function removes the subfile1 from the list of files comprising the superfile and replaces it with subfile2.

This function may be included in a superfile transaction.

Example:

```plaintext
SEQUENTIAL(
  STD.File.StartSuperFileTransaction(),
  STD.File.ReplaceSuperFile('MySuperFile', 'MyOldSubFile', 'MyNewSubFile'),
  STD.File.FinishSuperFileTransaction()
);
```
PromoteSuperFileList


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>supernames</td>
<td>A set of null-terminated strings containing the logical names of the superfiles to act on. Any that don't exist will be created. The contents of each superfile will be moved to the next in the list (NB -- each superfile must contain different sub-files).</td>
</tr>
<tr>
<td>addhead</td>
<td>Optional. A null-terminated string containing a comma-delimited list of logical file names to add to the first superfile after the promotion process is complete.</td>
</tr>
<tr>
<td>deltail</td>
<td>Optional. A boolean value specifying whether to physically delete the contents moved out of the last superfile. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>createjustone</td>
<td>Optional. A boolean value specifying whether to only create a single superfile (truncate the list at the first non-existent superfile). If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>reverse</td>
<td>Optional. A boolean value specifying whether to reverse the order of processing the supernames list, effectively &quot;demoting&quot; instead of &quot;promoting&quot; the sub-files. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>oldlist</td>
<td>The name of the attribute that receives the returned string containing the list of the previous subfile contents of the emptied superfile.</td>
</tr>
</tbody>
</table>

Return: PromoteSuperFileList returns Null; fPromoteSuperFileList returns a string.

The PromoteSuperFileList function moves the subfiles from the first entry in the list of supernames to the next in the list, subsequently repeating the process through the list of supernames.

This function does not use superfile transactions, it is an atomic operation.

Example:

STD.File.PromoteSuperFileList(['Super1','Super2','Super3'],
  'NewSub1');
//Moves what was in Super1 to Super2,
//what was in Super2 to Super3, replacing what was in Super3,
//and putting NewSub1 in Super1
FinishSuperFileTransaction

STD.File.FinishSuperFileTransaction([ rollback ])

<table>
<thead>
<tr>
<th>rollback</th>
<th>Optional. A boolean flag that indicates whether to commit (FALSE) or roll back (TRUE) the transaction. If omitted, the default is FALSE.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Null.</td>
</tr>
</tbody>
</table>

The FinishSuperFileTransaction function terminates a superfile maintenance transaction frame.

If the rollback flag is FALSE, the transaction is committed atomically and the transaction frame closes. Otherwise, the transaction is rolled back and the transaction frame closes.

At commit, the superfile is “write” locked for the transaction to actually execute, and all locks are released when the transaction frame closes. During the timeframe of the “write” lock at commit (usually small time window), no concurrent “read” locks are allowed.

Example:

```plaintext
IMPORT STD;
WeeklyRollup:='~Training::Examples::WeeklyRollup';
WeeklySF :='~Training::Examples::Weekly';
DailySF  :='~Training::Examples::Daily';

DailyDS := DATASET(DailySF,{string Myfield},THOR);

SEQUENTIAL(STD.File.StartSuperFileTransaction(),
           STD.File.ClearSuperFile(DailySF),
           OUTPUT(DailyDS,,WeeklyRollup),
           STD.File.AddSuperFile(WeeklySF,WeeklyRollup),
           STD.File.FinishSuperFileTransaction());

//executes the OUTPUT after a "read" lock on the superfile DailySF
//has been initiated by the ClearSuperFile Maintenance function,
//which in turn executes only at the FinishTransaction
```
External Files
### ExternalLogicalFileName

**STD.File.ExternalLogicalFileName** ( `machineIP`, `filename` )

<table>
<thead>
<tr>
<th><strong>machineIP</strong></th>
<th>A null-terminated string containing the IP address of the remote machine.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>filename</strong></td>
<td>A null-terminated string containing the path/name of the file.</td>
</tr>
</tbody>
</table>

**Return:** ExternalLogicalFileName returns a VARSTRING (null-terminated) value.

The **ExternalLogicalFileName** function returns an appropriately encoded external logical file name that can be used to directly read a file from any node that is running the dafilesrv utility (typically a landing zone). It handles upper case characters by escaping those characters in the return string.

**Example:**

```plaintext
IP := '10.150.254.6';
file := '/c$/training/import/AdvancedECL/people';
DS1 := DATASET(STD.File.ExternalLogicalFileName(IP, file),
    Training_Advanced.Layout_PeopleFile, FLAT);
OUTPUT(STD.File.ExternalLogicalFileName(IP, file));
//returns:
//~file::10.150.254.6::c$::training::import::^advanced^e^c^l::people
OUTPUT(DS1);
//returns records from the external file
```
MoveExternalFile

STD.File.MoveExternalFile( location, frompath, topath )

<table>
<thead>
<tr>
<th>location</th>
<th>A null-terminated string containing the IP address of the remote machine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>frompath</td>
<td>A null-terminated string containing the path/name of the file to move.</td>
</tr>
<tr>
<td>topath</td>
<td>A null-terminated string containing the path/name of the target file.</td>
</tr>
</tbody>
</table>

The MoveExternalFile function moves the single physical file specified by the frompath to the topath. Both frompath and topath are on the same remote machine, identified by the location. The dafileserv utility program must be running on the location machine.

Example:

```plaintext
IP := '10.150.254.6';
infile := '/c$/training/import/AdvancedECL/people';
outfile := '/c$/training/import/DFUtest/people';
STD.File.MoveExternalFile(IP,infile,outfile);
```
DeleteExternalFile

STD.File.DeleteExternalFile( location, path )

<table>
<thead>
<tr>
<th>location</th>
<th>A null-terminated string containing the IP address of the remote machine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>A null-terminated string containing the path/name of the file to remove.</td>
</tr>
</tbody>
</table>

The `DeleteExternalFile` function removes the single physical file specified by the `path` from the `location`. The `dafilenserv` utility program must be running on the `location` machine.

Example:

```plaintext
IP := '10.150.254.6';
infile := '/c$/training/import/AdvancedECL/people';
STD.File.DeleteExternalFile(IP,infile);
```
CreateExternalDirectory

STD.File.CreateExternalDirectory( location, path )

<table>
<thead>
<tr>
<th>location</th>
<th>A null-terminated string containing the IP address of the remote machine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>A null-terminated string containing the directory path to create.</td>
</tr>
</tbody>
</table>

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

Example:

IP    := '10.150.254.6';
path := '/c$/training/import/NewDir';
STD.File.CreateExternalDirectory(IP,path);
RemoteDirectory


<table>
<thead>
<tr>
<th>\textit{machineIP}</th>
<th>A null-terminated string containing the IP address of the remote machine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{dir}</td>
<td>A null-terminated string containing the path to the directory to read. This must be in the appropriate format for the operating system running on the remote machine.</td>
</tr>
<tr>
<td>\textit{mask}</td>
<td>Optional. A null-terminated string containing the filemask specifying which files to include in the result. If omitted, the default is &quot;*&quot; (all files).</td>
</tr>
<tr>
<td>\textit{recurse}</td>
<td>Optional. A boolean flag indicating whether to include files from sub-directories under the \textit{directory}. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>

Return: RemoteDirectory returns a dataset in the following format:

\begin{verbatim}
EXPORT FsFilenameRecord := RECORD
    STRING name;        //filename
    UNSIGNED8 size;    //filesize
    STRING19 modified; //date-time stamp
END;
\end{verbatim}

The RemoteDirectory function returns a list of files as a dataset in the format listed above from the specified \textit{machineIP} and \textit{directory}. If \textit{recurse} is set to TRUE, then the name field contains the relative path to the file from the specified \textit{directory}.

The mask argument is a string that can include wildcard characters. Valid wildcard characters are '*' (to match zero or more characters) and '?' (to match exactly one character). Non-wild characters are matched exactly and are case-sensitive.

Example:

\begin{verbatim}
OUTPUT(STD.File.RemoteDirectory('edata12','\in','*.d00'));
\end{verbatim}
File Browsing
SetColumnMapping

STD.File.SetColumnMapping( file, mapping );

<table>
<thead>
<tr>
<th>file</th>
<th>A null-terminated string containing the logical filename.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mapping</td>
<td>A null-terminated string containing a comma-delimited list of field mappings.</td>
</tr>
</tbody>
</table>

The **SetColumnMapping** function defines how the data in the fields of the file must be transformed between the actual data storage format and the input format used to query that data.

The format for each field in the *mapping* list is:

```plaintext
<field>{set(<transform>( args),...),get(<transform>,...),displayname(<name>))
```

- **<field>** - The name of the field in the file.
- **set** - Optional. Specifies the transforms applied to the values supplied by the user to convert them to values in the file.
- **<transform>** - Optional. The name of a function to apply to the value. This is typically the name of a plugin function. The value being converted is always provided as the first parameter to the function, but extra parameters can be specified in brackets after the transform name (similar to SALT hygiene).
- **get** - Optional. Specifies the transforms applied to the values in the file to convert them to the formatted values as they are understood by the user.
- **displayname** - Optional. Allows a different name to be associated with the field than the user would naturally understand.

Note that you may mix unicode and string functions, as the system automatically converts the parameters to the appropriate types expected for the functions.

**Example:**

```plaintext
// A file where the firstname(string) and lastname(unicode) are //always upper-cased:
// There is no need for a displayname since it isn't really a //different field as far as the user is concerned, and there is //obviously no get transformations.
firstname{set(stringlib.StringToUpperCase)},
surname{set(unicodelib.UnicodeToUpperCase)}
// A name translated using a phonetic key
// it is worth specifying a display name here, because it will make //more sense to the user, and the user may want to enter either the //translated or untranslated names.
dph_lname{set(metaphonelib.DMetaPhone1),
displayname(lname)}
// A file where a name is converted to a token using the namelib //functions. (I don't think we have an example of this)
// (one of the few situations where a get() attribute is useful)
fnametoken{set(namelib.nameToToken),
    get(namelib.tokenToName),
displayname(fname)}
// upper case, and only include digits and alphabetic.
searchname{set(stringlib.StringToUpperCase,
    stringlib.StringFilter( 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789'))}
// A file with a field that that needs to remove accents and then //uppercase:
```
lastname(set(unicodeLib.CleanAccents,stringLib.StringToUpperCase))
GetColumnMapping

\[ \text{result} := \text{STD.File.GetColumnMapping}(\text{file}); \]

<table>
<thead>
<tr>
<th>file</th>
<th>A null-terminated string containing the logical filename.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>GetColumnMapping returns a null-terminated string containing the comma-delimited list of field mappings for the file.</td>
</tr>
</tbody>
</table>

The GetColumnMapping function returns the field mappings for the file, in the same format specified for the SetColumnMapping function.

Example:

\[ \text{Maps} := \text{STD.File.GetColumnMapping('Thor::in::SomeFile')}; \]
AddFileRelationship

STD.File.AddFileRelationship(  primary, secondary, primaryfields, secondaryfields,  [  relationship ],  cardinality, payload  [,  description ] );

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>primary</td>
<td>A null-terminated string containing the logical filename of the primary file.</td>
</tr>
<tr>
<td>secondary</td>
<td>A null-terminated string containing the logical filename of the secondary file.</td>
</tr>
<tr>
<td>primaryfields</td>
<td>A null-terminated string containing the name of the primary key field for the primary file. The value &quot;<strong>fileposition</strong>&quot; indicates the secondary is an INDEX that must use FETCH to access non-keyed fields.</td>
</tr>
<tr>
<td>secondaryfields</td>
<td>A null-terminated string containing the name of the foreign key field relating to the primary file.</td>
</tr>
<tr>
<td>relationship</td>
<td>A null-terminated string containing either &quot;link&quot; or &quot;view&quot; indicating the type of relationship between the primary and secondary files. If omitted, the default is &quot;link.&quot;</td>
</tr>
<tr>
<td>cardinality</td>
<td>A null-terminated string containing the kind of relationship between the primary and secondary files. The format is X:Y where X indicates the primary and Y indicates the secondary. Valid values for X and Y are &quot;1&quot; or 'M'.</td>
</tr>
<tr>
<td>payload</td>
<td>A BOOLEAN value indicating whether the primary or secondary are payload INDEXes.</td>
</tr>
<tr>
<td>description</td>
<td>A null-terminated string containing the relationship description.</td>
</tr>
</tbody>
</table>

The AddFileRelationship function 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.

The primaryfields and secondaryfields parameters can have the same format as the column mappings for a file (see the SetColumnMappings function documentation), although they will often be just a list of fields.

They are currently used in two different ways:

First, the roxie browser needs a way of determining which indexes are built from which files. A "view" relationship should be added each time an index is built from a file, like this:

```plaintext
STD.File.AddFileRelationship(DG_FlatFileName, DG_IndexFileName, '', '', 'view', '1:1', false);
```

To implement the roxie browser there is no need to define the primaryfields or secondaryfields, so passing blank strings is recommended.

Second, the browser needs a way of finding all the original information from the file from an index.

This stage depends on the nature of the index:

a) If the index contains all the relevant data from the original file you don't need to do anything.

b) If the index uses a fileposition field to FETCH extra data from the original file then add a relationship between the original file and the index, using a special value of __fileposition__ to indicate the record is retrieved using a FETCH.

```plaintext
STD.File.AddFileRelationship('fetch_file',  
  'new_index',  
  '__fileposition__',  
  'index_filepos_field',  
  'link',  
  true);
```
The original file is the primary, since the rows are uniquely identified by the fileposition (also true of the index), and the retrieval is efficient.

c) If the index uses a payload field which needs to be looked up in another index to provide the information, then you need to define a relationship between the new index and the index that provides the extra information. The index providing the extra information is the primary.

```plaintext
STD.File.AddFileRelationship('related_index', 'new_index', 'related_key_fields', 'index_filepos_field', 'link', '1:M', true);
```

The *payload* flag is there so that the roxie browser can distinguish this link from a more general relationship between two files.

You should ensure any super-file names are expanded if the relation is defined between the particular sub-files.

While going through all the attributes it may be worth examining whether it makes sense to add relationships for any other combinations of files. It won't have any immediate beneficial effect, but would once we add an ER diagram to the system. A couple of examples may help illustrate the syntax.

For a typical example, datasets with a household and person file, the following defines a relationship linking by house hold id (hhid):

```plaintext
STD.File.AddFileRelationship('HHFile', 'PersonFile', 'hhid', 'hhid', 'link', '1:M', false);
```

Here's a more hypothetical example--a file query with firstname, lastname related to an index with phonetic names you might have:

```plaintext
STD.File.AddFileRelationship('names', 'inquiries', 'plastname{set(phonetic)}, pfirstname{set(phonetic)}', 'lastname{set(fail)}, firstname{set(fail)}', 'link', '1:M', false);
```

Note, the fail mapping indicates that you can use the phonetic mapping from inquiries to names, but there is no way of mapping from names to inquiries. There could equally be get(fail) attributes on the index fields.

Example:

```plaintext
Maps := STD.File.GetColumnMapping('Thor::in::SomeFile');
```
**FileRelationshipList**

STD.File.FileRelationshipList(  
  primary, secondary  [ , primaryfields ] [ , secondaryfields ] [ , relationship ] );

<table>
<thead>
<tr>
<th>primary</th>
<th>A null-terminated string containing the logical filename of the primary file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>secondary</td>
<td>A null-terminated string containing the logical filename of the secondary file.</td>
</tr>
<tr>
<td>primaryfields</td>
<td>A null-terminated string containing the name of the primary key field for the primary file. The value &quot;<strong>fileposition</strong>&quot; indicates the secondary is an INDEX that must use FETCH to access non-keyed fields. If omitted, the default is an empty string.</td>
</tr>
<tr>
<td>secondaryfields</td>
<td>A null-terminated string containing the name of the foreign key field relating to the primary file. If omitted, the default is an empty string.</td>
</tr>
<tr>
<td>relationship</td>
<td>A null-terminated string containing either &quot;link&quot; or &quot;view&quot; indicating the type of relationship between the primary and secondary files. If omitted, the default is &quot;link.&quot;</td>
</tr>
</tbody>
</table>

Return: FileRelationshipList returns a dataset in the FsFileRelationshipRecord format.

The **FileRelationshipList** function returns a list file relationships between the primary and secondary files. The return records are structured in the FsFileRelationshipRecord format:

```plaintext
EXPORT FsFileRelationshipRecord := RECORD
  STRING primaryfile {MAXLENGTH(1023)};
  STRING secondaryfile {MAXLENGTH(1023)};
  STRING primaryflds {MAXLENGTH(1023)};
  STRING secondaryflds {MAXLENGTH(1023)};
  STRING kind {MAXLENGTH(16)};
  STRING cardinality {MAXLENGTH(16)};
  BOOLEAN payload;
  STRING description {MAXLENGTH(1023)};
END;
```

Example:

```plaintext
OUTPUT(STD.File.FileRelationshipList('names', 'inquiries'));
```

See Also: AddFileRelationship
RemoveFileRelationship

STD.File.RemoveFileRelationship( primary, secondary, [ , primaryfields ] [ , secondaryfields ] [ , relationship ] );

<table>
<thead>
<tr>
<th>primary</th>
<th>A null-terminated string containing the logical filename of the primary file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>secondary</td>
<td>A null-terminated string containing the logical filename of the secondary file.</td>
</tr>
<tr>
<td>primaryfields</td>
<td>A null-terminated string containing the name of the primary key field for the primary file. The value &quot;<strong>fileposition</strong>&quot; indicates the secondary is an INDEX that must use FETCH to access non-keyed fields. If omitted, the default is an empty string.</td>
</tr>
<tr>
<td>secondaryfields</td>
<td>A null-terminated string containing the name of the foreign key field relating to the primary file. If omitted, the default is an empty string.</td>
</tr>
<tr>
<td>relationship</td>
<td>A null-terminated string containing either &quot;link&quot; or &quot;view&quot; indicating the type of relationship between the primary and secondary files. If omitted, the default is &quot;link.&quot;</td>
</tr>
</tbody>
</table>

The **RemoveFileRelationship** function removes a file relationship between the **primary** and **secondary** files.

Example:

STD.File.RemoveFileRelationship('names', 'inquiries');

See Also: AddFileRelationship
DfuPlusExec

STD.File.DfuPlusExec( commandline )

| `commandline` | A null-terminated string containing the DFUPlus command line to execute. The valid arguments are documented in the Client Tools manual, in the section describing the Command Line DFU program. |

The **DfuPlusExec** action executes the specified `commandline` just as the DfuPlus executable program would. This allows you to have all the functionality of DfuPlus available within your ECL code.

Unless you need to access a foreign instance of the platform, the `server=` parameter for DfuPlus should be omitted, which then defaults to the value contained in the environment's configuration. In a containerized system, this defaults to the local eclservices service. If for some reason that doesn't work, the default can be overridden by the value set in `global.defaultEsp`. In a bare-metal system, this is the service named WsSMC (internal to ECLWatch).

**Example:**

```ecl
IMPORT STD;
usr := 'username=emilyd ';
pwd := 'password=password ';
ovr := 'overwrite=1 ';
repl := 'replicate=1 ';
action := 'action=spray ';
srcline := 'srcline=mydropzone ';
srfile := 'srfile=originalperson ';
dstname := 'dstname=EmilyTutorial::originalperson ';
dstcluster := 'dstcluster=data ';
fmt := 'format=fixed ';
recsize := 'recordsize=124 ';
cmd := usr + pwd + ovr + repl + action + srcline
    + srfile + dstname + dstcluster + fmt + recsize;
STD.File.DfuPlusExec(cmd);
```
AbortDfuWorkunit

STD.File.AbortDfuWorkunit( dfuwuid [, espserverIPport ] )

| dfuwuid | A null-terminated string containing the DFU workunit ID (DFUWUID) for the job to abort. This value is returned by the “leading-f” versions of the Copy, SprayFixed, SprayVariable, SprayXML, and Despray FileServices functions. |
|----------------------|
| espserverIPport | Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with "/FileSpray" appended. |

The **AbortDfuWorkunit** function aborts the specified DFU workunit. Typically that workunit will have been launched with its timeout parameter set to zero (0).

Example:

```c
STD.File.AbortDfuWorkunit('D20051108-143758');
```
### STD.File.Copy

**STD.File.Copy** (sourceLogicalName, destinationGroup, destinationLogicalName, [sourceDali], [time-Out], [espServerIPPort], [maxConnections], [allowOverwrite], [replicate], [asSuperfile], [compress], [forcePush], [transferBufferSize], [preserveCompression], [noSplit], [expireDays], [ensure]);

dfuwuid := STD.File.ICopy( sourceLogicalName, destinationGroup, destinationLogicalName, [sourceDali], [timeOut], [espServerIPPort], [maxConnections], [allowOverwrite], [replicate], [asSuperfile], [compress], [forcePush], [transferBufferSize], [preserveCompression], [noSplit], [expireDays], [ensure]);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceLogicalName</td>
<td>A null-terminated string containing the logical name of the file.</td>
</tr>
<tr>
<td>destinationGroup</td>
<td>A null-terminated string containing the destination cluster for the file.</td>
</tr>
<tr>
<td>destinationLogicalName</td>
<td>A null-terminated string containing the new logical name of the file.</td>
</tr>
<tr>
<td>sourceDali</td>
<td>Optional. A null-terminated string containing the IP and Port of the Dali containing the file to copy. If omitted, the default is an intra-Dali copy.</td>
</tr>
<tr>
<td>timeOut</td>
<td>Optional. An integer value indicating the timeout setting. If omitted, the default is -1. If set to zero (0), execution control returns immediately to the ECL workunit without waiting for the DFU workunit to complete.</td>
</tr>
<tr>
<td>espServerIPPort</td>
<td>Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with &quot;/FileSpray&quot; appended.</td>
</tr>
<tr>
<td>maxConnections</td>
<td>Optional. An integer specifying the maximum number of connections. If omitted, the default is -1, which indicates the system chooses a suitable default based on the size of the cluster.</td>
</tr>
<tr>
<td>allowOverwrite</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to allow the new file to overwrite an existing file of the same name. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>replicate</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to automatically replicate the new file. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>asSuperfile</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to treat the file as a superfile. If omitted, the default is FALSE. If TRUE and the file to copy is a superfile, then the operation creates a superfile on the target, creating subfiles as needed while overwriting only those already existing subfiles whose content has changed. If FALSE and the file to copy is a superfile, then the operation consolidates all the superfile content into a single logical file on the target, not a superfile. If FALSE and the file to copy is a superfile containing INDEXes, then the operation is not valid and will produce an error.</td>
</tr>
<tr>
<td>compress</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to LZW compress the new file. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>forcePush</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to execute the copy process on the source nodes and push to the targets instead of executing on the targets and pulling from the source. This option is only valid within the same environment. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>
## File Movement

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>transferBufferSize</code></td>
<td>Optional. An integer value to override the DFU Server’s buffer size value (default is 64k)</td>
</tr>
<tr>
<td><code>preserveCompression</code></td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to preserve the compression of the old file when copying. If omitted, the default is TRUE.</td>
</tr>
<tr>
<td><code>noSplit</code></td>
<td>Optional. A boolean TRUE or FALSE flag indicating to not split a file part to multiple target parts. Default is FALSE.</td>
</tr>
<tr>
<td><code>expireDays</code></td>
<td>Optional. Specifies the file is a temporary file to be automatically deleted after the specified number of days since the file was read. If omitted, the default is -1 (never expires). If set to 0, the file is automatically deleted when it reaches the threshold set in Sasha Server’s <code>expiryDefault</code> setting.</td>
</tr>
<tr>
<td><code>ensure</code></td>
<td>Optional. Copies logical file, but does not copy file parts if they already exist. Default is FALSE.</td>
</tr>
<tr>
<td><code>dfuwuid</code></td>
<td>The attribute name to receive the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job.</td>
</tr>
</tbody>
</table>

**Return:** Copy returns a null-terminated string containing the DFU workunit ID (DFUWUID).

The **Copy** function takes a logical file and copies it to another logical file. This may be done within the same cluster or to another cluster. The Destination cannot be foreign file.

**Example:**

```plaintext
```
**DeSpray**

STD.File.DeSpray( **logicalname**, **destinationIP**, **destinationpath**, [ **timeout** ], [ **espserverIP-port** ], [ **maxConnections** ], [ **allowoverwrite** ], [ **destinationPlane** ])

dfuwuid := STD.File.fDeSpray( **logicalname**, **destinationIP**, **destinationpath**, [ **timeout** ], [ **espserverIP-port** ], [ **maxConnections** ], [ **allowoverwrite** ], [ **destinationPlane** ])

| **logicalname** | A null-terminated string containing the logical name of the file. |
| **destinationIP** | A null-terminated string containing the destination IP address of the file. Deprecated, you should use destinationPlane instead. |
| **destinationpath** | A null-terminated string containing the path and name of the file. |
| **timeout** | Optional. An integer value indicating the timeout setting. If omitted, the default is -1. If set to zero (0), execution control returns immediately to the ECL workunit without waiting for the DFU workunit to complete. |
| **espserverIP-port** | Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with "/FileSpray" appended. |
| **maxConnections** | Optional. An integer specifying the maximum number of connections. If omitted, the default is -1, which indicates the system chooses a suitable default based on the size of the cluster. |
| **allowoverwrite** | Optional. A boolean TRUE or FALSE flag indicating whether to allow the new file to overwrite an existing file of the same name. If omitted, the default is FALSE. |
| **destinationPlane** | Optional. The destination storage plane. Note: destinationPlane should not be used at the same time as destinationIP. In a containerized deployment, destinationPlane is required if you have more than one Landing Zone. |
| **dfuwuid** | The attribute name to receive the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job. |

**Return:** fDeSpray returns a null-terminated string containing the DFU workunit ID (DFUWUID).

The **DeSpray** function takes a logical file and desprays it (combines all parts on each supercomputer node into a single physical file) to the landing zone.

Example:

```plaintext
STD.File.DeSpray('OUT::MyFile',
    '10.150.50.14',
    'c:\OutputData\MyFile.txt',
    -1,
    'http://10.150.50.12:8010/FileSpray');
```
## RemotePull

The `STD.File.RemotePull` function is used to move files from one location to another across a network. It accepts several parameters to control the behavior of the file movement. Here's a breakdown of the function and its parameters:

**Function Signature:**

```plaintext
STD.File.RemotePull( remoteURL, sourcelogicalname, destinationGroup, destinationlogicalname, [ ,timeout] [ ,maxConnections] [ ,allowoverwrite] [ ,replicate] [ ,asSuperfile] [ ,forcePush] [ ,transferBufferSize] [ ,wrap] [ ,compress] [ ,noSplit] [ ,expireDays])
```

**Parameters:**

- **remoteURL**
  A null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the remote ESP server program. This is usually the same IP and port as its ECL Watch, with "/FileSpray" appended.

- **sourcelogicalname**
  A null-terminated string containing the local logical name of the file.

- **destinationGroup**
  A null-terminated string containing the name of the destination cluster.

- **destinationlogicalname**
  A null-terminated string containing the logical name to give the file on the remote cluster (this must be completely specified, including the domain).

- **timeout**
  Optional. An integer value indicating the timeout setting. If omitted, the default is -1. If set to zero (0), execution control returns immediately to the ECL workunit without waiting for the DFU workunit to complete.

- **maxConnections**
  Optional. An integer specifying the maximum number of connections. If omitted, the default is -1, which indicates the system chooses a suitable default based on the size of the cluster.

- **allowoverwrite**
  Optional. A boolean TRUE or FALSE flag indicating whether to allow the new file to overwrite an existing file of the same name. If omitted, the default is FALSE.

- **replicate**
  Optional. A boolean TRUE or FALSE flag indicating whether to automatically replicate the new file. If omitted, the default is FALSE.

- **asSuperfile**
  Optional. A boolean TRUE or FALSE flag indicating whether to treat the file as a superfile. If omitted, the default is FALSE. If TRUE and the file to copy is a superfile, then the operation creates a superfile on the target, creating subfiles as needed while overwriting only those already existing subfiles whose content has changed. If FALSE and the file to copy is a superfile, then the operation consolidates all the superfile content into a single logical file on the target, not a superfile.

- **forcePush**
  Optional. A boolean TRUE or FALSE flag indicating whether to execute the copy process on the source nodes and push to the targets instead of executing on the targets and pulling from the source. If omitted, the default is FALSE.

- **transferBufferSize**
  Optional. An integer specifying the size in bytes of the transfer buffer. Sometimes using larger values can speed the process. If omitted, a default buffer size of 64K is used.

- **wrap**
  Optional. A boolean TRUE or FALSE flag indicating whether to automatically wrap the file parts when copying to smaller sized clusters. For example, copying from a 6-node cluster to a 3-node cluster, two file parts will end up on each node; the difference is whether node 1 gets parts 1 and 2 or parts 1 and 4. If omitted, the default is FALSE.

- **compress**
  Optional. A boolean TRUE or FALSE flag indicating whether to automatically LZW compress the new file. If omitted, the default is FALSE.

- **noSplit**
  Optional. A boolean TRUE or FALSE flag indicating to not split a file part to multiple target parts. Default is FALSE.
**expireDays**
Optional. Specifies the file is a temporary file to be automatically deleted after the specified number of days since the file was read. If omitted, the default is -1 (never expires). If set to 0, the file is automatically deleted when it reaches the threshold set in Sasha Server's **expiryDefault** setting.

**dfuwuid**
The definition name to receive the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job.

Return: fRemotePull returns a null-terminated string containing the DFU workunit ID (DFUWUID).

The **RemotePull** function executes on the remoteURL, copying the **sourcelogicalname** from the local environment that instantiated the operation to the remote environment's **destinationGroup** cluster, giving it the **destinationlogicalname**. This is very similar to using the STD.File.Copy function and specifying its **espserverlPport** parameter. 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.

Example:

```sql
                    '~THOR::LOCAL::MyFile',
                    'RemoteThor',
                    '~REMOTETHOR::LOCAL::MyFile');
```
Replicate

STD.File.Replicate ( filename [ , timeout ] [ , espserverIPport ])

dfuwuid := STD.File.Replicate( filename [ , timeout ] [ , espserverIPport ]);  

| filename  | A null-terminated string containing the logical name of the file. |
| timeout   | Optional. An integer value indicating the timeout setting. If omitted, the default is -1. If set to zero (0), execution control returns immediately to the ECL workunit without waiting for the DFU workunit to complete. |
| espserverIPport | Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with "/FileSpray" appended. |
| dfuwuid   | The attribute name to receive the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job. |

The Replicate function copies the individual parts of the filename to the mirror disks for the cluster. Typically, this means that the file part on one node’s C drive is copied to its neighbors D drive.

Example:

A := STD.File.Replicate('Fred');
SprayFixed


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceIP</td>
<td>A null-terminated string containing the IP address or hostname of the Dropzone where the file is located.</td>
</tr>
<tr>
<td>sourcepath</td>
<td>A null-terminated string containing the path and name of the file.</td>
</tr>
<tr>
<td>recordsize</td>
<td>An integer containing the size of the records in the file.</td>
</tr>
<tr>
<td>destinationgroup</td>
<td>A null-terminated string containing the name of the specific supercomputer within the target cluster.</td>
</tr>
<tr>
<td>destinationlogicalname</td>
<td>A null-terminated string containing the logical name of the file.</td>
</tr>
<tr>
<td>timeout</td>
<td>Optional. An integer value indicating the timeout setting. If omitted, the default is -1. If set to zero (0), execution control returns immediately to the ECL workunit without waiting for the DFU workunit to complete.</td>
</tr>
<tr>
<td>espserverIPport</td>
<td>Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with &quot;FileSpray&quot; appended.</td>
</tr>
<tr>
<td>maxConnections</td>
<td>Optional. An integer specifying the maximum number of connections. If omitted, the default is -1, which indicates the system chooses a suitable default based on the size of the cluster.</td>
</tr>
<tr>
<td>allowoverwrite</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to allow the new file to overwrite an existing file of the same name. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>replicate</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to replicate the new file. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>compress</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to compress the new file. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>failIfNoSourceFile</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether a missing file triggers a failure. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>expireDays</td>
<td>Optional. Specifies the file is a temporary file to be automatically deleted after the specified number of days since the file was read. If omitted, the default is -1 (never expires). If set to 0, the file is automatically deleted when it reaches the threshold set in Sasha Server's expiryDefault setting.</td>
</tr>
<tr>
<td>dfuServerQueue</td>
<td>Name of target DFU Server queue. Default is &quot;&quot; (empty) for the first DFU queue in the environment.</td>
</tr>
<tr>
<td>noSplit</td>
<td>Optional. A boolean TRUE or FALSE flag indicating to not split a file part to multiple target parts. Default is FALSE.</td>
</tr>
</tbody>
</table>
### Standard Library Reference
#### File Movement

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>noCommon</strong></td>
<td>Optional. A boolean TRUE or FALSE flag for &quot;commoning up&quot; of pull or push processes on same host. Set to FALSE to &quot;common up&quot; the operation on same host. Default can be set in configuration. Use GetNoCommonDefault to retrieve default setting. The value of this parameter can have a significant impact on performance.</td>
</tr>
<tr>
<td><strong>sourcePlane</strong></td>
<td>The name of the landing zone containing the file</td>
</tr>
<tr>
<td><strong>destinationNumParts</strong></td>
<td>Override the number of parts to be created when spraying. The default is 0 which means it will create the same number of parts as the target cluster.</td>
</tr>
<tr>
<td><strong>dfuwuid</strong></td>
<td>The attribute name to receive the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job.</td>
</tr>
</tbody>
</table>

**Return:**

fSprayFixed returns a null-terminated string containing the DFU workunit ID (DFUWUID).

The **SprayFixed** function takes fixed-format file from the landing zone and distributes it across the nodes of the destination supercomputer.

**Example:**

```java
```
SprayDelimited / SprayVariable


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceIP</td>
<td>A null-terminated string containing the IP address or hostname of the Dropzone where the file is located.</td>
</tr>
<tr>
<td>sourcePath</td>
<td>A null-terminated string containing the path and name of the file.</td>
</tr>
<tr>
<td>sourceMaxRecordSize</td>
<td>Optional. An integer containing the maximum size of the records in the file. If omitted, the default is 4096.</td>
</tr>
<tr>
<td>sourceCsvSeparate</td>
<td>Optional. A null-terminated string containing the CSV field separator. If omitted, the default is '\,'.</td>
</tr>
<tr>
<td>sourceCSVterminate</td>
<td>Optional. A null-terminated string containing the CSV record separator. If omitted, the default is '\n,\r\n'.</td>
</tr>
<tr>
<td>sourceCSVquote</td>
<td>Optional. A null-terminated string containing the CSV quoted field delimiter. If omitted, the default is '&quot;'.</td>
</tr>
<tr>
<td>destinationGroup</td>
<td>A null-terminated string containing the name of the specific supercomputer within the target cluster.</td>
</tr>
<tr>
<td>destinationLogicalName</td>
<td>A null-terminated string containing the logical name of the file.</td>
</tr>
<tr>
<td>timeout</td>
<td>Optional. An integer value indicating the timeout setting. If omitted, the default is -1. If set to zero (0), execution control returns immediately to the ECL workunit without waiting for the DFU workunit to complete.</td>
</tr>
<tr>
<td>espServerIpPort</td>
<td>Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with &quot;/FileSpray&quot; appended.</td>
</tr>
<tr>
<td>maxConnections</td>
<td>Optional. An integer specifying the maximum number of connections. If omitted, the default is -1, which indicates the system chooses a suitable default based on the size of the cluster.</td>
</tr>
<tr>
<td>allowOverwrite</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to allow the new file to overwrite an existing file of the same name. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>replicate</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to replicate the new file. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>compress</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to compress the new file. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>
**sourceCsvEscape**

Optional. A null-terminated string containing the CSV escape characters. If omitted, the default is none.

**failIfNoSourceFile**

Optional. A boolean TRUE or FALSE flag indicating whether to allow the spray to fail if no source file is found. If omitted, default is FALSE.

**recordStructurePresent**

Optional. A boolean TRUE or FALSE flag indicating whether to derive the record structure from the header of the file. If omitted, the default is FALSE.

**quotedTerminator**

Optional. A boolean TRUE or FALSE flag indicating whether the terminator character can be included in a quoted field. Defaults to TRUE. If FALSE, it allows quicker partitioning of the file (avoiding a complete file scan).

**expireDays**

Optional. Specifies the file is a temporary file to be automatically deleted after the specified number of days since the file was read. If omitted, the default is -1 (never expires). If set to 0, the file is automatically deleted when it reaches the threshold set in Sasha Server's `expiryDefault` setting.

**encoding**

A null-terminated string containing the encoding. Can be set to one of the following: ascii, utf8, utf8n, utf16, utf16e, utf16be, utf32, utf32le, utf32be. If omitted, the default is ascii.

**dfuServerQueue**

Name of target DFU Server queue. Default is "" (empty) for the first DFU queue in the environment.

**noSplit**

Optional. A boolean TRUE or FALSE flag indicating to not split a file part to multiple target parts. Default is FALSE.

**noCommon**

Optional. A boolean TRUE or FALSE flag for "commoning up" of pull or push processes on same host. Set to FALSE to "common up" the operation on same host. Default can be set in configuration. Use `GetNoCommonDefault` to retrieve default setting. The value of this parameter can have a significant impact on performance.

**sourcePlane**

The name of the landing zone containing the file

**destinationNumParts**

Override the number of parts to be created when spraying. The default is 0 which means it will create the same number of parts as the target cluster.

**dfuwuid**

The definition name to receive the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job.

Return:

fSprayDelimited returns a null-terminated string containing the DFU workunit ID (DFUWUID).

The `SprayDelimited` function takes a variable length file from the landing zone and distributes it across the nodes of the destination supercomputer.

The `SprayVariable` function is now called `SprayDelimited` and the `fSprayVariable` function is now called `fSprayDelimited`. The old names are still available for backward compatibility.

Example:

```plaintext
STD.File.SprayDelimited('10.150.50.14',
   'c:\\InputData\\MyFile.txt',
   
   '400way',
   'IN::MyFile',
   -1,
   'http://10.150.50.12:8010/FileSpray');
```
SprayXML

STD.File.SprayXML( sourceIP, sourcepath, [ maxrecordsize ], srcRowTag, [ srcEncoding ], destinationgroup, destinationlogicalname [ timeout ], [ espserverIPport ], [ maxConnections ], [ allowoverwrite ], [ replicate ], [ compress ], [ failIfNoSourceFile ], [ expireDays ], [ dfuServerQueue ], [ noSplit ], [ noCommon ], [ sourcePlane ], [ destinationNumParts ])

dfuwuid := STD.File.fSprayXML( sourceIP, sourcepath, [ maxrecordsize ], srcRowTag, [ srcEncoding ], destinationgroup, destinationlogicalname, [ timeout ], [ espserverIPport ], [ maxConnections ], [ allowoverwrite ], [ replicate ], [ compress ], [ failIfNoSourceFile ], [ expireDays ], [ dfuServerQueue ], [ noSplit ], [ noCommon ], [ sourcePlane ], [ destinationNumParts ])

<table>
<thead>
<tr>
<th>sourceIP</th>
<th>A null-terminated string containing the IP address or hostname of the Dropzone where the file is located.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourcepath</td>
<td>A null-terminated string containing the path and name of the file.</td>
</tr>
<tr>
<td>maxrecordsize</td>
<td>Optional. An integer containing the maximum size of the records in the file. If omitted, the default is 8192.</td>
</tr>
<tr>
<td>srcRowTag</td>
<td>A null-terminated string containing the row delimiting XML tag. Required.</td>
</tr>
<tr>
<td>srcEncoding</td>
<td>Optional. A null-terminated string containing the encoding. If omitted, the default is 'utf8'.</td>
</tr>
<tr>
<td>destinationgroup</td>
<td>A null-terminated string containing the name of the specific supercomputer within the target cluster.</td>
</tr>
<tr>
<td>destinationlogicalname</td>
<td>A null-terminated string containing the logical name of the file.</td>
</tr>
<tr>
<td>timeout</td>
<td>Optional. An integer value indicating the timeout setting. If omitted, the default is -1. If set to zero (0), execution control returns immediately to the ECL workunit without waiting for the DFU workunit to complete.</td>
</tr>
<tr>
<td>espserverIPport</td>
<td>Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with &quot;/FileSpray&quot; appended.</td>
</tr>
<tr>
<td>maxConnections</td>
<td>Optional. An integer specifying the maximum number of connections. If omitted, the default is -1, which indicates the system chooses a suitable default based on the size of the cluster.</td>
</tr>
<tr>
<td>allowoverwrite</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to allow the new file to overwrite an existing file of the same name. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>replicate</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to replicate the new file. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>compress</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to compress the new file. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>failIfNoSourceFile</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether a missing file triggers a failure. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>expireDays</td>
<td>Optional. Specifies the file is a temporary file to be automatically deleted after the specified number of days since the file was read. If omitted, the default is -1 (never expires). If set to 0, the file is automatically deleted when it reaches the threshold set in Sasha Server’s expiryDefault setting.</td>
</tr>
</tbody>
</table>

© 2024 HPCC Systems®. All rights reserved
### File Movement

**dfuServerQueue**
Name of target DFU Server queue. Default is "" (empty) for the first DFU queue in the environment.

**noSplit**
Optional. A boolean TRUE or FALSE flag indicating to not split a file part to multiple target parts. Default is FALSE.

**noCommon**
Optional. A boolean TRUE or FALSE flag for "commoning up" of pull or push processes on same host. Set to FALSE to "common up" the operation on same host. Default can be set in configuration. Use GetNoCommonDefault to retrieve default setting. The value of this parameter can have a significant impact on performance.

**sourcePlane**
The name of the landing zone containing the file

**destinationNumParts**
Override the number of parts to be created when spraying. The default is 0 which means it will create the same number of parts as the target cluster.

**dfuwuid**
The attribute name to receive the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job.

**Return:**

fSprayXML returns a null-terminated string containing the DFU workunit ID (DFUWUID).

### Example:

```plaintext
STD.File.SprayXML('10.150.50.14', 'c:\InputData\MyFile.txt', 
  'Row', '400way', 'IN::MyFile', -1, 
  'http://10.150.50.12:8010/FileSpray');
```
SprayJson


<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceIP</td>
<td>A null-terminated string containing the IP address or hostname of the Drop-zone where the file is located.</td>
</tr>
<tr>
<td>sourcepath</td>
<td>A null-terminated string containing the path and name of the file.</td>
</tr>
<tr>
<td>maxrecordsize</td>
<td>Optional. An integer containing the maximum size of the records in the file. If omitted, the default is 8192.</td>
</tr>
<tr>
<td>srcRowPath</td>
<td>The JSON path that is used to delimit records in the source file. Required.</td>
</tr>
<tr>
<td>srcEncoding</td>
<td>Optional. A null-terminated string containing the encoding (utf8,utf8n,utf16be,utf16le,utf32be,utf32le). If omitted, the default is 'utf8'</td>
</tr>
<tr>
<td>destinationgroup</td>
<td>A null-terminated string containing the name of the group to distribute the file across.</td>
</tr>
<tr>
<td>destinationlogicalname</td>
<td>A null-terminated string containing the logical name of the file to create.</td>
</tr>
<tr>
<td>timeout</td>
<td>Optional. An integer value indicating the timeout setting. If omitted, the default is -1. If set to zero (0), execution control returns immediately to the ECL workunit without waiting for the DFU workunit to complete.</td>
</tr>
<tr>
<td>espserverIPort</td>
<td>Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with &quot;/:FileSpray&quot; appended.</td>
</tr>
<tr>
<td>maxConnections</td>
<td>Optional. An integer specifying the maximum number of connections. If omitted, the default is -1, which indicates the system chooses a suitable default based on the size of the cluster.</td>
</tr>
<tr>
<td>allowoverwrite</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to allow the new file to overwrite an existing file of the same name. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>replicate</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to replicate the new file. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>compress</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether to compress the new file. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>failIfNoSourceFile</td>
<td>Optional. A boolean TRUE or FALSE flag indicating whether a missing file triggers a failure. If omitted, the default is FALSE.</td>
</tr>
<tr>
<td>expireDays</td>
<td>Optional. Specifies the file is a temporary file to be automatically deleted after the specified number of days since the file was read. If omitted, the default is -1 (never expires). If set to 0, the file is automatically deleted when it reaches the threshold set in Sasha Server's expiryDefault setting.</td>
</tr>
</tbody>
</table>

© 2024 HPCC Systems®. All rights reserved
**Standard Library Reference**

**File Movement**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dfuServerQueue</code></td>
<td>Name of target DFU Server queue. Default is &quot;&quot; (empty) for the first DFU queue in the environment.</td>
</tr>
<tr>
<td><code>noSplit</code></td>
<td>Optional. A boolean TRUE or FALSE flag indicating to not split a file part to multiple target parts. Default is FALSE.</td>
</tr>
<tr>
<td><code>noCommon</code></td>
<td>Optional. A boolean TRUE or FALSE flag for &quot;commoning up&quot; of pull or push processes on same host. Set to FALSE to &quot;common up&quot; the operation on same host. Default can be set in configuration. Use GetNoCommonDefault to retrieve default setting. The value of this parameter can have a significant impact on performance.</td>
</tr>
<tr>
<td><code>sourcePlane</code></td>
<td>The name of the landing zone containing the file</td>
</tr>
<tr>
<td><code>destinationNumParts</code></td>
<td>Override the number of parts to be created when spraying. The default is 0 which means it will create the same number of parts as the target cluster.</td>
</tr>
<tr>
<td><code>dfuwuid</code></td>
<td>The attribute name to receive the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job.</td>
</tr>
<tr>
<td><code>username</code></td>
<td>Optional. String containing a username to use for authenticated access to the ESP process; an empty string value indicates that no user authentication is required. If omitted, the default is an empty string.</td>
</tr>
<tr>
<td><code>userPw:</code></td>
<td>Optional. String containing the password to be used with the user cited in the <code>username</code> argument; if <code>username</code> is empty then this is ignored. If omitted, the default is an empty string.</td>
</tr>
</tbody>
</table>

**Return:**

fSprayJson returns a null-terminated string containing the DFU workunit ID (DFUWUID).

The **SprayJson** function takes a well-formed JSON file from a landing zone and distributes it across the nodes of the destination cluster, producing a well-formed JSON file on each node.

**Example:**

```plaintext
```
## WaitDfuWorkunit

STD.File.WaitDfuWorkunit(  *dfuwuid* [ ,*timeout* ] [ ,*espserverIPport* ] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>dfuwuid</em></td>
<td>A null-terminated string containing the DFU workunit ID (DFUWUID) for the job to wait for. This value is returned by the &quot;leading-f&quot; versions of the Copy, DKC, SprayFixed, SprayVariable, SprayXML, and Despray FileServices functions.</td>
</tr>
<tr>
<td><em>timeout</em></td>
<td>Optional. An integer value indicating the timeout setting. If omitted, the default is -1. If set to zero (0), execution control returns immediately to the ECL workunit without waiting for the DFU workunit to complete.</td>
</tr>
<tr>
<td><em>espserverIPport</em></td>
<td>Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with &quot;/FileSpray&quot; appended.</td>
</tr>
</tbody>
</table>

Return: WaitDfuWorkunit returns a null-terminated string containing the final status string of the DFU workunit (such as: scheduled, queued, started, aborted, failed, finished, or monitoring).

The **WaitDfuWorkunit** function waits for the specified DFU workunit to finish. Typically that workunit will have been launched with its *timeout* parameter set to zero (0).

**Example:**

```c
STD.File.WaitDfuWorkunit('D20051108-143758');
```
SetExpireDays

STD.File.SetExpireDays(lfn, expireDays)

<table>
<thead>
<tr>
<th>lfn</th>
<th>A string containing the logical name of the file.</th>
</tr>
</thead>
<tbody>
<tr>
<td>expireDays</td>
<td>Number of days before the file expires. Setting to 0 specifies to use the system's default expire value (specified in the Sasha server's ExpiryDefault attribute).</td>
</tr>
</tbody>
</table>

The **SetExpireDays** action sets a logical file’s expiration criteria (the expireDays attribute). The file is deleted by the Sasha server when a file has not been accessed for the number of days specified.

Example:

```
STD.File.SetExpireDays('-samples::myscope::myfile',30);
//file expires and is deleted after 30 days w/o access
```

See Also: GetExpireDays, ClearExpireDays
GetExpireDays

STD.File.GetExpireDays(\textit{lfn})

| \textit{lfn} | A string containing the logical name of the file. |

The \textbf{GetExpireDays} function retrieves a logical file's expiration criteria (the \textit{expireDays} attribute). A return of -1 indicates that there is no expiration set.

Example:

\begin{verbatim}
A := STD.File.GetExpireDays('~samples::myscope::myfile');
//returns a file's expireDays
\end{verbatim}

See Also: SetExpireDays, ClearExpireDays
### ClearExpireDays

**STD.File.ClearExpireDays(lfn)**

| lfn | A string containing the logical name of the file. |

The **ClearExpireDays** function clears a logical file's expiration criteria (the `expireDays` attribute).

**Example:**

```plaintext
A := STD.File.ClearExpireDays('~samples::myscope::myfile'); //clears a file's expireDays
```

See Also: GetExpireDays, SetExpireDays
CleanAccents

STD.Uni.CleanAccents( source )

**source** A string containing the data to clean.

Return: CleanAccents returns a UNICODE value.

The **CleanAccents** function returns the `source` string with all accented characters replaced with unaccented.

Example:

```plaintext
UNICODE A := STD.Uni.CleanAccents(u'caf\u00E9');  //café - U+00E9 is lowercase e with acute
//a contains 'cafe'
```
CleanSpaces

STD.Str.CleanSpaces( source )

STD.Uni.CleanSpaces( source )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the data to clean.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>CleanSpaces returns either a STRING or UNICODE value, as appropriate.</td>
</tr>
</tbody>
</table>

All variations of the CleanSpaces function return the source string with all instances of multiple adjacent space characters (2 or more spaces together, or a tab character) reduced to a single space character. It also trims off all leading and trailing spaces.

Example:

A := STD.Str.CleanSpaces('ABCDE  ABCDE');  // A contains 'ABCDE ABCDE'
UNICODE C := STD.Uni.CleanSpaces(U'ABCDE ABCDE');  // C contains U'ABCDE ABCDE'
## CompareAtStrength

**STD.Uni.CompareAtStrength** (source1, source2, strength)

**STD.Uni.LocaleCompareAtStrength** (source1, source2, locale, strength)

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source1</td>
<td>A string containing the data to compare.</td>
</tr>
<tr>
<td>source2</td>
<td>A string containing the data to compare.</td>
</tr>
<tr>
<td>strength</td>
<td>An integer value indicating how to compare. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>1 ignores accents and case, differentiating only between letters.</td>
</tr>
<tr>
<td></td>
<td>2 ignores case but differentiates between accents.</td>
</tr>
<tr>
<td></td>
<td>3 differentiates between accents and case but ignores e.g. differences between Hiragana and Katakana</td>
</tr>
<tr>
<td></td>
<td>4 differentiates between accents and case and e.g. Hiragana/Katakana, but ignores e.g. Hebrew cantillation marks</td>
</tr>
<tr>
<td></td>
<td>5 differentiates between all strings whose canonically decomposed forms (NFD--Normalization Form D) are non-identical</td>
</tr>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
</tbody>
</table>

Return:

CompareAtStrength returns an INTEGER value.

The **CompareAtStrength** functions return zero (0) if the source1 and source2 strings contain the same data, ignoring any differences in the case of the letters. These functions return negative one (-1) if source1 < source2 or positive one (1) if source1 > source2.

Example:

```plaintext
base := u'café';   // U+00E9 is lowercase e with acute
prim := u'coffee shop'; // 1st difference, different letters
seco := u'cafe';     // 2nd difference, accents (no acute)
tert := u'Café';     // 3rd, caps (U+00C9 is u/c E + acute)

A := STD.Uni.CompareAtStrength(base, prim, 1) != 0; // base and prim differ at all strengths
A := STD.Uni.CompareAtStrength(base, seco, 1) = 0;  // base and seco same at strength 1 (differ only at strength 2)
A := STD.Uni.CompareAtStrength(base, tert, 1) = 0;  // base and tert same at strength 1 (differ only at strength 3)
A := STD.Uni.CompareAtStrength(base, seco, 2) != 0; // base and seco differ at strength 2
A := STD.Uni.CompareAtStrength(base, tert, 2) = 0;  // base and tert same at strength 2 (differ only at strength 3)
A := STD.Uni.CompareAtStrength(base, seco, 3) != 0; // base and seco differ at strength 2
A := STD.Uni.CompareAtStrength(base, tert, 3) != 0; // base and tert differ at strength 3
```
CompareIgnoreCase

STD.Str.CompareIgnoreCase( source1,source2 )

STD.Uni.CompareIgnoreCase( source1,source2 )

STD.Uni.LocaleCompareIgnoreCase( source1,source2, locale )

<table>
<thead>
<tr>
<th>source1</th>
<th>A string containing the data to compare.</th>
</tr>
</thead>
<tbody>
<tr>
<td>source2</td>
<td>A string containing the data to compare.</td>
</tr>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
</tbody>
</table>

Return: CompareIgnoreCase returns an INTEGER value.

The CompareIgnoreCase functions return zero (0) if the source1 and source2 strings contain the same data, ignoring any differences in the case of the letters. These functions return negative one (-1) if source1 < source2 or positive one (1) if source1 > source2.

Example:

A := STD.Str.CompareIgnoreCase('ABCDE','abcde');
//A contains 0 -- they "match"

B := STD.Str.CompareIgnoreCase('ABCDE','edcba');
//B contains -1 -- they do not "match"
Contains

STD.Str.Contains( source, pattern, nocase )

STD.Uni.Contains( source, pattern, nocase )

<table>
<thead>
<tr>
<th><strong>source</strong></th>
<th>A string containing the data to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>pattern</strong></td>
<td>A string containing the characters to compare. An empty string (&quot;&quot;&quot;) always returns true.</td>
</tr>
<tr>
<td><strong>nocase</strong></td>
<td>A boolean true or false indicating whether to ignore the case.</td>
</tr>
</tbody>
</table>

Return: Contains returns a BOOLEAN value.

The Contains functions return true if all the characters in the pattern appear in the source, otherwise they return false.

Example:

A := STD.Str.Contains("the quick brown fox jumps over the lazy dog", "ABCDEFGHIJKLMNOPQRSTUVWXYZ", true); //returns TRUE

B := STD.Str.Contains("the speedy ochre vixen leapt over the indolent retriever", "abcdefghijklmnopqrstuvwxyz", false); //returns FALSE -- 'z' is missing
CountWords

STD.Str.CountWords( source, separator, [allow_blank] )

STD.Uni.CountWords( source, separator, [allow_blank] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>A string containing the words to count.</td>
</tr>
<tr>
<td>separator</td>
<td>A string containing the word delimiter to use.</td>
</tr>
<tr>
<td>allow_blank</td>
<td>Optional, A BOOLEAN value indicating if empty/blank string items are included in the results. Defaults to FALSE.</td>
</tr>
</tbody>
</table>

Return: CountWords returns an integer value.

The CountWords function returns the number of words in the source string based on the specified separator.

Words are separated by one or more separator strings. No spaces are stripped from either string before matching.

Example:

```
IMPORT Std;

str1 := 'a word a day keeps the doctor away';
str2 := 'a|word|a|day|keeps|the|doctor|away';

OUTPUT(LENGTH(TRIM(Str1,LEFT,RIGHT)) - LENGTH(TRIM(Str1,ALL)) + 1);  //finds eight words by removing spaces
STD.Str.CountWords(str1,' ');  //finds eight words based on space delimiter
STD.Str.CountWords(str2,'#');  //finds eight words based on bar delimiter
```
**DecodeBase64**

STD.Str.DecodeBase64( `value` )

<table>
<thead>
<tr>
<th><code>value</code></th>
<th>A STRING value containing the data to decode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>DecodeBase64 returns a DATA value.</td>
</tr>
</tbody>
</table>

The **DecodeBase64** function returns a DATA value containing the decoded binary data.

Example:

```plaintext
IMPORT STD;
str:='AQIDBAU=';
DecodedData:= STD.Str.DecodeBase64(str);
DecodedData;
```

See Also: EncodeBase64
# EditDistance

**STD.Str.EditDistance** (string1, string2, radius)

**STD.Uni.EditDistance** (string1, string2, locale, radius)

<table>
<thead>
<tr>
<th>string1</th>
<th>The first of a pair of strings to compare.</th>
</tr>
</thead>
<tbody>
<tr>
<td>string2</td>
<td>The second of a pair of strings to compare.</td>
</tr>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
<tr>
<td>radius</td>
<td>Optional. The maximum acceptable edit distance, or 0 for no limit. Defaults to 0.</td>
</tr>
</tbody>
</table>

**Return:** EditDistance returns an UNSIGNED4 value.

The `EditDistance` function returns a standard Levenshtein distance algorithm score for the edit distance between `string1` and `string2`. This score reflects the minimum number of operations needed to transform `string1` into `string2`.

If the edit distance is larger than the `radius` it will return an arbitrary value > `radius`, but it may not be accurate. This allows the function to terminate faster if the strings are significantly different.

**Example:**

```
STD.Str.EditDistance('CAT','CAT');  //returns 0
STD.Str.EditDistance('CAT','BAT');  //returns 1
STD.Str.EditDistance('BAT','BAIT');  //returns 2
STD.Str.EditDistance('CAT','BAIT');  //returns 2
STD.Str.EditDistance('CARTMAN','BATMAN');   //returns 2
STD.Str.EditDistance('CARTMAN','BATMAN',1); //returns arbitrary number > 1
```
EditDistanceWithinRadius

STD.Str.EditDistanceWithinRadius( string1, string2, radius )
STD.Uni/EditDistanceWithinRadius( string1, string2, radius, locale )

<table>
<thead>
<tr>
<th>string1</th>
<th>The first of a pair of strings to compare.</th>
</tr>
</thead>
<tbody>
<tr>
<td>string2</td>
<td>The second of a pair of strings to compare.</td>
</tr>
<tr>
<td>radius</td>
<td>An integer specifying the maximum acceptable edit distance.</td>
</tr>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
</tbody>
</table>

Return: EditDistanceWithinRadius returns a BOOLEAN value.

The EditDistanceWithinRadius function returns TRUE if the edit distance between string1 and string2 is within the radius. The two strings are trimmed before comparison.

Example:

```
IMPORT STD;
STD.Str.EditDistance('CAT','BAIT');           //returns 2
STD.Str/EditDistanceWithinRadius('CAT','BAIT',1); //returns FALSE
STD.Str/EditDistanceWithinRadius('CAT','BAIT',2); //returns TRUE
```
**EncodeBase64**

STD.Str.EncodeBase64( value [ , insertLF ] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>A DATA value containing the data to encode.</td>
</tr>
<tr>
<td>insertLF</td>
<td>Optional. A boolean TRUE/FALSE flag indicating that, when TRUE, causes linefeeds to be inserted periodically in the output, potentially resulting in a multi-line string. If omitted, the default is TRUE.</td>
</tr>
</tbody>
</table>

Return: EncodeBase64 returns a STRING value.

The **EncodeBase64** function returns a STRING containing the binary data encoded in Base64.

Example:

```hsql
IMPORT STD;
dat:=X'0102030405';
EncodedStr:= STD.Str.EncodeBase64(dat);
EncodedStr;
```

See Also: DecodeBase64
EndsWith

STD.Str.EndsWith( src, suffix )

STD.Uni.EndsWith( src, suffix, form )

<table>
<thead>
<tr>
<th></th>
<th>The string to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>The string to find.</td>
</tr>
<tr>
<td>suffix</td>
<td>The type of Unicode normalization to be employed. (NFC, NFD, NFKC, or NFKD)</td>
</tr>
<tr>
<td>form</td>
<td>EndsWith returns a BOOLEAN value.</td>
</tr>
</tbody>
</table>

The EndsWith function returns TRUE if the src ends with the text in the suffix parameter.

Trailing and Leading spaces are stripped from the suffix before matching.

For the Unicode version, unless specified, normalization will not occur. Unless initiated as hex and then converted to Unicode using TRANSFER, ECL will perform its own normalization on your declared Unicode string.

Example:

```ecl
IMPORT STD;
STD.Str.EndsWith('a word away','away');   //returns TRUE
STD.Str.EndsWith('a word a way','away');  //returns FALSE
```
**EqualIgnoreCase**

**STD.Str.EqualIgnoreCase (source1, source2)**

<table>
<thead>
<tr>
<th>source1</th>
<th>A string containing the data to compare.</th>
</tr>
</thead>
<tbody>
<tr>
<td>source2</td>
<td>A string containing the data to compare.</td>
</tr>
</tbody>
</table>

Return: EqualIgnoreCase returns a BOOLEAN value.

The **EqualIgnoreCase** function returns TRUE if the source1 and source2 strings contain the same data, ignoring any differences in the case of the letters.

**Example:**

```plaintext
A := STD.Str.EqualIgnoreCase('ABCDE','abcde');
// A contains TRUE -- they "match"

B := STD.Str.CompareIgnoreCase('ABCDE','edcba');
// B contains FALSE -- they do not "match"
```
ExcludeFirstWord

STD.Str.ExcludeFirstWord( text )

STD.Uni.ExcludeFirstWord( text, localename )

<table>
<thead>
<tr>
<th>text</th>
<th>A string containing words separated by whitespace.</th>
</tr>
</thead>
<tbody>
<tr>
<td>localename</td>
<td>Optional. The locale to use for the break semantics. Defaults to &quot;</td>
</tr>
</tbody>
</table>

Return: ExcludeFirstWord returns a STRING or UNICODE value, as appropriate.

The **ExcludeFirstWord** function returns the `text` string with the first word removed.

Words are separated by one or more whitespace characters. For the Unicode version, words are marked by the Unicode break semantics.

Whitespace before the first word is also removed.

Example:

```plaintext
A := STD.Str.ExcludeFirstWord('The quick brown fox');
//A contains 'quick brown fox'
```
ExcludeLastWord

STD.Str.ExcludeLastWord( text )

STD.Uni.ExcludeLastWord( text, localename )

<table>
<thead>
<tr>
<th>text</th>
<th>A string containing words separated by whitespace.</th>
</tr>
</thead>
<tbody>
<tr>
<td>localename</td>
<td>Optional. The locale to use for the break semantics. Defaults to &quot;</td>
</tr>
</tbody>
</table>

Return: ExcludeLastWord returns a STRING or UNICODE value, as appropriate.

The **ExcludeLastWord** function returns the *text* string with the last word removed.

Words are separated by one or more whitespace characters. For the Unicode version, words are marked by the Unicode break semantics.

Whitespace after the last word is also removed.

Example:

```plaintext
A := STD.Str.ExcludeLastWord('The quick brown fox');
//@A contains 'The quick brown'
```
ExcludeNthWord

STD.Str.ExcludeNthWord( text, n )

STD.Uni.ExcludeNthWord( text, n, localename )

<table>
<thead>
<tr>
<th>text</th>
<th>A string containing words separated by whitespace.</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>A integer containing the ordinal position of the word to remove.</td>
</tr>
<tr>
<td>localename</td>
<td>Optional. The locale to use for the break semantics. Defaults to “”</td>
</tr>
</tbody>
</table>

Return:

ExcludeNthWord returns a STRING or UNICODE value, as appropriate.

The ExcludeNthWord function returns the text string with the nth word removed.

Words are separated by one or more whitespace characters. For the Unicode version, words are marked by the Unicode break semantics.

Trailing whitespaces are always removed with the word. Leading whitespaces are only removed with the word if the nth word is the first word.

Returns a blank string if there are no words in the source string. Returns the source string if the number of words in the string is less than the n parameter's assigned value.

Example:

```plaintext
A := STD.Str.ExcludeNthWord('The quick brown fox',2);
//A contains 'The brown fox'
```
Extract

STD.Str.Extract( source, instance )

STD.Uni.Extract( source, instance )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing a comma-delimited list of data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>An integer specifying the ordinal position of the data item within the source to return.</td>
</tr>
</tbody>
</table>

Return: Extract returns either a STRING or UNICODE value, as appropriate.

The Extract function returns the data at the ordinal position specified by the instance within the comma-delimited source string.

Example:

//all these examples result in 'Success'
A := IF(STD.Str.Extract('AB,CD,,G,E',0) = '', 'Success', 'Failure -1');
B := IF(STD.Str.Extract('AB,CD,,G,E',1) = 'AB', 'Success', 'Failure -2');
C := IF(STD.Str.Extract('AB,CD,,G,E',2) = 'CD', 'Success', 'Failure -3');
D := IF(STD.Str.Extract('AB,CD,,G,E',3) = '', 'Success', 'Failure -4');
E := IF(STD.Str.Extract('AB,CD,,G,E',4) = 'G', 'Success', 'Failure -5');
F := IF(STD.Str.Extract('AB,CD,,G,E',5) = 'E', 'Success', 'Failure -6');
G := IF(STD.Str.Extract('AB,CD,,G,E',6) = '', 'Success', 'Failure -7');
ExtractMultiple

STD.Str.ExtractMultiple( source, instance )

STD.Uni.ExtractMultiple( source, instance )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing a comma-delimited list of data.</th>
</tr>
</thead>
<tbody>
<tr>
<td>mask</td>
<td>A bitmask specifying the ordinal position of the data item within the source to return where bit 0 is item 1, bit 1 is item 2, etc..</td>
</tr>
</tbody>
</table>

Return: ExtractMultiple returns either a STRING or UNICODE value, as appropriate.

The ExtractMultiple function returns the data at the bitmask positions specified by the mask within the comma-delimited source string., where bit 0 is item 1, bit 1 is item 2, etc.

Example:

IMPORT STD;
MyTestString:= 'You, only, live, twice';
STD.Str.ExtractMultiple(MyTestString, 0b10011 ); //returns 'You, only'
Filter

STD.Str.Filter( source, filterstring )
STD.Uni.Filter( source, filterstring )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the data to filter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>filterstring</td>
<td>A string containing the characters to use as the filter.</td>
</tr>
</tbody>
</table>

Return:
Filter returns a STRING or UNICODE value, as appropriate.

The StringFilter functions return the source string with all the characters except those in the filterstring removed.

Example:

```plaintext
//all these examples result in 'Success'
A := IF(STD.Str.Filter('ADCBE', 'BD') = 'DB',
   'Success',
   'Failure - 1');
B := IF(STD.Str.Filter('ADCBE', 'BDG') = 'DBBD',
   'Success',
   'Failure - 2');
C := IF(STD.Str.Filter('ABC', '') = '',
   'Success',
   'Failure - 3');
D := IF(STD.Str.Filter('', 'BD') = '',
   'Success',
   'Failure - 4');
E := IF(STD.Str.Filter('ABCDE', 'EDCBA') = 'ABCDE',
   'Success',
   'Failure - 5');
```
FilterOut

STD.Str.FilterOut( source, filterstring )

STD.Uni.FilterOut( source, filterstring )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the data to filter.</th>
</tr>
</thead>
<tbody>
<tr>
<td>filterstring</td>
<td>A string containing the characters to use as the filter.</td>
</tr>
</tbody>
</table>

Return: FilterOut returns a STRING or UNICODE value, as appropriate.

The FilterOut functions return the source string with all the characters in the filterstring removed.

Example:

```plaintext
//all these examples result in 'Success'
A := IF(STD.Str.FilterOut('ABCDE', 'BD') = 'ACE',
    'Success',
    'Failure - 1');
B := IF(STD.Str.FilterOut('ABCDEABCDE', 'BD') = 'ACEACE',
    'Success',
    'Failure - 2');
C := IF(STD.Str.FilterOut('ABCDEABCDE', '') = 'ABCDEABCDE',
    'Success',
    'Failure - 3');
D := IF(STD.Str.FilterOut('', 'BD') = '',
    'Success',
    'Failure - 4');
```
Find

STD.Str.Find( source, target, instance )

STD.Uni.Find( source, target, instance )

STD.Uni.LocaleFind( source, target, instance, locale )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the data to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>A string containing the substring to search for.</td>
</tr>
<tr>
<td>instance</td>
<td>An integer specifying which occurrence of the target to find.</td>
</tr>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
</tbody>
</table>

Return: Find returns an INTEGER value.

The Find functions return the beginning index position within the source string of the specified instance of the target string. If the target is not found or the specified instance is greater than the number of occurrences of the target in the source, Find returns zero (0). Trailing spaces are considered to be significant when comparing.

Example:

A := IF(STD.Str.Find('ABCDE', 'BC', 1) = 2, 'Success', 'Failure - 1'); //success
B := IF(STD.Str.Find('ABCDEABCDE', 'BC', 2) = 7, 'Success', 'Failure - 2'); //success
C := IF(STD.Str.Find('ABCDEABCDE', '') = 0, 'Success', 'Failure - 3'); //syntax error, missing 3rd parameter
D := IF(STD.Str.Find('', 'BD', 1) = 0, 'Success', 'Failure - 4'); //success
FindCount

STD.Str.FindCount( src, sought )

STD.Uni.FindCount( src, sought, form )

<table>
<thead>
<tr>
<th>src</th>
<th>A string containing the data to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sought</td>
<td>A string containing the substring to search for.</td>
</tr>
<tr>
<td>form</td>
<td>The type of Unicode normalization to be employed. (NFC, NFD, NFKC, or NFKD)</td>
</tr>
</tbody>
</table>

Return: StringFindCount returns an INTEGER value.

The FindCount function returns the number of non-overlapping instances of the sought string within the src string.

Example:

A := IF(STD.Str.FindCount('ABCDE', 'BC') = 1, 'Success', 'Failure - 1'); //success

B := IF(STD.Str.FindCount('ABCDEABCDE', 'BC') = 1, 'Success', 'Failure - 1'); //failure
FindAtStrength

STD.Uni.LocaleFindAtStrength( source, target, instance, locale, strength )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>A string containing the data to search.</td>
</tr>
<tr>
<td>target</td>
<td>A string containing the substring to search for.</td>
</tr>
<tr>
<td>instance</td>
<td>An integer specifying which occurrence of the target to find.</td>
</tr>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct</td>
</tr>
<tr>
<td></td>
<td>sort order and other operations.</td>
</tr>
<tr>
<td>strength</td>
<td>An integer value indicating how to compare. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>1 ignores accents and case, differentiating only between letters</td>
</tr>
<tr>
<td></td>
<td>2 ignores case but differentiates between accents.</td>
</tr>
<tr>
<td></td>
<td>3 differentiates between accents and case but ignores e.g. differences between Hiragana and</td>
</tr>
<tr>
<td></td>
<td>Katakana.</td>
</tr>
<tr>
<td></td>
<td>4 differentiates between accents and case and e.g. Hiragana/Katakana, but ignores e.g. Hebrew</td>
</tr>
<tr>
<td></td>
<td>cantillation marks.</td>
</tr>
<tr>
<td></td>
<td>5 differentiates between all strings whose canonically decomposed forms (NFD--Normalization</td>
</tr>
<tr>
<td></td>
<td>Form D) are non-identical.</td>
</tr>
</tbody>
</table>

Return: FindAtStrength returns an INTEGER value.

The FindAtStrength function returns the beginning index position within the source string of the specified instance of the target string. If the target is not found or the specified instance is greater than the number of occurrences of the target in the source, StringFind returns zero (0).

Example:

```plaintext
base := u'caf\00E9'; // U+00E9 is lowercase e with acute
prim := u'coffee shop'; // 1st difference, different letters
seco := u'cafe';       // 2nd difference, accents (no acute)
tert := u'Caf\00C9';   // 3rd, caps (U+00C9 is u/c E + acute)
search := seco + tert + base;
STD.Uni.LocaleFindAtStrength(search, base, 1, 'fr', 1) = 1;
// at strength 1, base matches seco (only secondary diffs)
STD.Uni.LocaleFindAtStrength(search, base, 1, 'fr', 2) = 5;
// at strength 2, base matches tert (only tertiary diffs)
STD.Uni.LocaleFindAtStrength(search, base, 1, 'fr', 3) = 9;
// at strength 3, base doesn't match either seco or tert
STD.Uni.LocaleFindAtStrength(u'le caf\00E9 vert',
    u'cafe', 1, 'fr', 2) = 4;
// however, an accent on the source,
STD.Uni.LocaleFindAtStrength(u'le caf\00E9 vert',
    u'cafe', 1, 'fr', 3) = 4;
// rather than on the pattern,
STD.Uni.LocaleFindAtStrength(u'le caf\00E9 vert',
    u'cafe', 1, 'fr', 4) = 4;
// is ignored at strengths up to 4,
STD.Uni.LocaleFindAtStrength(u'le caf\00E9 vert',
    u'cafe', 1, 'fr', 5) = 0;
// and only counts at strength 5
```
FindAtStrengthReplace

STD.Uni.LocaleFindAtStrengthReplace( source, target, replacement, locale, strength )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>A string containing the data to search.</td>
</tr>
<tr>
<td>target</td>
<td>A string containing the substring to search for.</td>
</tr>
<tr>
<td>replacement</td>
<td>A string containing the replacement data.</td>
</tr>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
<tr>
<td>strength</td>
<td>An integer value indicating how to compare. Valid values are:</td>
</tr>
<tr>
<td></td>
<td>1 ignores accents and case, differentiating only between letters.</td>
</tr>
<tr>
<td></td>
<td>2 ignores case but differentiates between accents.</td>
</tr>
<tr>
<td></td>
<td>3 differentiates between accents and case but ignores e.g. differences between Hiragana and Katakana</td>
</tr>
<tr>
<td></td>
<td>4 differentiates between accents and case and e.g. Hiragana/Katakana, but ignores e.g. Hebrew cantellation marks</td>
</tr>
<tr>
<td></td>
<td>5 differentiates between all strings whose canonically decomposed forms (NFD—Normalization Form D) are non-identical</td>
</tr>
</tbody>
</table>

Return: FindAtStrengthReplace returns a UNICODE value.

The FindAtStrengthReplace functions return the source string with the replacement string substituted for all instances of the target string. If the target string is not in the source string, it returns the source string unaltered.

Example:

```
STD.Uni.LocaleFindAtStrengthReplace(u'êêëëêë', u'êë', 'xyz', 'fr', 1) = u'xyzxyzxyz';
STD.Uni.LocaleFindAtStrengthReplace(u'êêëëêë', u'êë', 'xyz', 'fr', 2) = u'êêxyzxyz';
STD.Uni.LocaleFindAtStrengthReplace(u'êêëëêë', u'êë', 'xyz', 'fr', 3) = u'êêêêêê';
```
## FindReplace

**STD.Str.FindReplace** (source, target, replacement)

**STD.Uni.FindReplace** (source, target, replacement)

**STD.Uni.LocaleFindReplace** (source, target, replacement, locale)

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the data to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>A string containing the substring to search for.</td>
</tr>
<tr>
<td>replacement</td>
<td>A string containing the replacement data.</td>
</tr>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
</tbody>
</table>

Return: FindReplace returns a STRING or UNICODE value, as appropriate.

The FindReplace functions return the source string with the replacement string substituted for all instances of the target string. If the target string is not in the source string, it returns the source string unaltered.

Example:

```plaintext
A := STD.Str.FindReplace('ABCDEABCDE', 'BC', 'XY');  // A contains 'AXYDEAXYDE'
A := STD.Uni.FindReplace(u'abcde', u'a', u'AAAAA');    // A contains u'AAAAAbcde'
A := STD.Uni.FindReplace(u'aaaaa', u'aa', u'b');       // A contains u'bba'
A := STD.Uni.FindReplace(u'aaaaaa', u'aa', u'b');      // A contains u'bbb'
A := STD.Uni.LocaleFindReplace(u'ghıklm', u'hyk', u'XxXxX', 'lt');  // A contains u'gXxXxXlm'
A := STD.Uni.LocaleFindReplace(u'gh\u0131klm', u'hyk', u'X', 'lt');  // A contains u'gXlm'
```

© 2024 HPCC Systems®. All rights reserved

100
FindWord

STD.Str.FindWord( src, word, ignore_case )

STD.Uni.FindWord( src, word, ignore_case )

<table>
<thead>
<tr>
<th>src</th>
<th>A string containing the data to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>word</td>
<td>A string containing the substring to search for.</td>
</tr>
<tr>
<td>ignore_case</td>
<td>A boolean true or false to indicate whether to ignore the case.</td>
</tr>
</tbody>
</table>

Return: FindWord returns a BOOLEAN value.

The FindWord functions return TRUE if the word string is found in src string.

Example:

```
IMPORT STD;
src := 'Now is the winter of our discontent';
word := 'now';
STD.Str.FindWord(src,word);       // false - case not ignored
STD.Str.FindWord(src,word,TRUE);  // true - with case ignored word is found
```
FromHexPairs

STD.Str.FromHexPairs( source )

<table>
<thead>
<tr>
<th>source</th>
<th>The string containing the hex pairs to process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>FromHexPairs returns a data value with each byte created from a pair of hex digits.</td>
</tr>
</tbody>
</table>

The **FromHexPairs** function returns a data value with each byte created from a pair of hex digits.

Example:

```plaintext
A := STD.Str.FromHexPairs('0001FF80');
```
GetNthWord

STD.Str.GetNthWord( source, instance )

STD.Uni.GetNthWord ( source, instance [, locale] )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the space-delimited words.</th>
</tr>
</thead>
<tbody>
<tr>
<td>instance</td>
<td>An integer specifying the word to return.</td>
</tr>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
</tbody>
</table>

Return: GetNthWord returns a string value.

The GetNthWord function returns the word in the instance position in the source string.

Example:

```plaintext
IMPORT Std;

str1 := 'a word a day keeps the doctor away';

STD.Str.GetNthWord(str1,2);  //returns "word"
```
RemoveSuffix

STD.Str.RemoveSuffix( src, suffix )

STD.Uni.RemoveSuffix( src, suffix, form )

<table>
<thead>
<tr>
<th>src</th>
<th>The string to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>suffix</td>
<td>The ending string to remove.</td>
</tr>
<tr>
<td>form</td>
<td>The type of Unicode normalization to be employed. (NFC, NFD, NFKC, or NFKD)</td>
</tr>
</tbody>
</table>

Return: RemoveSuffix returns a string value.

The RemoveSuffix function returns the src string with the ending text in the suffix parameter removed. If the src string does not end with the suffix, then the src string is returned unchanged. Trailing spaces are stripped from both strings before matching.

Example:

```hpe
IMPORT STD;
STD.Str.RemoveSuffix('a word away','away');  //returns 'a word'
STD.Str.RemoveSuffix('a word a way','away');  //returns 'a word a way'
```
# Repeat

\[ \text{STD.Str.Repeat( } \text{text, } n \text{ )} \]

\[ \text{STD.Uni.Repeat( } \text{text, } n \text{ )} \]

<table>
<thead>
<tr>
<th>text</th>
<th>The string to be repeated (maximum length is 255 characters).</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>The number of repetitions.</td>
</tr>
</tbody>
</table>

Return:
Repeat returns a STRING containing \( n \) concatenations of the string text.

The **Repeat** function returns the `text` string repeated \( n \) times.

Example:

\[
A := \text{STD.Str.Repeat('ABC',3);} \quad // A \text{ contains 'ABCABCABC'}
\]
Reverse

STD.Str.Reverse( source )

STD.Uni.Reverse( source )

<table>
<thead>
<tr>
<th>parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>A string containing the data to reverse.</td>
</tr>
</tbody>
</table>

Return: Reverse returns a STRING or UNICODE value, as appropriate.

The Reverse functions return the source string with all characters in reverse order.

Example:

A := STD.Str.Reverse('ABCDE'); //A contains 'EDCBA'
SplitWords

STD.Str.SplitWords( src, separator [ , allow_blank ] )
STD.Uni.SplitWords( src, separator [ , allow_blank ] )

<table>
<thead>
<tr>
<th>src</th>
<th>A string containing the words to extract.</th>
</tr>
</thead>
<tbody>
<tr>
<td>separator</td>
<td>A string containing the word delimiter to use.</td>
</tr>
<tr>
<td>allow_blank</td>
<td>Optional. If TRUE, specifies allowing blank items in the result. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>

Return: SplitWords returns a SET OF STRING or a UnicodeSet, as appropriate.

The SplitWords function returns the list of words in the src string split out by the specified separator. No spaces are stripped from either string before matching.

Example:

```plaintext
IMPORT Std;

str1 := 'a word a day keeps the doctor away';
str2 := 'a|word|a|day|keeps|the|doctor|away';

STD.Str.SplitWords(str1,' ');
  //returns ['a', 'word', 'a', 'day', 'keeps', 'the', 'doctor', 'away']

STD.Str.SplitWords(str2,'|');
  //returns ['a', 'word', 'a', 'day', 'keeps', 'the', 'doctor', 'away']
```
### SubstituteExcluded

**STD.Str.SubstituteExcluded** *(source, target, replacement)*

**STD.Uni.SubstituteExcluded** *(source, target, replacement)*

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the data to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>A string containing the characters to search for.</td>
</tr>
<tr>
<td>replacement</td>
<td>A string containing the replacement character as its first character.</td>
</tr>
</tbody>
</table>

Return: SubstituteExcluded returns a STRING or UNICODE value, as appropriate.

The **SubstituteExcluded** functions return the *source* string with the *replacement* character substituted for all characters except those in the *target* string. If the *target* string is not in the *source* string, it returns the *source* string with all characters replaced by the *replacement* character.

**Example:**

```
IMPORT STD;
A := STD.Uni.SubstituteExcluded(u'abcdeabcdec', u'cd', u'x');
//A contains u'xxcdxxcdxc';
```
SubstituteIncluded

STD.Str.SubstituteIncluded( source, target, replacement )

STD.Uni.SubstituteIncluded( source, target, replacement )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the data to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>A string containing the characters to search for.</td>
</tr>
<tr>
<td>replacement</td>
<td>A string containing the replacement character as its first character.</td>
</tr>
</tbody>
</table>

Return:

SubstituteIncluded returns a STRING or UNICODE value, as appropriate.

The SubstituteIncluded functions return the source string with the replacement character substituted for all characters that exist in both the source and the target string. If no target string characters are in the source string, it returns the source string unaltered.

Example:

```
IMPORT STD;
A := STD.Uni.SubstituteIncluded(u'abcde', u'cd', u'x');
  //A contains u'abxxe';
B := STD.Str.SubstituteIncluded('abcabc','ac','yz');
  //B contains 'ybyyby'
```
StartsWith

STD.Str.StartsWith( src, prefix )

STD.Uni.StartsWith( src, prefix, form )

<table>
<thead>
<tr>
<th>src</th>
<th>The string to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>prefix</td>
<td>The string to find.</td>
</tr>
<tr>
<td>form</td>
<td>The type of Unicode normalization to be employed. (NFC, NFD, NFKC, or NFKD)</td>
</tr>
</tbody>
</table>

Return: StartsWith returns a BOOLEAN value.

The **StartsWith** function returns TRUE if the src starts with the text in the prefix parameter.

Trailing and Leading spaces are stripped from the prefix before matching.

For the Unicode version, unless specified, normalization will not occur. Unless initiated as hex and then converted to Unicode using TRANSFER, ECL will perform its own normalization on your declared Unicode string.

Example:

```ecl
IMPORT STD;
STD.Str.StartsWith('a word away','a word');   //returns TRUE
STD.Str.StartsWith('a word away','aword');   //returns FALSE
```
ToHexPairs

STD.Str.ToHexPairs( source )

<table>
<thead>
<tr>
<th>source</th>
<th>The data value that should be expanded as a sequence of hex pairs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>ToHexPairs returns a string containing a sequence of hex pairs.</td>
</tr>
</tbody>
</table>

The ToHexPairs function Converts the data value to a sequence of hex pairs.

Example:

A := STD.Str.ToHexPairs(D'\000\001\377\200');
ToLowerCase

STD.Str.ToLowerCase( source )

STD.Uni.ToLowerCase( source )

STD.Uni.LocaleToLowerCase( source, locale )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the data to change case.</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
<tr>
<td>Return:</td>
<td>ToLowerCase returns a STRING or UNICODE value, as appropriate.</td>
</tr>
</tbody>
</table>

The ToLowerCase functions return the source string with all upper case characters converted to lower case.

Example:

A := STD.Str.ToLowerCase('ABCDE'); //A contains 'abcde'
ToTitleCase

STD.Str.ToTitleCase( source )

STD.Uni.ToTitleCase( source )

STD.Uni.LocaleToTitleCase( source, locale )

| source | A string containing the data to change case. |
|.locale | A null-terminated string containing the language and country code to use to determine correct sort order and other operations. |

Return: ToTitleCase returns a STRING or UNICODE value, as appropriate.

The ToTitleCase functions return the source string with the first letter of each word in upper case and all other letters lower cased.

Example:

A := STD.Str.ToTitleCase('ABCDE ABCDE '); //A contains 'Abcde Abcde'
B := STD.Str.ToTitleCase('john smith-jones'); //B contains 'John Smith-Jones'
ToUpperCase

STD.Str.ToUpperCase( source )

STD.Uni.ToUpperCase( source )

STD.Uni.LocaleToUpperCase( source, locale )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the data to change case.</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
</tbody>
</table>

Return: ToUpperCase returns a STRING value.

The ToUpperCase functions return the source string with all lower case characters converted to upper case.

Example:

A := STD.Str.ToUpperCase('abcde');
//A contains 'ABCDE'
Translate

STD.Str.Translate( src, search, replacement )

STD.Uni.Translate( src, search, replacement )

<table>
<thead>
<tr>
<th>src</th>
<th>A string containing the characters to search.</th>
</tr>
</thead>
<tbody>
<tr>
<td>search</td>
<td>A string containing the characters to be replaced by characters in the replacement string.</td>
</tr>
<tr>
<td>replacement</td>
<td>A string containing the characters to act as replacements.</td>
</tr>
</tbody>
</table>

Return:

Translate returns a STRING or UNICODE value, as appropriate.

The Translate functions return the src string with the replacement character substituted for all characters in the src string. The search string characters are replaced by the characters in the equivalent position in the replacement string.

If no search string characters are in the src string, it returns the src string unaltered.

Example:

IMPORT STD;
A := STD.Str.Translate('abcabc','ca','yz'); //A contains 'zbyzby'
**Version**

STD.Uni.Version()

| Return | Version returns a STRING value (e.g., '55.1'). |

The **Version** function returns the version of the International Components for Unicode (ICU) library installed.

Example:

```plaintext
IMPORT STD;
//Chinese dictionary based iterators added in ICU 50
OUTPUT(UniTest);
```
WildMatch

STD.Str.WildMatch( source, pattern, nocase )

STD.Uni.WildMatch( source, pattern, nocase )

| source | A string containing the data to search. |
| pattern | A string containing the wildcard expression to match. Valid wildcards are ? (single character) and * (multiple character). |
| nocase | A boolean true or false indicating whether to ignore the case. |
| Return: | WildMatch returns a BOOLEAN value. |

The WildMatch function returns TRUE if the pattern matches the source.

The case-insensitive version of the Unicode WildMatch has been optimized for speed over accuracy. For accurate case-folding, you should either use the Unicode ToUpperCase function explicitly and then a case-sensitive the Unicode WildMatch, or use REGEXFIND.

Example:

STD.Str.wildmatch('abcdeabcdec', 'a?c*', false) = TRUE;
WordCount

STD.Str.WordCount( source )

STD.Uni.WordCount( source [, locale ] )

<table>
<thead>
<tr>
<th>source</th>
<th>A string containing the words to count. Words are delimited by spaces.</th>
</tr>
</thead>
<tbody>
<tr>
<td>locale</td>
<td>A null-terminated string containing the language and country code to use to determine correct sort order and other operations.</td>
</tr>
</tbody>
</table>

Return: WordCount returns an integer value.

The **WordCount** function returns the number of words in the *source* string.

Example:

```plaintext
IMPORT Std;

str1 := 'a word a day keeps the doctor away';

OUTPUT(LENGTH(TRIM(Str1,LEFT,RIGHT)) - LENGTH(TRIM(Str1,ALL)) + 1);
    //finds eight words by removing spaces

STD.Str.WordCount(str1);           //finds eight words based on space delimiter
```
Metaphone Support

These functions provide a means to implement Double Metaphone or Metaphone 3 phonetic encoding or fuzzy-match algorithms which return a primary code, a secondary code, or both for a given string.
Primary

STD.Metaphone.Primary( source )

STD.Metaphone3.Primary( source )

<table>
<thead>
<tr>
<th>source</th>
<th>The string to process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>Primary returns a STRING value.</td>
</tr>
</tbody>
</table>

The **Primary** function returns a textual representation of the source data, similar to a Soundex code. This function returns the first return value from the Double Metaphone algorithm.

The **Metaphone3.Primary** function uses the newer Metaphone 3 libraries which improve phonetic encoding of English words, non-English words familiar to Americans, and first and last names commonly found in the United States (Enterprise Edition only).

Example:

```plaintext
r := RECORD
    STRING source;
    STRING M1;
    STRING M2;
    STRING Mboth;
END;

r XF(ProgGuide.Person.File L) := TRANSFORM
    SELF.source := L.LastName;
    SELF.M1     := STD.Metaphone.Primary( L.LastName );
    SELF.M2     := STD.Metaphone.Secondary( L.LastName );
    SELF.Mboth  := STD.Metaphone.Double( L.LastName );
END;

// Example using Metaphone 3 (available in Enterprise Edition)
/*
r XF(ProgGuide.Person.File L) := TRANSFORM
    SELF.source := L.LastName;
    SELF.M1     := STD.Metaphone3.Primary( L.LastName );
    SELF.M2     := STD.Metaphone3.Secondary( L.LastName );
    SELF.Mboth  := STD.Metaphone3.Double( L.LastName );
END;
*/

ds := PROJECT(ProgGuide.Person.File,XF(LEFT));

COUNT(ds);
COUNT(ds(M1 <> M2));
OUTPUT(ds);
OUTPUT(ds(M1 <> M2));
```
### Secondary

**STD.Metaphone.Secondary**(`source`)  
**STD.Metaphone3.Secondary**(`source`)

<table>
<thead>
<tr>
<th><code>source</code></th>
<th>The string to process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Secondary returns a STRING value.</td>
</tr>
</tbody>
</table>

The **Secondary** function returns a textual representation of the source data, similar to a Soundex code. This function returns the second return value from the Double Metaphone algorithm.

The **Metaphone3.SecondaryPrimary** function uses the newer Metaphone 3 libraries which improve phonetic encoding of English words, non-English words familiar to Americans, and first and last names commonly found in the United States (Enterprise Edition only).

**Example:**

```plaintext
r := RECORD  
  STRING source;  
  STRING M1;  
  STRING M2;  
  STRING Mboth;  
END;

r XF(ProgGuide.Person.File L) := TRANSFORM  
  SELF.source := L.LastName;  
  SELF.M1     := STD.Metaphone.Primary( L.LastName );  
  SELF.M2     := STD.Metaphone.Secondary( L.LastName );  
  SELF.Mboth  := STD.Metaphone.Double( L.LastName );
END;

// Example using Metaphone 3 (available in Enterprise Edition)  
/*
r XF(ProgGuide.Person.File L) := TRANSFORM  
  SELF.source := L.LastName;  
  SELF.M1     := STD.Metaphone3.Primary( L.LastName );  
  SELF.M2     := STD.Metaphone3.Secondary( L.LastName );  
  SELF.Mboth  := STD.Metaphone3.Double( L.LastName );
END;
*/

ds := PROJECT(ProgGuide.Person.File,XF(LEFT));

COUNT(ds);  
COUNT(ds(M1 <> M2));  
OUTPUT(ds);  
OUTPUT(ds(M1 <> M2));
```
Double

STD.Metaphone.Double( source )

STD.Metaphone3.Double( source )

<table>
<thead>
<tr>
<th>source</th>
<th>The string to process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Double returns a STRING value.</td>
</tr>
</tbody>
</table>

The Double function returns a textual representation of the source data, similar to a Soundex code. This function returns both return values from the Double Metaphone algorithm, concatenating the two into a single result string.

The Metaphone3.Double function uses the newer Metaphone 3 libraries which improve phonetic encoding of English words, non-English words familiar to Americans, and first and last names commonly found in the United States (Enterprise Edition only).

Example:

```plaintext
r := RECORD
    STRING source;
    STRING M1;
    STRING M2;
    STRING Mboth;
END;

r XF(ProgGuide.Person.File L) := TRANSFORM
    SELF.source := L.LastName;
    SELF.M1 := STD.Metaphone.Primary( L.LastName );
    SELF.M2 := STD.Metaphone.Secondary( L.LastName );
    SELF.Mboth := STD.Metaphone.Double( L.LastName );
END;

// Example using Metaphone 3 (available in Enterprise Edition)
/*
r XF(ProgGuide.Person.File L) := TRANSFORM
    SELF.source := L.LastName;
    SELF.M1 := STD.Metaphone3.Primary( L.LastName );
    SELF.M2 := STD.Metaphone3.Secondary( L.LastName );
    SELF.Mboth := STD.Metaphone3.Double( L.LastName );
END;
*/

ds := PROJECT(ProgGuide.Person.File,XF(LEFT));

COUNT(ds);
COUNT(ds(M1 <> M2));
OUTPUT(ds);
OUTPUT(ds(M1 <> M2));
```
Cryptography Support

This section provides support to perform cryptographic functions on data in ECL.
Cryptographic Library Overview

There are three classes of Cryptographic Algorithms in the Cryptography library: Hashing functions, Symmetric-Key Algorithms, and Asymmetric-Key Algorithms.

**Hashing Functions:**
- Useful to verify data integrity
- Transforms large random sized data to small fixed size
- Impossible to reverse a hash back to its original data (one-way)
- Fast

See Also: SupportedHashAlgorithms

**Symmetric-Key Algorithms:**
- Uses a single shared key to Encrypt/Decrypt data
- Supports Block algorithms
- Fast

See Also: SupportedSymmetricCipherAlgorithms

**Asymmetric-Key Algorithms** (Also known as Public-Key or PKI Algorithms):
- Mathematically associated Public and Private Key Pair
- Used to Encrypt/Decrypt data
- Used to create Digital Signatures
- Comparatively slower

See Also: SupportedPublicKeyAlgorithms
SupportedHashAlgorithms

STD.Crypto_SUPPORTEDHASHALGORITHMS();

Return: SET OF STRINGs containing all supported Hash Algorithms

The SupportedHashAlgorithms function returns the set of supported Hash Algorithms

Example:

IMPORT STD;
STD.Crypto.SupportedHashAlgorithms(); //returns SET of STRINGs
SupportedSymmetricCipherAlgorithms

STD.Crypto.SupportedSymmetricCipherAlgorithms();

Return: SET OF STRINGs containing all supported Cipher Algorithms

The SupportedSymmetricCipherAlgorithms function returns the set of supported Cipher Algorithms

Example:

IMPORT STD;
STD.Crypto.SupportedSymmetricCipherAlgorithms(); //returns SET of STRINGs
**SupportedPublicKeyAlgorithms**

STD.Crypto.SupportedPublicKeyAlgorithms();

<table>
<thead>
<tr>
<th>Return</th>
<th>SET OF STRINGs containing all supported Public Key Algorithms</th>
</tr>
</thead>
</table>

The **SupportedPublicKeyAlgorithms** function returns the set of supported Public Key Algorithms.

Example:

```plaintext
IMPORT STD;
STD.Crypto.SupportedPublicKeyAlgorithms(); //returns SET of STRINGs
```
Hashing Module

\( myHashModule := \text{STD.Crypto.Hashing}(hashAlgorithm); \)

<table>
<thead>
<tr>
<th>myHashModule</th>
<th>The name of the Hashing module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>hashAlgorithm</td>
<td>The hashing algorithm to use, as returned by SupportedHashAlgorithms()</td>
</tr>
</tbody>
</table>

A Hashing module is defined in ECL. Subsequent function definitions use the module definitions specified in the Hashing module definition.

Example:

```ecl
Import STD;

//Hashing module definition
myHashModuleSha512 := Std.Crypto.Hashing('sha512');
myHashModuleSha256 := Std.Crypto.Hashing('sha256');

DATA hash1 := myHashModuleSha512.Hash((DATA)'The quick brown fox jumps over the lazy dog');
DATA hash2 := myHashModuleSha256.Hash((DATA)'The quick brown fox jumps over the lazy dog');

OUTPUT(hash1);
OUTPUT(hash2);
```
Hash

`myHashModule.Hash(inputData);`

<table>
<thead>
<tr>
<th><strong>myHashModule</strong></th>
<th>The name of the Hashing module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inputData</strong></td>
<td>The data to hash in DATA format</td>
</tr>
<tr>
<td><strong>Return:</strong></td>
<td>Hashed contents in DATA format</td>
</tr>
</tbody>
</table>

The Hash function creates a hash of the given `inputData`, using the hash algorithm defined in the Hashing module.

Example:

```plaintext
Import STD;

// Hashing module definition
myHashModuleSha512 := Std.Crypto.Hashing('sha512');
myHashModuleSha256 := Std.Crypto.Hashing('sha256');

DATA hash1 := myHashModuleSha512.Hash((DATA)'The quick brown fox jumps over the lazy dog');
DATA hash2 := myHashModuleSha256.Hash((DATA)'The quick brown fox jumps over the lazy dog');

OUTPUT(hash1);
OUTPUT(hash2);
```
SymmetricEncryption Module

\[ \text{mySymEncModule} := \text{STD.Crypto.SymmetricEncryption} (\text{algorithm}, \text{passphrase}); \]

<table>
<thead>
<tr>
<th>mySymEncModule</th>
<th>The name of the Symmetric Encryption module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{algorithm}</td>
<td>The algorithm to use, as returned by SupportedSymmetricCipherAlgorithms()</td>
</tr>
<tr>
<td>\text{passphrase}</td>
<td>The passphrase to use for encryption/decryption</td>
</tr>
</tbody>
</table>

A Symmetric Encryption module is defined in ECL. Subsequent function definitions use the options specified in the Symmetric Encryption module definition.

Example:

```ecl
IMPORT STD;

//Symmetric Encryption module definition
mySymEncModule := Std.Crypto.SymmetricEncryption('aes-256-cbc', '12345678901234567890123456789012');

//encrypt/decrypt string literals
STRING myStr := 'The quick brown fox jumps over the lazy dog';
DATA encryptedStr := mySymEncModule.Encrypt((DATA)myStr);
STRING decryptedStr := (STRING)mySymEncModule.Decrypt(encryptedStr);

OUTPUT(myStr);
OUTPUT(decryptedStr);
```
Encrypt (Symmetric)

```
mySymEncModule.Encrypt(inputData);
```

<table>
<thead>
<tr>
<th><code>mySymEncModule</code></th>
<th>The name of the Symmetric Encryption module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inputData</code></td>
<td>The data to encrypt in DATA format</td>
</tr>
<tr>
<td>Return:</td>
<td>Encrypted contents in DATA format</td>
</tr>
</tbody>
</table>

The Encrypt function encrypts the given `inputData`, using the options defined in the Symmetric Encryption module.

Example:

```
IMPORT STD;

//Symmetric Encryption module definition
mySymEncModule := Std.Crypto.SymmetricEncryption('aes-256-cbc',
'12345678901234567890123456789012');

//encrypt/decrypt string literals
STRING myStr := 'The quick brown fox jumps over the lazy dog';
DATA encryptedStr := mySymEncModule.Encrypt((DATA)myStr);
STRING decryptedStr := (STRING)mySymEncModule.Decrypt(encryptedStr);

OUTPUT(myStr);
OUTPUT(decryptedStr);
```
Decipher (Symmetric)

```
mySymEncModule.Decrypt(encryptedData);
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mySymEncModule</code></td>
<td>The name of the Symmetric Encryption module structure</td>
</tr>
<tr>
<td><code>encryptedData</code></td>
<td>The data to decrypt in DATA format</td>
</tr>
</tbody>
</table>

Return: Decrypted contents in DATA format

The Decrypt function decrypts the given `encryptedData` using the options defined in the Symmetric Encryption module. You can only decrypt data that was encrypted by the Standard Library’s Encrypt method.

Example:

```hs
IMPORT STD;

//Symmetric Encryption module definition
mySymEncModule := Std.Crypto.SymmetricEncryption('aes-256-cbc',
                                              '12345678901234567890123456789012');

//encrypt/decrypt string literals
STRING myStr := 'The quick brown fox jumps over the lazy dog';
DATA encryptedStr := mySymEncModule.Encrypt((DATA)myStr);
STRING decryptedStr := (STRING)mySymEncModule.Decrypt(encryptedStr) ;

OUTPUT(myStr);
OUTPUT(decryptedStr);
```
Public Key Encryption Module

```ecl
myPKEModule := STD.Crypto.PublicKeyEncryption(pkAlgorithm, publicKeyFile, privateKeyFile, passphrase);
```

<table>
<thead>
<tr>
<th>myPKEModule</th>
<th>The name of the Public Key Encryption module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>pkAlgorithm</td>
<td>The algorithm to use, as returned by SupportedPublicKeyAlgorithms()</td>
</tr>
<tr>
<td>publicKeyFile</td>
<td>The File Specification of the PEM formatted Public Key file</td>
</tr>
<tr>
<td>privateKeyFile</td>
<td>The File Specification of the PEM formatted Private Key file</td>
</tr>
<tr>
<td>passphrase</td>
<td>The passphrase to use for encryption, decryption, signing, verifying</td>
</tr>
</tbody>
</table>

A Public Key Encryption module is defined in ECL. Subsequent function definitions use the options defined in the Public Key Encryption module to perform asymmetric encryption/decryption/digital signing/signature verification.

Example:

```ecl
IMPORT STD;

privKeyFile := '/var/lib/HPCCSystems/myesp/test.key';
pubKeyFile := '/var/lib/HPCCSystems/myesp/test.key.pub';

myPKEModule := STD.Crypto.PublicKeyEncryption('RSA', pubKeyFile, privKeyFile, '');

DATA encrypted := myPKEModule.Encrypt((DATA)'The quick brown fox jumps over the lazy dog');

OUTPUT( (STRING)myPKEModule.Decrypt(encrypted) );
```
Encrypt (PKE)

`myPKEModule.Encrypt(inputData);`

<table>
<thead>
<tr>
<th><code>myPKEModule</code></th>
<th>The name of the Public Key Encryption module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inputData</code></td>
<td>The data to encrypt in DATA format</td>
</tr>
</tbody>
</table>

Return: Encrypted contents in DATA format

The Encrypt function encrypts the given `inputData` using the options specified in the Public Key Encryption module definition.

Example:

```plaintext
IMPORT STD;
privKeyFile := '/var/lib/HPCCSystems/myesp/test.key';
pubKeyFile := '/var/lib/HPCCSystems/myesp/test.key.pub';

//PKE Encryption module definition
myPKEModule := STD.Crypto.PublicKeyEncryption('RSA', pubKeyFile, privKeyFile,'');

DATA encrypted := myPKEModule.Encrypt((DATA)'The quick brown fox jumps over the lazy dog');

OUTPUT( (STRING)myPKEModule.Decrypt(encrypted) );
```
Decrypt (PKE)

\( \text{myPKEModule.decrypt(encryptedData);} \)

<table>
<thead>
<tr>
<th>\text{myPKEModule}</th>
<th>The name of the Public Key Encryption module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{encryptedData}</td>
<td>The data to decrypt in DATA format</td>
</tr>
<tr>
<td>Return:</td>
<td>Decrypted contents in DATA format</td>
</tr>
</tbody>
</table>

The Decrypt function decrypts the given \text{encryptedData}, using the options specified in the Public Key Encryption module definition. You can only decrypt data that was encrypted by the Standard Library's Encrypt method.

Example:

\[
\begin{align*}
\text{IMPORT STD;} \\
\text{privKeyFile := '/var/lib/HPCCSystems/myesp/test.key';} \\
\text{pubKeyFile := '/var/lib/HPCCSystems/myesp/test.key.pub';} \\
\text{//PKE Encryption module definition} \\
\text{myPKEModule := STD.Crypto.PublicKeyEncryption('RSA', pubKeyFile, privKeyFile,'');} \\
\text{DATA encrypted := myPKEModule.Encrypt((DATA)'The quick brown fox jumps over the lazy dog');} \\
\text{OUTPUT((STRING)myPKEModule.Decrypt(encrypted));}
\end{align*}
\]
Sign (PKE)

mySymEncModule.Sign(encryptedData);

<table>
<thead>
<tr>
<th>myPKEModule</th>
<th>The name of the Public Key Encryption module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputData</td>
<td>The data to sign in DATA format</td>
</tr>
</tbody>
</table>

Return: Computed Digital signature

The Sign function creates a digital signature of the given inputData, using the options specified in the Public Key Encryption module definition.

Example:

```plaintext
import STD;

privKeyFile := '/var/lib/HPCCSystems/myesp/test.key';
pubKeyFile := '/var/lib/HPCCSystems/myesp/test.key.pub';

//PKE Encryption module definition
myPKEModule := STD.Crypto.PublicKeyEncryption('RSA', pubKeyFile, privKeyFile,'');

DATA signature := myPKEModule.Sign((DATA)'The quick brown fox jumps');
OUTPUT(TRUE = myPKEModule.VerifySignature(signature, (DATA)'The quick brown fox jumps'));
```
VerifySignature (PKE)

`myPKEModule.VerifySignature(signature, signedData);`

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>myPKEModule</code></td>
<td>The name of the Public Key Encryption module structure</td>
</tr>
<tr>
<td><code>signature</code></td>
<td>The Digital signature to verify</td>
</tr>
<tr>
<td><code>signedData</code></td>
<td>Data used to create the signature in DATA format</td>
</tr>
</tbody>
</table>

Return: A BOOLEAN value to indicate verification

The `VerifySignature` function verifies the given digital `signature` using the options specified in the Public Key Encryption module definition.

Example:

```plaintext
IMPORT STD;
privKeyFile := '/var/lib/HPCCSystems/myesp/test.key';
pubKeyFile := '/var/lib/HPCCSystems/myesp/test.key.pub';

//PKE Encryption module definition
myPKEModule := STD.Crypto.PublicKeyEncryption('RSA', pubKeyFile, privKeyFile,'');

DATA signature := myPKEModule.Sign((DATA)'The quick brown fox jumps');
OUTPUT(TRUE = myPKEModule.VerifySignature(signature, (DATA)'The quick brown fox jumps'));
```
myPKEModule := STD.Crypto.PublicKeyEncryptionFromBuffer(pkAlgorithm, publicKeyFile, privateKeyFile, passphrase);

myPKEModule, The name of the Public Key Encryption From Buffer module structure

pkAlgorithm, The algorithm to use, as returned by SupportedPublicKeyAlgorithms()

publicKeyBuff, PEM formatted Public Key buffer

privateKeyBuff, PEM formatted Private Key buffer

passphrase, The passphrase to use for encryption, decryption, signing, verifying

A Public Key Encryption From Buffer module is defined in ECL. Subsequent function definitions use the options defined in the Public Key Encryption From Buffer module to perform asymmetric encryption/decryption/digital signing/signature verification.

Example:

```haskell
IMPORT STD;
STRING publicKey := '-----BEGIN PUBLIC KEY-----' + '
' + 'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KnMwRAOafo+' + '\n' + 'vod+3AUS/1DngT39j3iov9a12N8g4W5ipqfKtReSm2q61/T1UQcmi42soUXmCee+' + '\n' + 'BHq1MDydw9aHOQG17CB0GYw31f6f12o77RC7ocQ63cRzhO68K0ryW63efWnMoy+' + '\n' + 'hr9R9xMclDf+3WM/X0IX0Aps8s7kuVPV9Gy42O+1FVO/XNwRjukJG5ASuXpQaVQv+' + '\n' + '/o9/j6q7k1nEiVhLiGc3m3PtW666zVz3h/huLbEXvEy50j2fQaC5E635mpdk/CW3' + '\n' + 'J67kNTN3sYsWz2/2/Z0wnxZrdr79Xc/59b18B/a3280eM2PfZwPmVnVvU0' + '\n' + '/wIBAQAB' + '\n' + '-----END PUBLIC KEY-----';

STRING privateKey := '-----BEGIN RSA PRIVATE KEY-----' + '
' + 'MIIEowIBAAKCAQEAr64RncTp5pV0KnMwRAOafo+3AUS/1DngT39j3iov9a12N8g' + '\n' + '4W5ipqfKtReSm2q61/T1UQcmi42soUXmCee+3WM/X0IX0Aps8s7kuVPV9Gy42O+' + '\n' + '80Ls07cQ0e3OCrE0k6gl0kF370wV63e3FwM0yEyFR9M66lCldf+3WM/X0IX0Aps8s' + '\n' + 'VPGV4+yRd2+1FVO/XNwRjukJG5ASuXpQaVQv/o9/j6q7k1nEiVhLiGc3m3PtW666z' + '\n' + '62Vz3h/huLbEXvEy50j2fQaC5E635mpdk/CW3J67kNTN3sYsWz2/2/Z0wnxZrdr79Xc' + '\n' + 'goNL69/59P18B/a3280eM2PfZwPmVnVvU0/bwIDAQABoIABAQCNAT6KYO00u8Wv' + '\n' + 'F50id3AKnwyPycF21ILWq3vAcFUI7qle+SMRy+kb6Le/+u5R5UqH/HSc9Jm3IaJW' + '\n' + 'V1lyrV2d3VfYJCKzorcXYSz0OMau5g6B777VNUA05Gt3c6FCA61l5C5C1oEom' + '\n' + 'GCK6U18m9K69L6Fq66kEEo2yyDuN7O4BxrPMBM61UHcHx2tc6b2yV8zz+Et1ADG' + '\n' + '70pRw2M6zc+cR+416mvw1Yg0vU6L5j3q66U65lPZhm3/HQFGwsaFGQ6B2' + '\n' + 'Z9HjK5WBYT75rB1LKhKeC61JW+d5hrszgTdk79F7wA1zK6CIdyHe2aANAzUc/0' + '\n' + 'sBoCHeUAr0gB0AXC1Av66zF2Fq+60MBXCT4m7dCJnAc+qhteF249uOlBeF3dsDp' + '\n' + 'f0hjnhV0O0V81YBb/Z1yRJDU5G6mh/2BYSkydelaLkojTxeKR2ba4xq5dJt5DpoF' + '\n' + '0c15A1t07gk02nxP/azDICYCY/cvdIdU9L9po1zthuf1jgw26G6acEnanoGBAMzn' + '\n' + 'lWRUS3dbqemoc+c48W5rS3/q5ss1G81S22Y8W+oYW7hINA506aacc8YFEWE2RBjDjx' + '\n' + 'YEAD8bnsDzdzo00JEj3vVNZt6nx8Q3BOOrYKnqHvI/HO3V3V6kedUKdB8R7Kc' + '\n' + 'eC2zO1VcCeZtsHuU40t7NrhDgHx7TV+JlVzObQarAgAI5c0D/nu66nryP8re5D' + '\n' + 'tw67Ktxx12r8C8CoQua/Au5kjeE/4DxtXr09QvmAcivERN/sZBiZnjf52s5wv' + '\n' + 'wPB2/3PMKVYkaVbd1cN3rvNHGUKzawbxBSrzmBQ5/YmCwrDauYchnEcmWMCuU9' + '\n' + '87I15s3J69pOix1HYDdV9CvY1bY6762nuQnmpy9npd/H81iJxuil7z31HTu4MLMR' + '\n' + '9DMaJkxx3ds5olu3zE+N8P17UVa06Rj3B1vLg4+MXY1DBH/0sDDpm+IzgCNDR' + '\n' + 'spmmn965boBsbWmAzK09/qrt/2FszLO4JLsFmTPU/y58b/dhjY06DFQop4R4zF' + '\n' + 'ICNQXQZQf7KZ0OQV4Ynhk1U1WYHns815O7Wtsa51Q+k1b08PDr9rql3Lzyk' + '\n' + 'tKf8e6w+rk7xij8G8KZ/vUdQl3fI1mByEjJHRQ81W5+K59hzn101escjQoavw9s' + '\n' + 'ih35sueb6xoj/N0luXaBoUT5acXKv9HRE74eqy0U6M+qxUCoJj9R' + '\n' + '-----END RSA PRIVATE KEY-----';

// PKE Encryption module definition
MyPKEModule := STD.Crypto.PublicKeyEncryptionFromBuffer('RSA', PublicKey, PrivateKey, '');
```
DATA encrypted := MyPKEModule.Encrypt((DATA)'The quick brown fox jumps over the lazy dog');
OUTPUT( (STRING)MyPKEModule.Decrypt(encrypted));
Encrypt (PKE From Buffer)

```plaintext
myPKEModule.Encrypt(inputData);
```

**myPKEModule**
The name of the Public Key Encryption module structure

**inputData**
The data to encrypt in DATA format

Return:
Encrypted contents in DATA format

The Encrypt function encrypts the given `inputData`, using the options specified in the Public Key Encryption From Buffer module definition.

Example:

```plaintext
//PKE Encryption module definition
MyPKEModule := STD.Crypto.PublicKeyEncryptionFromBuffer('RSA', PublicKey, PrivateKey, '');

DATA encrypted := MyPKEModule.Encrypt((DATA)"The quick brown fox jumps over the lazy dog");
OUTPUT( (STRING)MyPKEModule.Decrypt(encrypted));
```

© 2024 HPCC Systems®. All rights reserved
Decrypt (PKE From Buffer)

myPKEModule.Decrypt(encryptedData);

<table>
<thead>
<tr>
<th>myPKEModule</th>
<th>The name of the Public Key Encryption module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>encryptedData</td>
<td>The data to decrypt in DATA format</td>
</tr>
<tr>
<td>Return:</td>
<td>Decrypted contents in DATA format</td>
</tr>
</tbody>
</table>

The Decrypt function decrypts the given encryptedData using the options specified in the Public Key Encryption From Buffer module definition. You can only decrypt data that was encrypted using the Standard Library's Encrypt method.

Example:

```plaintext
//PKE Encryption module definition
MyPKEModule := STD.Crypto.PublicKeyEncryptionFromBuffer('RSA', PublicKey, PrivateKey, '');
DATA encrypted := MyPKEModule.Encrypt((DATA)'The quick brown fox jumps over the lazy dog');
OUTPUT( (STRING)MyPKEModule.Decrypt(encrypted));
```

© 2024 HPCC Systems®. All rights reserved
141
Sign (PKE From Buffer)

mySymEncModule.Sign(encryptedData);

<table>
<thead>
<tr>
<th>myPKEModule</th>
<th>The name of the Public Key Encryption module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>inputData</td>
<td>The data to sign in DATA format</td>
</tr>
</tbody>
</table>

Return: Computed Digital signature in DATA format

The Sign function creates a digital signature of the given `inputData`, using the options specified in the Public Key Encryption From Buffer module definition.

Example:

```
IMPORT STD;
STRING publicKey := '-----BEGIN PUBLIC KEY-----' + 
   'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KnMwRAoF' + 
   '+'
   '3AUS1DnqT39j31ovv9a12N8g4W5ipqKhTfRESmqZ61/T1UQcmi42oaUXmCeE' + 
   'V9aHqMldyd9aHQOG17CB30Gysw3lf812o77c0eQ3c0rHz0eBk0ryW6X3efWnMoy' + 
   '+
   'ir9Mxcl1df+3W/X01X0ApSs7kuv2PVGF4vJ022+1FV0/XNWjMukJ0S5AuxpYAqv' + 
   '+
   '/
   '/
   '/
   '/
   '
   '-----END PUBLIC KEY-----';

STRING privateKey := '-----BEGIN RSA PRIVATE KEY-----' + 
   'MIIEowIBAAKCAQExr5ZDQmAuzg5PBVV7V/N0A05Gzk6FC8/lDUCeMm' + 
   '+

string PKE Encrption module definition

myPKEModule := STD.Crypto.PublicKeyEncryptionFromBuffer('RSA', publicKey, privateKey,'');

DATA signature := myPKEModule.Sign((DATA)'The quick brown fox jumps');
OUTPUT(TRUE = myPKEModule.VerifySignature(signature, (DATA)'The quick brown fox jumps'));
VerifySignature (PKE From Buffer)

myPKEModule.VerifySignature(signature, signedData);

myPKEModule The name of the Public Key Encryption module structure
signature The Digital signature to verify
signedData Data used to create the signature in DATA format
Return: A BOOLEAN value to indicate verification

The VerifySignature function verifies the given digital signature using the options specified in the Public Key Encryption From Buffer module definition.

Example:

IMPORT STD;
STRING publicKey := '-----BEGIN PUBLIC KEY-----' + '
' + 'MIIBiJANBgkqhkiG9w0BAQEFAASCAR8AMIIBCgKCAQEA64RncpTspV0KmWaRoof' + '
' + '4dAS4U1/IDntG39j10v9a12N98g4W5ipqkftfREsmqZ6l/T1Uqcm42oaUXmCeE' + '
' + '9BhQ1MDwy9aHOQG17CB30Gysw3f1817O7RC7cOQE30CrHn08KOryW6X3fWnmOy' + '
' + 'hir9MxclDf+3W/IX0IaPa97kuVPGV4Yj2021FVO/XNWjMukJG5AsuxpYajQvV' + '
' + 'okJ6q7k1nEtlhLfg3b3mPzQ66zV2z/hublE6Exeyj0j2fQcaC5e35mdp/CW' + '
' + 'J6Tk4NY3MySZe/E/2/ZOWxzdR79XC+5oGNL6/5gP8B/a32B0eM2PsfZwPmVvU0' + '
' + 'bwIDQAFB' + '
' + '-----END PUBLIC KEY-----';
STRING privateKey := '-----BEGIN RSA PRIVATE KEY-----' + '
' + 'MIIEowIBAAKCAQEAr64RncpTspV0KmWaRoof+3ASU/IDntG39j10v9a12N98g' + '
' + '4W5ipqkftfREsmqZ6l/T1Uqcm42oaUXmCeE9BhQ1MDwy9aHOQG17CB30Gysw3f' + '
' + '8107C70C7qE30C7r06eBoryW6X3fEm7nMoyhIR9MmExCLf+3W/X0I10AsPa79' + '
' + 'VPGV4Yj2021FVO/XNWjMukJG5AsuxpYajQvVokJ6q7k1nEtlhLfg3b3mPzQ66z' + '
' + '6zV3z/hublE6Exeyj0j2fQcaC5e35mdp/CW3J6Tk4NY3MySZe/E/2/ZOWxzdR79' + '
' + '+5oGNL6/5gP8B/a3ZB0eM2PsfZwPmVvU0bwIDQAABQ +n +
' + '-----END RSA PRIVATE KEY-----';

//PKE Encryption module definition
myPKEModule := STD.Crypto.PublicKeyEncryptionFromBuffer('RSA', publicKey, privateKey,'');
DATA signature := myPKEModule.Sign((DATA)'The quick brown fox jumps');
OUTPUT(TRUE = myPKEModule.VerifySignature(signature, (DATA)'The quick brown fox jumps'));
Public Key Encryption From LFN Module

myPKEModule := STD.Crypto.PublicKeyEncryptionFromLFN(pkAlgorithm, publicKeyFile, privateKeyFile, passphrase);

### myPKEModule
- **The name of the Public Key Encryption From LFN (Logical FileName) module structure**

### pkAlgorithm
- The algorithm to use, as returned by SupportedPublicKeyAlgorithms()

### publicKeyLFN
- PEM formatted Public Key logical file

### privateKeyLFN
- PEM formatted Private Key logical file

### passphrase
- The passphrase to use for encryption, decryption, signing, verifying

A Public Key Encryption From LFN module is defined in ECL. Subsequent function definitions use the options defined in the Public Key Encryption From LFN module to perform asymmetric encryption/decryption/digital signing/signature verification.

#### Example:

```ecl
IMPORT Std;

PublicKeyFile := '~Examples::certificates::public::pubkey.pem';
PrivateKeyFile:= '~Examples::certificates::private::privkey.pem';

// You can restrict access using file scope security
// on the ~Examples::certificates::private scope
pubKey := RECORD
  STRING Key;
END;

dPubKey := DATASET([
  '-----BEGIN PUBLIC KEY-----' + newline +
  'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KmnWRaofof+3AUS/1DngT39j3Iovv9aI2N8g4W5ipqghKftRESmzq6I7I1Uqcmi42soUXmCE' + newline +
  'BHq1MDydw9aHQOGQ17CB30GSw3l8f8i2o7RC7oQ83cOZ0h0eBkoryW6X36FwNnMoy' + newline +
  'hi99Me91c1df+SWM/X0I0QpAs57kuVPGV4yj202+1F0V/XN0jMUKjG5SAuxYQQv' + newline +
  '/O/KJ6Q7kInEvIhLIGfc3mpB2WQd6Z2v3z/7hLbMEXy5o7j2fQaeC5E5a5mdpk/CW3' + newline +
  '/J6TkYN3YnSw2/2ZOWxZDr79XC+goN61v/5gP18B/a3Z80eM2PSzWvMvVvU0' + newline +
  '/w1DAQAqA' + newline +
  '-----END PUBLIC KEY------' + newline], pubKey);

OUTPUT(dPubKey,,PublicKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);

PrivKey := RECORD
  STRING Key;
END;

dPrivKey := DATASET([
  '-----BEGIN RSA PRIVATE KEY------' + newline +
  'MIIEowIBAAKCAQEAr64RncTp5pV0KmnWRaofof+3AUS/1DngT39j3Iovv9aI2N8g4W5ipqghKftRESmzq6I7I1Uqcmi42soUXmCE' + newline +
  'BHq1MDydw9aHQOGQ17CB30GSw3l8f8i2o7RC7oQ83cOZ0h0eBkoryW6X36FwNnMoy' + newline +
  'hi99Me91c1df+SWM/X0I0QpAs57kuVPGV4yj202+1F0V/XN0jMUKjG5SAuxYQQv' + newline +
  '/O/KJ6Q7kInEvIhLIGfc3mpB2WQd6Z2v3z/7hLbMEXy5o7j2fQaeC5E5a5mdpk/CW3' + newline +
  '/J6TkYN3YnSw2/2ZOWxZDr79XC+goN61v/5gP18B/a3Z80eM2PSzWvMvVvU0' + newline +
  '/w1DAQAqA' + newline +
  '-----END RSA PRIVATE KEY------' + newline], privKey);

OUTPUT(dPrivKey,,PublicKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);
```

© 2024 HPCC Systems®. All rights reserved
Standard Library Reference
Cryptography Support

'Z9HJt5YW47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AANAUzc/G' + '\n' +
'sBo0cNeUeAOGBAXtavF96z2F4Q+k0MBXGt4m7dCJaNC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jvhuvO00Cn8lYbr/Z1YRJDUs6mHh/2BYSkdeaLkojXTxKr2bA4xQk5dtJCodeP' + '\n' +
'0c1t5A7gQK2oNXp/azIcJY/t/cdV1dUL9P41oZthu1Fjwg2666GacEnNAoGBAMzn' + '\n' +
'lwRlX5ldbqemoc+g48w5r35/qs1G8PsZzY8W+cYW7d1NA506acc8YPEWE2RbJdBX' + '\n' +
'Y0AD8nR3zzod00JE34VnKZEx6nQhBOOrtyK9nqHVI/Oxz3Vv6kedUKB6R87K' + '\n' +
'ecZ0jVceEeZsThUlo4t7NmdHqGnxTV++jLvzBoQsAoGAC+fOD+nz6znirkYPr5Ed' + '\n' +
'tw67KtXylr28+CcQoUaq/Au5kjjzE9/4jXtrT09QmVAMcINEnc/sZBjiNzf52wv' + '\n' +
'wZP/bPZMVYKtbasvKd1cNJranHGurkzswbzxSRzmBQ5/YmCrDAuYcnhEqmMWcuU9' + '\n' +
'8ji3j9P9hOX1HdyIBYdV0CqyBV67znuQngz9P9qP/HH1jxllitz31HTu4mMLq' + '\n' +
'9mdAjMszdLTg5uuE+z7N8rp17VuseoRjb3LwLG4+MXIyDbH/0sDdPm1I9qvCNDI' + '\n' +
'spmh9Mgb0hjBSbWaZ0s9/qr/FcSLZ04jLsfRmTPU/Y5y8/dhjY6o0fqhp44RZ' + '\n' +
'ICqNcKQBhF7KZIOQKv4YNYphkIUYW1Nz8Y5o7WtcaQ5lq+kiuB89Prd9rqJLZy2' + '\n' +
'tKf8e6w+wtKjxj18QGK3E/IdkQqlFmBlyEjjRHQ81wS+K5NnjN1c0IEscJQ0Avw9s' + '\n' +
'1lhG5ueb6xoj/N0LuXs8lU5aChKxwRHYdeq948f+qXucJj9R' + '\n' +
'-----END RSA PRIVATE KEY-----' + '\n' +

OUTPUT(dPrivKey,,PrivateKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);

//PKE Encryption module definition
MyPKEModule := STD.Crypto.PublicKeyEncryptionFromLFN('RSA', PublicKeyFile, PrivateKeyFile, '');

DATA encrypted := MyPKEModule.Encrypt((DATA)'The quick brown fox jumps over the lazy dog');
OUTPUT( (STRING)MyPKEModule.Decrypt(encrypted));
Encrypt (PKE From LFN)

**myPKEModule.Encrypt(inputData);**

- **inputData** - The data to encrypt in DATA format

The Encrypt function encrypts the given `inputData`, using the options specified in the Public Key Encryption From LFN module definition.

**Example:**

```plaintext
IMPORT Std;

PublicKeyFile := '-Examples::certificates::public::pubkey.pem';
PrivateKeyFile := '-Examples::certificates::private::privkey.pem';

// You can restrict access using file scope security
// on the 'Examples::certificates::private' scope

dPubKey := RECORD
  STRING  Key;
END;

dPrivKey := RECORD
  STRING  Key;
END;

pubKey := DATASET(
  "BEGIN RSA PRIVATE KEY-----
  MIIEowIBAAKCAQEAr64RncTp5pV0kmWRAofd+3AUS/IDngT39j31ovv9AI28g4WSpiqhKtRESmwQG1/TLQcmi42soU5kmCEf
  od+3AUS/IDngT39j31ovv9AI28g4WSpiqhKtRESmwQG1/TLQcmi42soU5kmCEf
  '81zo7rc7oQ3e0Crdh0eBkoryW6X3efWnMoYIR9MexCldf+3WM/X01X0APs7u
  'PVPG4Yj202+i5F0/X0w1MukjG5ASuxpYAQQv/kOj6q7KnINeVhLiGcm3bPnT9Q6
  '62zv3/z/hUz/ExGY90j2FqAc5E35mdpk/CW3J6tk4NY35wzE/2/Z0WxzdR79XC
  'goNL6/5gPiB8/a3Zo0E2Pf5zPmVnVvU0owbIDQAAb0IABAOQcnATyKOoBu8wW
  'F501d3AKWnPy7m211Wqh3v2AcU17q6+SMR1+yKB6+u55U5QH+HSc9Jm3A7wJ
  'V9PvDrVZvINFYJlCP2orCXYS5Z0mM5z5G6f6B8aC81LSKU5C0e
  'GCK6U18Sr9XEBLPQfKeeTo2yDu9Q4OEKRPBmM61uNHChx2tDb2yv18Zz+EL1ADG
  '79pzRWN0OOCrzc+1LrmmDfYU05VF7l5153j96061LPlZhm3/+QFRGyWsfQB63J
  'Z9HtJW5B4T7RsBRLHxKMc61Jw+dV63rZgTw79P7lA9Z8C6JHy2aArunzG/c
  'sB0ChNeUROAIxG0cTsvWA62F4Q+0kMBXct4mdCJnacv+y71949E0IEF3s9FQ
  'l0jhuwV00CN8bYbr/2Y1YRDaVsm+hZBYSkideaKJXKX2b4AkS50dJ3COPF
  'a1517G0YK20XPF/a2/07CJY7/cdWdU19P41o20hI1Fw66G6AoNaGBAMZ
  'lw1RUKS1dBqgcm+c48wB5j3/q1G8P8ZZW0+0Y7DWolNso6accY8PENW2BdbX
  'Y08A8nRBszzD0oOJEj4VnNZeX6nQbH0oRytKnnqHVI/Ox93VvU6kedUkDRB87KC
  ',

  OUTPUT(dPubKey,,PublicKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);

PrivKey := RECORD
  STRING  Key;
END;

dPrivKey := DATASET(
  "BEGIN RSA PRIVATE KEY-----
  MIIEowIBAAKCAQEAr64RncTp5pV0kmWRAofd+3AUS/IDngT39j31ovv9AI28g4WSpiqhKtRESmwQG1/TLQcmi42soU5kmCEf
  od+3AUS/IDngT39j31ovv9AI28g4WSpiqhKtRESmwQG1/TLQcmi42soU5kmCEf
  '81zo7rc7oQ3e0Crdh0eBkoryW6X3efWnMoYIR9MexCldf+3WM/X01X0APs7u
  'PVPG4Yj202+i5F0/X0w1MukjG5ASuxpYAQQv/kOj6q7KnINeVhLiGcm3bPnT9Q6
  '62zv3/z/hUz/ExGY90j2FqAc5E35mdpk/CW3J6tk4NY35wzE/2/Z0WxzdR79XC
  'goNL6/5gPiB8/a3Zo0E2Pf5zPmVnVvU0owbIDQAAb0IABAOQcnATyKOoBu8wW
  'F501d3AKWnPy7m211Wqh3v2AcU17q6+SMR1+yKB6+u55U5QH+HSc9Jm3A7wJ
  'V9PvDrVZvINFYJlCP2orCXYS5Z0mM5z5G6f6B8aC81LSKU5C0e
  'GCK6U18Sr9XEBLPQfKeeTo2yDu9Q4OEKRPBmM61uNHChx2tDb2yv18Zz+EL1ADG
  '79pzRWN0OOCrzc+1LrmmDfYU05VF7l5153j96061LPlZhm3/+QFRGyWsfQB63J
  'Z9HtJW5B4T7RsBRLHxKMc61Jw+dV63rZgTw79P7lA9Z8C6JHy2aArunzG/c
  'sB0ChNeUROAIxG0cTsvWA62F4Q+0kMBXct4mdCJnacv+y71949E0IEF3s9FQ
  'l0jhuwV00CN8bYbr/2Y1YRDaVsm+hZBYSkideaKJXKX2b4AkS50dJ3COPF
  'a1517G0YK20XPF/a2/07CJY7/cdWdU19P41o20hI1Fw66G6AoNaGBAMZ
  'lw1RUKS1dBqgcm+c48wB5j3/q1G8P8ZZW0+0Y7DWolNso6accY8PENW2BdbX
  'Y08A8nRBszzD0oOJEj4VnNZeX6nQbH0oRytKnnqHVI/Ox93VvU6kedUkDRB87KC
  ',

  OUTPUT(dPrivKey,,PrivateKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);

```

© 2024 HPCC Systems®. All rights reserved

146
'ecO1VeEztsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAIfO0+nz6znirYSpRe5D' + '"' +
'tW67KtxlIr28+cCqOuAp/Au5kjz9E9/4DjXrT09QmAVMcINc/sZBjiNzF525wv' + '"' +
'wP/bFMVKtbsaVkd1cN3ranHGUrzswbxSRzmBq5/YmCNwDAnYcnhEgqMWcuU9' + '"' +
'8jSiJ3P9hXLHDyIBYdhVuQOgYV6TzuQgnzp9Wpq/H8ijl1lz31HTw4mLMLR' + '"' +
'0mdA)MkszdLTg5uuE+z+N8rp17VuseoRj3h3LvLG4+MKIy DbH/0sDDpm+IjqvCNDR' + '"' +
'spnn9MgbOJbsbNawZK0s9/qri/FcSLZ04JLsfMTPU/Y5y8/DHjYOfdfDqhp44RZf' + '"' +
'iCqxKQBgH7KZIOkqV4YNyphkIUYWHNz8Y5o7WtaQ51q+ktbU8PRd9rqJLZyk2' + '"' +
'tkFse6z+wtKjxi8GKQzz/IdkQqFmBlyEjjRHZQ81WS+K5UnjN1t0IEscJoQAOwv9s' + '"' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWXrRHEYdeqQ48f+qXucJj9R' + '"' +
'-----END RSA PRIVATE KEY-----' + '"' +
]);PrivKey);

OUTPUT(dPrivKey,,PrivateKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);

//PKE Encryption module definition
MyPKEmodule := STD.Crypto.PublicKeyEncryptionFromLFN('RSA', PublicKeyFile, PrivateKeyFile, ');

DATA encrypted := MyPKEmodule.Encrypt((DATA)'The quick brown fox jumps over the lazy dog');
OUTPUT( (STRING)MyPKEmodule.Decrypt(encrypted));
Decrypt (PKE From LFN)

myPKEModule_decrypt(encryptedData);

myPKEModule
The name of the Public Key Encryption From LFN (Logical File Name) module structure

encryptedData
The data to decrypt in DATA format

Return:
Decrypted contents in DATA format

The Decrypt function decrypts the given encryptedData, using the options specified in the Public Key Encryption From LFN module definition. You can only decrypt data that was encrypted by the Standard Library's Encrypt method.

Example:

IMPORT Std;

PublicKeyFile := '~Examples::certificates::public::pubkey.pem';
PrivateKeyFile:= '~Examples::certificates::private::privkey.pem';

//You can restrict access using file scope security
//on the ~Examples::certificates::private scope

pubKey := RECORD
  STRING  Key;
END;

dPubKey := DATASET([{"-----BEGIN PUBLIC KEY-----" + '
' + 'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA6rzT93Izov9aI2N8g4W9pghKftfRESmzQ61/T1UQcmi42soUXmCeE' + '
' + '81zo7C0Qe30Ch0eByW63efWnMoYhIR9MxelCldF+3WM/x0XOpaSs7ku' + '
' + 'VPVg4y12220+1FVO/XNwMukjG5ASupsYPQv/\oK6q7kInEIvLIGfcm3bZuMQ66zVz23z/hulb3EY5ojoWfaQ5E3s5mdpk/CW3' + '
' + 'J6Tk4NY3lySWez/2/7GWxZd79XCXgONL6v/5/PI8B/a3Z03eM2PfszWmVuVU0' + '
' + 'bw1DAQB+" + '-----END PUBLIC KEY-----' + '
' + '}),pubKey]);

PrivKey := RECORD
  STRING  Key;
END;

dPrivKey := DATASET([{"-----BEGIN RSA PRIVATE KEY-----" + '
' + 'MIIEowIBAAKCAQEAr64RncTp5v90WnWRAof+3AUS/IDngT39j31ov9aI2N8g' + '
' + '81zo7C0Qe30Ch0eByW63efWnMoYhIR9MxelCldF+3WM/x0XOpaSs7ku' + '
' + 'VPVg4y12220+1FVO/XNwMukjG5ASupsYPQv/\oK6q7kInEIvLIGfcm3bZuMQ66zVz23z/hulb3EY5ojoWfaQ5E3s5mdpk/CW3' + '
' + 'J6Tk4NY3lySWez/2/7GWxZd79XCXgONL6v/5/PI8B/a3Z03eM2PfszWmVuVU0' + '
' + 'bw1DAQB+" + '-----END RSA PRIVATE KEY-----' + '
' + '}),PrivKey]);

OUTPUT(dPubKey,,PublicKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);

INPUT(dPrivKey,,PrivKeyFile, CSV(SEPARATOR(','), TERMINATOR(',')), REPLACE);
Cryptography Support

//PKE Encryption module definition
MyPKEModule := STD.Crypto.PublicKeyEncryptionFromLFN('RSA', PublicKeyFile, PrivateKeyFile, '');

DATA encrypted := MyPKEModule.Encrypt((DATA)'The quick brown fox jumps over the lazy dog');
OUTPUT( (STRING)MyPKEModule.Decrypt(encrypted));
Sign (PKE From LFN)

`mySymEncModule.Sign(encryptedData);`

<table>
<thead>
<tr>
<th><strong>myPKEModule</strong></th>
<th>The name of the Public Key Encryption From LFN (Logical FileName) module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inputData</strong></td>
<td>The data to sign in DATA format</td>
</tr>
<tr>
<td><strong>Return</strong></td>
<td>Computed Digital signature in DATA format</td>
</tr>
</tbody>
</table>

The Sign function creates a digital signature of the given `inputData`, using the options specified in the Public Key Encryption From LFN module definition.

Example:

```
IMPORT Std;

PublicKeyFile := '~Examples::certificates::public::pubkey.pem';
PrivateKeyFile:= '~Examples::certificates::private::privkey.pem';
//You can restrict access using file scope security
//on the ~Examples::certificates scope

pubKey := RECORD
  STRING Key;
END;

dPubKey := DATASET({
  '-----BEGIN PUBLIC KEY-----' + '
' +
  'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KmnWRAof' + '
' +
  'od+3AUS/IDngT39j31ovv9a12N8g4W5ipqgKftRESmzq6I/T119cm142soUXmCeE' + '
' +
  'c9Hg1Mdyd9aHQQG17CB30Gysw3f18i3o7RC7ocQR30cRzhObkoryW6X3efWnMoy' + '
' +
  'hI9RMxKclDF+3WM/X0I0ApSa7kuVPVG4Yj202+1FVO/XNWjMukJG5ASuxyAQVv' + '
' +
  '/0kJ6q71nEihLhGcm3bpTzWQ66zVz3z/huLhEKEYy5oj2fQac5E3s5mdpk/CW3' + '
' +
  ',/W67kYQzN3y5Swze/2/Z0XZdR79XcQo3mNLv/5gP18B/a3200mrEPfSzFmPuVuvU0' + '
' +
  '+bwIDAQAB' + '
' +
  '-----END PUBLIC KEY-----'
},dPubKey);

OUTPUT(dPubKey,,PublicKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);

PrivKey := RECORD
  STRING Key;
END;

dPrivKey := DATASET({
  '-----BEGIN RSA PRIVATE KEY-----' + '
' +
  'MIIEowIBAAKCAQEAr64RncTp5pV0KmnWRAofd+3AUS/IDngT39j31ovv9a12N8g4W5ipqgKftRESmzq6I/T119cm142soUXmCeE' + '
' +
  'c9Hg1Mdyd9aHQQG17CB30Gysw3f18i3o7RC7ocQR30cRzhObkoryW6X3efWnMoy' + '
' +
  'hI9RMxKclDF+3WM/X0I0ApSa7kuVPVG4Yj202+1FVO/XNWjMukJG5ASuxyAQVv' + '
' +
  '/0kJ6q71nEihLhGcm3bpTzWQ66zVz3z/huLhEKEYy5oj2fQac5E3s5mdpk/CW3' + '
' +
  ',/W67kYQzN3y5Swze/2/Z0XZdR79XcQo3mNLv/5gP18B/a3200mrEPfSzFmPuVuvU0' + '
' +
  '+bwIDAQAB' + '
' +
  '-----END PUBLIC KEY-----'
},dPrivKey);
```

© 2024 HPCC Systems®. All rights reserved
'eCzOlVcEeZtsTHuLO4t7NmdHGqNxTV+jlVzBoQsrAoGAIf0D+nz6znirYSpRe5D' + '
' +
tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DJxrT9qmVMciNEnc/szBjiNzF525wv' + '
' +
'wZP/bP2MvyKtbsaVkd1cN3ranHGuKzswebxSRzmBQ5/YmCNwDAuYcnhEeqmMWWcuU9' + '
' +
'8js13jP9hOXLHdyIBDhV0CGYBV6TznUQgnzp9WpQ/H8ijxl1Iz31HTh4mLMJR' + '
' +
'9mdaJMKs2dLtg5uue+e+N8rp17VUseoRj3b3LmLG4+MXIyDbH/0sDdp+IjqvCNDR' + '
' +
'spmh9MgBh0JbbaWzK0s9/qRI/FcSLX04JlsfTMu/Y5y8/DhjYO6fdQhp44RZf' + '
' +
'iCqNQKbgHF7KZlOKqV4YNyphk1UYWHNz9yY5o7WytsaQ51q+k1bU8PRd9rqJLZyk2' + '
' +
'tKf8e6z+wTJkx18GKQzE/IdkQqiFmBlAeEyjRHQ8lWS+K5NnjN1t0IEscJqoaWv9s' + '
' +
'iIHg5ue6xoj/NOLuxa8loUT5aChKWXrRHEYdegQU48f+qxUcJj9R' + '
' +
'--------END RSA PRIVATE KEY--------' + '
' +
],PrivKey);

OUTPUT(dPrivKey,,PrivateKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);

//PKE Encryption module definition
MyPKEModule := STD.Crypto.PublicKeyEncryptionFromLFN('RSA', PublicKeyFile, PrivateKeyFile, '');

DATA signature := myPKEModule.Sign((DATA)'The quick brown fox jumps');
OUTPUT(TRUE = myPKEModule.VerifySignature(signature, (DATA)'The quick brown fox jumps'));

© 2024 HPCC Systems®. All rights reserved
151
VerifySignature (PKE From LFN)

**myPKEModule**.VerifySignature(\texttt{signature, signedData});

<table>
<thead>
<tr>
<th><strong>myPKEModule</strong></th>
<th>The name of the Public Key Encryption From LFN (Logical FileName) module structure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>signature</strong></td>
<td>The Digital signature to verify</td>
</tr>
<tr>
<td><strong>signedData</strong></td>
<td>Data used to create the signature in DATA format</td>
</tr>
</tbody>
</table>

Return: A BOOLEAN value to indicate verification

The VerifySignature function verifies the given digital \texttt{signature} using the options specified in the Public Key Encryption From LFN module definition.

Example:

```plaintext
IMPORT Std;

PublicKeyFile := '~Examples::certificates::public::pubkey.pem';
PrivateKeyFile:= '~Examples::certificates::private::privkey.pem';
//You can restrict access using file scope security
//on the ~Examples::certificates::private scope
pubKey := RECORD
  STRING Key;
END;
dPubKey := DATASET([{"-----BEGIN PUBLIC KEY-----" + '\n' +
    "MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCQEAr46RncT5pV0KMNwRAof' + '\n' +
    "0d+i3/AUS/1Dnq393Iov9aI2N8g4W5ipqKhKftRESmzQ61I1UQcmi420sUXMcEe' + '\n' +
    "BHq1MDYdw9aHOOG17CB3OGYsw3lfsb1zo7RC7cQ0X3cRz0H0eBBoryW63FefNnMoy' + '\n' +
    "hIR9Mx1CldF+3W3/1X9X0ApSa7kuUPVPGyYj202+1FV0/XNWjMk5G5ASuxpYAAv' + '\n' +
    "/oKj6q7jIeVhLgFcm3bTzW6Q66ZvZ3h/\u81bhbEKExy5oj2faC53s5mdpk/CW3' + '\n' +
    "J6Tk4N3ySwZe/2/ZGwzDr79XC/goNLL6v/5gPI8B/a3Z8eM2FzSwPMnVvU0' + '\n' +
    "\'bwIDAQAB' + '\n' +
    "-----END PUBLIC KEY------' + '\n' +
}],pubKey);

OUTPUT(dPubKey,,PublicKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);

PrivKey := RECORD
  STRING Key;
END;
dPrivKey := DATASET([{"-----BEGIN RSA PRIVATE KEY-----" + '\n' +
    "MIIEowIBAAKCAQEAr46RncT5pV0KMNwRAof+3AU/iDnq393Iov9aI2N8g' + '\n' +
    "4W5ipqKhKftRESmzQ61I1UQcmi420sUXMcEe' + '\n' +
    "81zOE0c7cOcRh0EBkoryW63FefNnMoyh199emC1dF+3W3/1X9X0ApSa7ku' + '\n' +
    "VPVPGyYj202+1FV0/XNWjMk5G5ASuxpYAAv' + '\n' +
    "6zV3ZL/huLhEXEySoj2fQac53s5mdpk/CW3sDr79XC' + '\n' +
    "+ogNL6v/5gPI8B/a3Z8eM2FzSwPMnVvU0bwIDAQABAoIBAQCnGAtNYkOuBwWh' + '\n' +
    "F50i5aKwNPyF21lWNhY2AcF1U7qle+zSRLy+yKBL6+u5RQqU+HSc9Jm31AJW' + '\n' +
    "VlYpRvXVIF3YJCPPzorX35Z9oMmAu2g5PBV7V7VNA0A5GzC6FC8AllD30cEom' + '\n' +
    "GCR6u1Sm89XnELBq6KeeTo2yDuG7Q4oBXRnPBN61uHChxht2xBvIy18z+EL1ADG' + '\n' +
    "70pzRWN0ZGczc+ArIImDFw0gyFV7LNsin53jO6Ou1LPzhkm3/OFQRYWwFqQ863' + '\n' +
    "Z9HJKB5YB47TR5rbLHKeMcZI2N+i+dw+ShrzgTdzK797wAKn8JCDYe2aAANACG" + '\n' +
    "bSc8Uk5URoABmQG4TvA62zFQg+komBcxtcm7dC3nac+qthF249uEI1BeF3dasF9e" + '\n' +
    "F0j9nV0UCN81Yl8p/2YSKdeahLKo7XTXKR2hAh4xQk5dJCD0p" + '\n' +
    "0c15AL7gOHr40XxPA/azDICYT/cvdIUDUL940Zthup1jwG266GaoEnNa0GBAMZ" + '\n' +
    "1wUXS1dCqemoc+g48Wj53rqSTGBsPZ2Y8W+oY7dINAN506aacc8YFEWE2RBD0b" + '\n' +
},dPrivKey);
```
'YEADBnRsdzzOdo0JEj4VbNZETx6nQhBO0rtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
'eczo1VcEezZtsThuL0t7NmdHqGqNxTV+jlVzBoQasAoGAI+fGD+nv6znirYSpRe5D' + '\n' +
'tw67KtXyrl28+CCqoUaq/Au5kjsE9/4DjXrT09QeVAMcINEnc/sZBjjiNZf525wv' + '\n' +
'wEP/bP2MYKtb5aVkdLoNJrvnG4UYkzswbxSRzmBo5/YmGNnDAuYchnEgmNWcuU9' + '\n' +
'8jiS13JP9bX1HDyIRYDhV0CgYBV6TznupQqzn9NgQ/H8iJxlItz3lHTu4mLM1R' + '\n' +
'9mdA JMkszdLTg5uuE+z+N8rp17VUs0e0Rj3LvLG4+MXIyDbh/0sDdPm+IjqvCNDR' + '\n' +
'spnh9MqBh0jbNwa2K0s9/qrl/FcSLZ04JLsfRMTPU/Y5y8/dHjY0f7Qhp44RZF' + '\n' +
'icgNzQBhgHf7KZIOkqV4YNyphk1UYWHNh87T5Q51Q+k1bU8Pd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjx18GKQzE/IdkQqiFmBlyEjjHRQ81WS+K5NnJ1t0IEscJqOAwv9s' + '\n' +
'iHhsUeb6xoj/N0LunA8loUT5acHkwRxREydegqU48f+qxAjyJ9R' + '\n' +
'-----END RSA PRIVATE KEY-----' + '\n'
}),PrivKey);

OUTPUT(dPrivKey,,PrivateKeyFile, CSV(SEPARATOR(''), TERMINATOR('')), OVERWRITE);

//PKE Encryption module definition
MyPKEmodule := STD.Crypto.PublicKeyEncryptionFromLFN('RSA', PublicKeyFile, PrivateKeyFile, '');

DATA signature := myPKEmodule.Sign((DATA)'The quick brown fox jumps');
OUTPUT(TRUE = myPKEmodule.VerifySignature(signature, (DATA)'The quick brown fox jumps'));
Date and Time Handling
Date Data Types

STD.Date.Date_rec

STD.Date.Date_t

STD.Date.Days_t

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date_rec</td>
<td>A RECORD structure containing three fields, and INTEGER2 year, an UNSIGNED1 month, and an UNSIGNED1 day.</td>
</tr>
<tr>
<td>Date_t</td>
<td>An UNSIGNED4 containing a date value in YYYYMMDD format.</td>
</tr>
<tr>
<td>Days_t</td>
<td>An UNSIGNED4 containing a date value representing the number of elapsed days since a particular base date. This number can be the number of days in the common era (January 1, 1AD = 1) based on either the Julian or Gregorian calendars, or the number of elapsed days since the Gregorian calendar's January 1, 1900 (January 1, 1900 = 1).</td>
</tr>
</tbody>
</table>

The three Date data types defined in the Date Standard Library are:

```plaintext
// A record structure with the different elements separated out.
EXPORT Date_rec := RECORD
    INTEGER2   year;
    UNSIGNED1  month;
    UNSIGNED1  day;
END;

// An unsigned number holding a date in the decimal form YYYYMMDD.
// This type does not support dates prior to 1AD
EXPORT Date_t := UNSIGNED4;

// A number of elapsed days. Value depends on the function called.
EXPORT Days_t := UNSIGNED4;
```

See Also: Time Data Types
**Time Data Types**

**STD.Date.Time_rec**

**STD.Date.Time_t**

**STD.DateTime_rec**

**STD.Timestamp_t**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time_rec</td>
<td>A RECORD structure containing three fields, and INTEGER1 hour, an UNSIGNED1 minute, and an UNSIGNED1 second.</td>
</tr>
<tr>
<td>Time_t</td>
<td>An UNSIGNED3 holding a time of day in the decimal form HHMMDD.</td>
</tr>
<tr>
<td>Seconds_t</td>
<td>An INTEGER8 holding holding a number of seconds. Can be used to represent either a duration or the number of seconds since epoch (Jan 1, 1970).</td>
</tr>
<tr>
<td>DateTime_rec</td>
<td>A RECORD structure containing both a Date_rec and a Time_rec</td>
</tr>
<tr>
<td>Timestamp_t</td>
<td>An INTEGER8 holding a number of microseconds. Can be used to represent either a duration or the number of microseconds since epoch (Jan 1, 1970).</td>
</tr>
</tbody>
</table>

The **Time data types** defined in the Date Standard Library are:

```plaintext
// A record structure with the different time elements separated out.
EXPORT Time_rec := RECORD
  UNSIGNED1    hour;
  UNSIGNED1    minute;
  UNSIGNED1    second;
END;

// An unsigned number holding a time of day in the decimal form HHMMDD.
EXPORT Time_t := UNSIGNED3;

// A signed number holding a number of seconds. Can be used to represent either a duration or the number of seconds since epoch (Jan 1, 1970).
EXPORT Seconds_t := INTEGER8;

// A record structure with the different date and time elements separated out.
EXPORT DateTime_rec := RECORD
  Date_rec;
  Time_Rec;
END;

// A signed number holding a number of microseconds. Can be used to represent either a duration or the number of microseconds since epoch (Jan 1, 1970).
EXPORT Timestamp_t := INTEGER8;
```

See Also: Date Data Types
Year

STD.Date.Year( date )

date | A date value in the Date_t format.

Return: Year returns an INTEGER value.

The Year function returns the Year number from the date value.

Example:

```
IMPORT STD;
UNSIGNED4 MyDate := 20120101;  //January 1, 2012

Y := STD.Date.Year(MyDate);
   //Y contains 2012
```
## Month

**STD.Date.Month( date )**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>A date value in the Date_t format.</td>
</tr>
</tbody>
</table>

Return: Month returns an INTEGER value in the range of 1 through 12.

The `Month` function returns the month number from the `date` value.

**Example:**

```plaintext
IMPORT STD;
UNSIGNED4 MyDate := 20120101;  //January 1, 2012
M := STD.Date.Month(MyDate);
  //M contains 1, representing January
```
Day

STD.Date.Day( date )

date A date value in the Date_t format.

Return: Day returns an INTEGER value in the range of 1 through 31.

The Day function returns the Day number from the date value.

Example:

IMPORT STD;
UNSIGNED4 MyDate := 20120101; //January 1, 2012

D := STD.Date.Day(MyDate);
//D contains 1, representing the first of the month
### Hour

STD.Date.Hour( time )

<table>
<thead>
<tr>
<th>time</th>
<th>A time value in the Time_ format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Hour returns an INTEGER value representing the hour in the range of 0-23.</td>
</tr>
</tbody>
</table>

The **Hour** function returns the hour from the *time* value.

**Example:**

```plaintext
IMPORT STD;
MyTime:= STD.Date.CurrentTime(TRUE); //Local Time

t1 := STD.Date.Hour(MyTime);
//t1 contains the hour of the current local time
```
Minute

STD.Date.Minute( time )

<table>
<thead>
<tr>
<th>time</th>
<th>A time value in the Time_ format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Minute returns an INTEGER value representing the minute in the range of 0-59.</td>
</tr>
</tbody>
</table>

The **Minute** function returns the minute from the *time* value.

Example:

```pascal
IMPORT STD;
MyTime:= STD.Date.CurrentTime(TRUE);  //Local Time

t1 := STD.Date.Minute(MyTime);
    //t1 contains the minute of the current local time
```
Second

STD.Date.Second( time )

<table>
<thead>
<tr>
<th>time</th>
<th>A time value in the Time_ format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Second returns an INTEGER value representing the second in the range of 0-59.</td>
</tr>
</tbody>
</table>

The Second function returns the second from the time value.

Example:

```hpcc
IMPORT STD;
MyTime:= STD.Date.CurrentTime(TRUE);  //Local Time

t1 := STD.Date.Second(MyTime);
    //t1 contains the second of the current local time
```
DateFromParts

STD.Date.DateFromParts( year, month, day )

- **year**: An INTEGER2 year value in the range 0 to 9999.
- **month**: An UNSIGNED1 month value in the range 1 to 12.
- **day**: An UNSIGNED1 day value in the range 1 to 31.

Return: DateFromParts returns an UNSIGNED4 value.

The *DateFromParts* function returns a Date_t value from the *year*, *month*, and *day* parameters.

Example:

```plaintext
IMPORT STD;
INTEGER2 MyYear := 2012;
UNSIGNED1 MyMonth := 1;
UNSIGNED1 MyDay := 1;

D := STD.Date.DateFromParts(MyYear,MyMonth,MyDay);
  //D contains 20120101, representing January 1, 2012
```
### TimeFromParts

**STD.Date.TimeFromParts** (hour, minute, second)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hour</td>
<td>An INTEGER1 hour value in the range 0 to 23.</td>
</tr>
<tr>
<td>minute</td>
<td>An UNSIGNED1 minute value in the range 0 to 59.</td>
</tr>
<tr>
<td>second</td>
<td>An UNSIGNED1 second value in the range 0 to 59.</td>
</tr>
</tbody>
</table>

**Return:** TimeFromParts returns a Time_t (An UNSIGNED3 holding a time of day in the decimal form HHMMDD.)

The **TimeFromParts** function returns a Time_t value from the hour, minute, and second parameters.

**Example:**

```plaintext
IMPORT STD;
UNSIGNED1   MyHour   := 23;
UNSIGNED1   MyMinute := 59;
UNSIGNED1   MySecond := 50;
T := STD.Date.TimeFromParts(MyHour,MyMinute,MySecond);
//T contains 235950
```
IsLeapYear

STD.Date.IsLeapYear( year )

<table>
<thead>
<tr>
<th>year</th>
<th>A year value in the INTEGER2 format.</th>
</tr>
</thead>
</table>

Return: IsLeapYear returns a BOOLEAN value.

The IsLeapYear function returns TRUE if the year is a leap year in the Gregorian (or proleptic Gregorian) calendar.

Example:

```plaintext
IMPORT STD;
INTEGER2 MyYear := 2012; //2012
D := STD.Date.IsLeapYear(MyYear);
  //D contains TRUE, 2012 is a leap year
```
IsDateLeapYear

STD.Date.IsDateLeapYear( date )

date | A date in Date_t format. (An UNSIGNED4 containing a date value in YYYYMMDD format.)
Return: IsDateLeapYear returns a BOOLEAN value.

The IsDateLeapYear function returns TRUE if the year represented in the date is a leap year in the Gregorian (or proleptic Gregorian) calendar.

Example:

IMPORT STD;
MyDate := 20120112;  //Jan. 12, 2012
D := STD.Date.IsDateLeapYear(MyDate);
   //D contains TRUE, 2012 is a leap year
IsValidDate

STD.Date.IsValidDate( date, [yearLowerBound],[yearUpperBound] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>A date value in the Date_t format.</td>
</tr>
<tr>
<td>yearLowerBound</td>
<td>The minimum acceptable year. Optional; defaults to 1800.</td>
</tr>
<tr>
<td>yearUpperBound</td>
<td>The maximum acceptable year. Optional; defaults to 2100.</td>
</tr>
</tbody>
</table>

Return:
IsValidDateYear returns a BOOLEAN value.

The IsValidDate function returns TRUE if the date is valid, both by range-checking the year and by validating each of the other individual components.

Example:

```cpp
IMPORT STD;
d1 := 19631122;
d2 := 19990230;
firstTest := STD.Date.IsValidDate(d1); //d1 is valid
secondTest := STD.Date.IsValidDate(d2); //d2 is not valid
```
IsValidTime

STD.Date.IsValidTime( time )

<table>
<thead>
<tr>
<th>time</th>
<th>A time value in the Time_t format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return</td>
<td>IsValidTime returns a BOOLEAN value.</td>
</tr>
</tbody>
</table>

The IsValidTime function returns TRUE if the time is valid, by validating each of the individual components (hours, minutes, and seconds).

Example:

```
IMPORT STD;

t1 := 225922;
t2 := 275922;

firstTest := STD.Date.IsValidTime(t1); //true
secondTest := STD.Date.IsValidTime(t2); //false
```
IsValidGregorianDate

STD.Date.IsValidGregorianDate( date )

date A date value in the Date_t format. (An UNSIGNED4 containing a date value in YYYYMMDD format.)

Return: IsValidGregorianDateYear returns a BOOLEAN value.

The IsValidGregorianDate function returns TRUE if the date is valid in the Gregorian calendar. The year must be between 1601 and 30827.

Example:

```hcl
IMPORT STD;
d1 := 19991122;
d2 := 15130230;
firstTest := STD.Date.IsValidGregorianDate(d1); // TRUE
secondTest := STD.Date.IsValidGregorianDate(d2); // FALSE
```
**FromGregorianYMD**

**STD.Date.FromGregorianYMD (year, month, day)**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>An INTEGER2 year value in the range 0 to 9999.</td>
</tr>
<tr>
<td>month</td>
<td>An UNSIGNED1 month value in the range 1 to 12.</td>
</tr>
<tr>
<td>day</td>
<td>An UNSIGNED1 day value in the range 1 to 31.</td>
</tr>
</tbody>
</table>

**Return:**

FromGregorianYMD returns an UNSIGNED4 value.

The `FromGregorianYMD` function returns a Days_t value from the `year`, `month`, and `day` parameters representing the number of days since 31st December 1BC in the Gregorian calendar (see The Calendar FAQ by Claus Tondering at http://www.tondering.dk/claus/calendar.html).

**Example:**

```plaintext
IMPORT STD;
INTEGER2 MyYear := 2012;
UNSIGNED1 MyMonth := 1;
UNSIGNED1 MyDay := 1;

D := STD.Date.FromGregorianYMD(MyYear, MyMonth, MyDay);
//D contains 734503
```
ToGregorianYMD

STD.Date.ToGregorianYMD( days )

<table>
<thead>
<tr>
<th>days</th>
<th>A year value in the Days_t format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>ToGregorianYMD returns separate values for Year, Month, and Day.</td>
</tr>
</tbody>
</table>

The ToGregorianYMD function converts the number days since 31st December 1BC to a date in the Gregorian calendar. It returns a module with three exported values: Year, Month, and Day.

Example:

```plaintext
IMPORT STD;
INTEGER2 MyYear := 2012;
UNSIGNED1 MyMonth := 1;
UNSIGNED1 MyDay := 1;

J := STD.Date.FromGregorianYMD(MyYear,MyMonth,MyDay);
   //J contains 734503

X := STD.Date.ToGregorianYMD(J);
   // X is a module with exported values

Y := X.Year;   //Y contains 2012
M := X.Month;  //M contains 1
D := X.Day;    //D contains 1
```
FromStringToDate

**STD.DateFromStringToDate**( date_text, format )

<table>
<thead>
<tr>
<th>date_text</th>
<th>The string to be converted</th>
</tr>
</thead>
<tbody>
<tr>
<td>format</td>
<td>The format of the input string. See strftime documentation for details (<a href="http://strftime.org/">http://strftime.org/</a>)</td>
</tr>
<tr>
<td>return</td>
<td>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.</td>
</tr>
</tbody>
</table>

The `FromStringToDate` function converts a string to a Date_t using the relevant string format.

If the resulting date must be representable within the Gregorian calendar after the year 1600, you should use the `Std.Date.IsValidGregorianDate()` function to determine its validity.

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'

**Example:**

```plaintext
IMPORT STD;
D1 := STD.DateFromStringToDate('19720607', '%Y%m%d');  //D1 contains 19720607
D2 := STD.DateFromStringToDate('19720007', '%Y%m%d');  //D2 contains 0
D3 := STD.DateFromStringToDate('4/29/1974', '%m/%d/%Y');  //D3 contains 19740429
D4:= STD.DateFromStringToDate('29/4/1974', '%d/%m/%Y');  //D4 contains 19740429
```

See Also: IsValidGregorianDate
Today

STD.Date.Today()

Return: Today returns date_t (an UNSIGNED4 containing a date value in YYYMMDD format) representing the current date.

The Today function returns the current date in the local time zone.

Example:

```
IMPORT STD;

D1 := STD.Date.Today();

// D1 contains today's date
```
CurrentDate

STD.Date.CurrentDate ([in_local_time])

<table>
<thead>
<tr>
<th>in_local_time</th>
<th>TRUE if the returned value should be local to the cluster computing the date, FALSE for UTC. Optional, defaults to FALSE.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Today returns a Date_t representing the current date.</td>
</tr>
</tbody>
</table>

The CurrentDate function returns the current date. If the in_local_time parameter is TRUE the returned value is local to the cluster computing the date, if FALSE then the UTC value is returned.

Example:

```cpp
IMPORT STD;
d1 := STD.Date.CurrentDate(True);
//d1 contains the current local date
```
CurrentTime

STD.Date.CurrentTime ([in_local_time])

<table>
<thead>
<tr>
<th>in_local_time</th>
<th>TRUE if the returned value should be local to the cluster computing the time, FALSE for UTC. Optional, defaults to FALSE.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Today returns a time_t (An UNSIGNED3 holding a time of day in the decimal form HH-MMDD.)</td>
</tr>
</tbody>
</table>

The `CurrentTime` function returns the current time. If the `in_local_time` parameter is TRUE the returned value is local to the cluster computing the time, if FALSE then the UTC is returned.

On containerized systems, servers are usually set to UTC making local time and UTC identical.

Example:

```plaintext
IMPORT STD;
t1 := STD.Date.CurrentTime(True);
//t1 contains the current local time of day
```
DayOfWeek

STD.Date.DayOfWeek( date)

<table>
<thead>
<tr>
<th>date</th>
<th>A date value in the Date_t format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>DayOfWeek returns an INTEGER value representing the day of the week, where 1 = Sunday.</td>
</tr>
</tbody>
</table>

The DayOfWeek function returns a number representing the day of the week for the given date. The date must be in the Gregorian calendar after the year 1600.

Example:

```plaintext
IMPORT STD;
D1 := STD.Date.DayOfWeek(STD.Date.Today());
// D1 contains the day of the week for today's date
```
DayOfYear

STD.Date.DayOfYear( date)

date | A date value in the Date_t format.
Return: | DayOfYear returns an INTEGER value in the range of 1 through 366.

The DayOfYear function returns a number representing the day of the year for the given date. The date must be in the Gregorian calendar after the year 1600.

Example:

```
IMPORT STD;
D1 := STD.Date.DayOfYear(STD.Date.Today());
// D1 contains the day of the year for today's date
```
DaysBetween

STD.Date.DaysBetween( fromDate, toDate)

<table>
<thead>
<tr>
<th>fromDate</th>
<th>The first date value in Date_t format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>toDate</td>
<td>The last date value in Date_t format.</td>
</tr>
</tbody>
</table>

Return: DaysBetween returns an INTEGER value of the number of days between the two dates.

The DaysBetween function calculates the number of whole days between two dates.

Example:

```plaintext
IMPORT STD;
StartDate := 19631122;
numDays := STD.Date.DaysBetween(startDate,STD.Date.Today());
// numDays contains the number of days between the startDate and today's date
```
### MonthsBetween

STD.Date.MonthsBetween( fromDate, toDate)

<table>
<thead>
<tr>
<th><strong>fromDate</strong></th>
<th>The first date value in Date_t format.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>toDate</strong></td>
<td>The last date value in Date_t format.</td>
</tr>
<tr>
<td><strong>month_ends_equal</strong></td>
<td>Optional. If TRUE and both dates fall on the last day of their respective months, the difference between the dates will be treated as whole months regardless of the actual day values. If FALSE then the day value of each date is considered when calculating the difference. The default is FALSE</td>
</tr>
</tbody>
</table>

**Return:** MonthsBetween returns an INTEGER value of the number of whole months between the two dates.

The **MonthsBetween** function calculates the number of whole months between two dates.

**Example:**

```plaintext
IMPORT STD;
StartDate := 19631122;
numMonths := STD.Date.MonthsBetween(startDate,STD.Date.Today());
// numMonths contains the number of months between the startDate and today's date
```
AdjustDate

STD.Date.AdjustDate( date , [year_delta],[month_delta] ,[day_delta] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>A date value in the Date_t format.</td>
</tr>
<tr>
<td>year_delta</td>
<td>The minimum acceptable year. Optional; defaults to zero.</td>
</tr>
<tr>
<td>month_delta</td>
<td>The minimum acceptable year. Optional; defaults to zero.</td>
</tr>
<tr>
<td>day_delta</td>
<td>The maximum acceptable year. Optional; defaults to zero.</td>
</tr>
</tbody>
</table>

Return: AdjustDate returns date_t representing the adjusted date.

The AdjustDate function 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 is normalized according to mktime() rules. For example, 20140130 plus 1 month would be 20140302.

Example:

```plaintext
IMPORT std;
inDate :=19631123;
Std.Date.AdjustDate(inDate,5,1,3);  //returns 19681226
```

See Also: AdjustCalendar
AdjustCalendar

STD.Date.AdjustCalendar( date, [year_delta], [month_delta], [day_delta] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>A date value in the Date_t format.</td>
</tr>
<tr>
<td>year_delta</td>
<td>The minimum acceptable year. Optional; defaults to zero.</td>
</tr>
<tr>
<td>month_delta</td>
<td>The minimum acceptable year. Optional; defaults to zero.</td>
</tr>
<tr>
<td>day_delta</td>
<td>The maximum acceptable year. Optional; defaults to zero.</td>
</tr>
</tbody>
</table>

Return: AdjustDate returns date_t representing the adjusted date.

The AdjustCalendar function adjusts a date by incrementing or decrementing months and/or years. The date must be in the Gregorian calendar after the year 1600.

This 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.”

Note that day adjustments are performed after year and month adjustments using the preceding rules.


Example:

```pascal
IMPORT std;
inDate := 19631123;
Std.Date.AdjustCalendar(inDate, 5, 1, 3);  //returns 19681226
```

See Also: AdjustDate
MonthWeekNumFromDate

STD.Date.MonthWeekNumFromDate( date, startingDayOfWeek)

date: The date (in Date_t format) for which to compute the week number.

startingDayOfWeek: Optional, The index number of the first day of a week, 1-7, where 1 = Sunday. Default is 1.

Return: The 1-based week number of the date, relative to the beginning of the date's month.

The WeekNumFromDate function returns the 1-based week number of a date within the date's month. Week 1 always contains the first day of the month, and week 2 begins on the following day of the week indicated by the value of startingDayOfWeek.

This is not an ISO-8601 implementation of computing week numbers ("week dates").

Example:

```pascal
IMPORT STD;
startDate := STD.Date.Today();
weekNum := STD.Date.MonthWeekNumFromDate(startDate,2);
weekNum;
```

See Also: YearWeekNumFromDate
YearWeekNumFromDate

STD.Date.YearWeekNumFromDate( date, startingDayOfWeek)

<table>
<thead>
<tr>
<th>date</th>
<th>The date (in Date_t format) for which to compute the week number.</th>
</tr>
</thead>
<tbody>
<tr>
<td>startingDayOfWeek</td>
<td>Optional, The index number of the first day of a week, 1-7, where 1 = Sunday. Default is 1.</td>
</tr>
</tbody>
</table>

Return: The 1-based week number of the date, relative to the beginning of the date's year.

The YearWeekNumFromDate function returns the 1-based week number of a date within the date's year. Week 1 always contains the first day of the year, and week 2 begins on the following day of the week indicated by the value of startingDayOfWeek.

This is not an ISO-8601 implementation of computing week numbers ("week dates").

Example:

```plaintext
IMPORT STD;
startDate := STD.Date.Today();
weekNum := STD.Date.YearWeekNumFromDate(startDate,2);
weekNum;
```

See Also: MonthWeekNumFromDate
**TimestampToString**

**STD.Date.TimestampToString** *(timestamp, format)*

<table>
<thead>
<tr>
<th><strong>timestamp</strong></th>
<th>An INTEGER8 holding the number of microseconds since epoch (January 1, 1970 UTC)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>format</strong></td>
<td>OPTIONAL. The format of the string to return. See strftime documentation for details (<a href="http://strftime.org/">http://strftime.org/</a>). If omitted, it defaults to '%Y-%m-%dT%H:%M:%S.%@' which is YYYY-MM-DDTHH:MM:SS.sssss.</td>
</tr>
</tbody>
</table>

**Return:** The converted *timestamp* as a string in the specified format.

The **TimestampToString** function converts a Timestamp_t value containing the number of microseconds since epoch (January 1, 1970 UTC) into a human-readable string using a format template of strftime standards. Two additional format specifiers are available to show fractional seconds:

| %@ | Fraction of seconds in microseconds (6 digits) |
| %# | Fraction of seconds in milliseconds (3 digits) |

Milliseconds fractions are truncated from microseconds when necessary.

The maximum length of the resulting string is 255 characters.

**Example:**

```plaintext
IMPORT STD;
STD.Date.TimestampToString(1048998120000000, '%A %B %d, %Y T%H:%M:%S.%#');
// returns Sunday March 30, 2003 T04:22:00.000
```
UniqueTZAbbreviations

STD.Date.TimeZone.UniqueTZAbbreviations( )

Returns: A new DATASET({STRING5 tzAbbrev}) containing the unique time zone abbreviations.

The STD.Date.TimeZone.UniqueTZAbbreviations function returns a list of unique time zone abbreviations from the hardcoded dataset in the TimeZone module. All abbreviations are in uppercase.

Example:

IMPORT STD;
STD.Date.TimeZone.UniqueTZAbbreviations();
UniqueTZLocations

STD.Date.TimeZone.UniqueTZLocations()

Returns: A new DATASET({STRING name}) containing the unique location names.

The STD.Date.TimeZone.UniqueTZLocations function Return a list of unique location names from the hardcoded dataset. All names are in uppercase.

Example:

IMPORT STD;  
STD.Date.TimeZone.UniqueTZLocations();
TZDataForLocation

STD.Date.TimeZone.TZDataForLocation( location )

<table>
<thead>
<tr>
<th>location</th>
<th>REQUIRED. The name of the location to search for; must be a non-empty uppercase string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Returns:</td>
<td>A new DATASET(STRING5 tzAbbrev, INTEGER4 secondsOffset) containing the records found for the given location.</td>
</tr>
</tbody>
</table>

The **STD.Date.TimeZone.TZDataForLocation** function returns the time zone records for a given location.

Example:

```plaintext
IMPORT STD;
STD.Date.TimeZone.TZDataForLocation('ASIA');
```

See Also: FindTZData
FindTZData

STD.Date.TimeZone.FindTZData( timeZoneAbbrev, [location ])

<table>
<thead>
<tr>
<th>timeZoneAbbrev</th>
<th>REQUIRED. The time zone abbreviation to search for; must be a non-empty uppercase string.</th>
</tr>
</thead>
<tbody>
<tr>
<td>location</td>
<td>OPTIONAL. The name of the location to search for; if a location is not provided or is an empty string, all records matching only the abbreviation are returned.</td>
</tr>
</tbody>
</table>

Returns: A new DATASET(TZDataLayout) containing the found records.

EXPORT TZDataLayout := RECORD
  STRING5  tzAbbrev;       // Time zone abbreviation; always uppercase
  // may be duplicated between records
  INTEGER4  secondsOffset;  // Number of seconds east (positive) or west (negative) of UTC
  // or west (negative) of UTC
  SET OF STRING15 locations;      // Names of locations that use the given time zone abbreviation
END;

The STD.Date.TimeZone.TZDataForLocation function returns the time zone records for a given abbreviation and optional location. A location should be provided as a method of differentiation if the abbreviation has duplicate entries.

Example:

IMPORT STD;
STD.Date.TimeZone.FindTZData('CST','NORTH AMERICA');

See Also: TZDataForLocation
SecondsBetweenTZ

STD.Date.TimeZone.SecondsBetweenTZ( fromTimeZoneAbbrev, toTimeZoneAbbrev, [fromLocation, ] [toLocation] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromTimeZoneAbbrev</td>
<td>REQUIRED. The time zone abbreviation designated as the starting point; must be a non-empty uppercase string.</td>
</tr>
<tr>
<td>toTimeZoneAbbrev</td>
<td>REQUIRED. The time zone abbreviation designated as the ending point; must be a non-empty uppercase string.</td>
</tr>
<tr>
<td>fromLocation</td>
<td>OPTIONAL. The name of the location that goes along with fromTimeZoneAbbrev; if a location is not provided or is an empty string, the first record matching fromTimeZoneAbbrev is used.</td>
</tr>
<tr>
<td>toLocation</td>
<td>OPTIONAL. The name of the location that goes along with toTimeZoneAbbrev; if a location is not provided or is an empty string, the first record matching toTimeZoneAbbrev is used.</td>
</tr>
</tbody>
</table>

Returns: The number of seconds between the two time zones; returns zero if either time zone cannot be found

The STD.Date.TimeZone.SecondsBetweenTZ function computes the offset, in seconds, between two different time zones. Each time zone is designated by a required time zone abbreviation and an optional location name. The result is the number of seconds (which can be either positive or negative) that would have to be applied to a time when traveling from fromTimeZoneAbbrev to toTimeZoneAbbrev.

Be aware that some time zones explicitly represent daylight savings time, so it is entirely possible to change not only time zones but DST observance as well in a single call.

Example:

```
IMPORT STD;
STD.Date.TimeZone.SecondsBetweenTZ('CST','IST','NORTH AMERICA','');
```

See Also: AdjustTimeTZ
### AdjustTimeTZ

**STD.Date.TimeZone.AdjustTimeTZ**

STD.Date.TimeZone.AdjustTimeTZ( time, fromTimeZoneAbbrev, toTimeZoneAbbrev, [fromLocation, ] [toLocation] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>REQUIRED. The time value (in Time_t format) to adjust.</td>
</tr>
<tr>
<td>fromTimeZoneAbbrev</td>
<td>REQUIRED. The time zone abbreviation that the time value is assumed to be within; must be a non-empty uppercase string.</td>
</tr>
<tr>
<td>toTimeZoneAbbrev</td>
<td>REQUIRED. The time zone abbreviation designated as the ending point; must be a non-empty uppercase string.</td>
</tr>
<tr>
<td>fromLocation</td>
<td>OPTIONAL. The name of the location that goes along with fromTimeZoneAbbrev; if a location is not provided or is an empty string, the first record matching fromTimeZoneAbbrev is used.</td>
</tr>
<tr>
<td>toLocation</td>
<td>OPTIONAL. The name of the location that goes along with toTimeZoneAbbrev; if a location is not provided or is an empty string, the first record matching toTimeZoneAbbrev is used.</td>
</tr>
</tbody>
</table>

**Returns:** The given time value (in Time_t format) adjusted by the difference between the two given time zones; if either time zone cannot be found then the original time value is returned unchanged.

The STD.Date.TimeZone.AdjustTimeTZ function adjusts a given Time_t time value for another time zone. Both the given time and the destination time zone are designated by a required time zone abbreviation and an optional location name.

**Example:**

```plaintext
IMPORT STD;
STD.Date.TimeZone.AdjustTimeTZ(205246,'CST','IST','NORTH AMERICA','');
```

See Also: SecondsBetweenTZ
ToLocalTime

STD.Date.TimeZone.ToLocalTime( utcTime, timeZoneAbbrev, [location ] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>utcTime</td>
<td>REQUIRED. The UTC time value (in Time_t format) to adjust.</td>
</tr>
<tr>
<td>timeZoneAbbrev</td>
<td>REQUIRED. The time zone abbreviation designated as the ending point; must be a non-empty uppercase string.</td>
</tr>
<tr>
<td>location</td>
<td>OPTIONAL. The name of the location that goes along with timeZoneAbbrev; if a location is not provided or is an empty string, the first record matching timeZoneAbbrev is used.</td>
</tr>
</tbody>
</table>

Returns: The given UTC time value (in Time_t format) adjusted to the time zone defined by timeZoneAbbrev and location; if the time zone cannot be found then the original time value is returned unchanged.

The STD.Date.TimeZone.ToLocalTime function converts a UTC time to a time designated by a time zone abbreviation and optional location.

Example:

```plaintext
IMPORT STD;
STD.Date.TimeZone.ToLocalTime(205246,'CST','NORTH AMERICA');
```

See Also: AdjustTimeTZ, ToUTCTime
ToUTCTime

STD.Date.TimeZone.ToUTCTime( localTime, fromTimeZoneAbbrev, [fromLocation ] )

localTime | REQUIRED. The time value (in Time_t format) to adjust.
fromTimeZoneAbbrev | REQUIRED. The time zone abbreviation that the localTime value is assumed to be within; must be a non-empty uppercase string.
fromLocation | OPTIONAL. The name of the location that goes along with fromTimeZoneAbbrev; if a location is not provided or is an empty string, the first record matching fromTimeZoneAbbrev is used.

Returns: The given local time value adjusted to UTC time; if the given time zone cannot be found then the original UTC time value is returned unchanged.

The STD.Date.TimeZone.ToUTCTime function converts a local time, defined with a time zone abbreviation and optional location, to a UTC time.

Example:

IMPORT STD;
STD.Date.TimeZone.ToUTCTime(205246,'CST','NORTH AMERICA');

See Also: AdjustTimeTZ, ToLocalTime
## AppendTZOffset

**STD.Date.TimeZone.AppendTZOffset**

### Parameters:

- **inFile** (REQUIRED): The dataset to process.
- **timeZoneAbbrevField** (REQUIRED): The field within inFile that contains the time zone abbreviation to use for matching; the values in this field should be uppercase. This is not a string.
- **newOffsetField** (REQUIRED): The field to append to inFile that will contain the number of seconds offset from UTC. This is not a string.
- **fromLocationField** (OPTIONAL): The field within inFile that contains the time zone location for the time zone cited by timeZoneAbbrevField. This is not a string. Defaults to a null value (indicating that there is no time zone location field).
- **toTimeZoneAbbrev** (OPTIONAL): The to time zone abbreviation to use for all calculations, as a string. Defaults to 'UTC'.
- **toLocation** (OPTIONAL): The name of the location that goes along with toTimeZoneAbbrev; if a location is not provided or is an empty string, the first record matching toTimeZoneAbbrev is used. Defaults to an empty string.

### Returns:

A new dataset with the same record definition as inFile but with four new fields added. The new fields are named based on the name given as the newOffsetField attribute. The appended fields are:

- **INTEGER4 <newOffsetField>** // Offset, in seconds, between original time zone and toTimeZoneAbbrev
- **BOOLEAN <newOffsetField>_is_valid** // TRUE if <newOffsetField> contains a valid value
- **STRING5 <newOffsetField>_tz** // The value of toTimeZoneAbbrev
- **STRING15 <newOffsetField>_location** // The time zone location for <newOffsetField>_tz.

The **STD.Date.TimeZone.AppendTZOffset** takes a dataset that contains a time zone abbreviation and optional location, and appends four new attributes to the dataset that contain useful information for translating a time value into another time zone.

This could be useful as an ETL step where time data is made common in respect to one particular time zone (e.g., UTC). The actions within this function macro are conceptually similar to SecondsBetweenTZ() but applied to an entire dataset, and somewhat more efficiently.

**Note:** In order for this function macro to execute correctly, the calling code must import the Std library.
Example:

```plaintext
IMPORT STD;

ds := DATASET([
    {120000, 'CT'},
    {120000, 'ET'}
], {Std.Date.Time_t time, STRING tz});

utcOffsetDS := Std.Date.TimeZone.AppendTZOffset(ds, tz, seconds_to_utc);
OUTPUT(utcOffsetDS, NAMED('offset_to_utc_result'));

ptOffsetDS := Std.Date.TimeZone.AppendTZOffset (ds, tz, seconds_to_pacific_time,
    timeZoneAbbrev := 'PT',
    toLocation := 'NORTH AMERICA');
OUTPUT(ptOffsetDS, NAMED('offset_to_pacific_time_result'));
```

See Also: AppendTZAdjustedTime, SecondsBetweenTZ
AppendTZAdjustedTime

STD.Date.TimeZone.AppendTZAdjustedTime(  infile,  timeField,  timeZoneAbbrevField,  newTimeField,  [fromLocationField, ] [toTimeZoneAbbrev, ] [toLocation] )

infile REQUIRED. The dataset to process.
timeField REQUIRED. The field within inFile that contains a time represented in Time_t format. This is not a string.
timeZoneAbbrevField REQUIRED. The field within inFile that contains the time zone abbreviation to use for matching; the values in this field should be uppercase.
newTimeField REQUIRED. The field to append to inFile that will contain the adjusted value of timeField.
fromLocationField OPTIONAL. The field within inFile that contains the time zone location for the time zone cited by timeZoneAbbrevField. Defaults to a null value (indicating that there is no time zone location attribute.) If a location is not provided or is an empty string, the first record matching fromTimeZoneAbbrevField is used.
toTimeZoneAbbrev OPTIONAL. The to time zone abbreviation to use for all calculations, as a string. Defaults to ‘UTC’
toLocation OPTIONAL. The name of the location that goes along with toTimeZoneAbbrev; if a location is not provided or is an empty string, the first record matching toTimeZoneAbbrev is used; Defaults to an empty string

Returns: A new dataset with the same record definition as inFile but with four new fields added; the new fields are named based on the name given as the newOffsetField attribute:

| std.Date.Time_t <newOffsetField> | // Value of timeField expressed in new time zone |
| BOOLEAN <newOffsetField>_is_valid | // TRUE if <newOffsetField> contains a valid value |
| STRING5 <newOffsetField>_tz | // If <newOffsetField>_is_valid is FALSE then <newOffsetField> will have the same value as timeField. |
| STRING15 <newOffsetField>_location | // The time zone location for <newOffsetField>_tz |

The STD.Date.TimeZone.AppendTZAdjustedTime takes a given a dataset that contains a time (in Time_t format), a time zone abbreviation, and an optional time zone location, and appends four new fields to the dataset: A new Time_t attribute containing the original time expressed in a different time zone, and three attributes providing information regarding that destination time zone and the validity of the translation.

This could be useful as an ETL step where time data is made common in respect to one particular time zone (e.g., UTC). The actions within this function macro are conceptually similar to AdjustTimeTZ() but applied to an entire dataset, and somewhat more efficiently.

Note: In order for this function macro to execute correctly, the calling code must import the STD library.
Example:

```java
IMPORT STD;

dS := DATASET ( [
  {120000, 'CT'},
  {120000, 'ET'}
], (Std.Date.Time_t time, STRING tz));

utcRewriteDS := Std.Date.TimeZone.AppendTZAdjustedTime(ds, time, tz, utc_time);
OUTPUT(utcRewriteDS, NAMED('utc_result'));

ptRewriteDS := Std.Date.TimeZone.AppendTZAdjustedTime (ds, time, tz, pacific_time,
  toTimeZoneAbbrev := 'PT',
  toLocation := 'NORTH AMERICA');
OUTPUT(ptRewriteDS, NAMED('pacific_time_result'));
```

See Also: AppendTZOffset, AdjustTimeTZ
Cluster Handling
Node

STD.System.Thorlib.Node()

| Return: | Node returns an UNSIGNED INTEGER4 value. |

The Node function returns the (zero-based) number of the Data Refinery (Thor) or Rapid Data Delivery Engine (Roxie) node.

Example:

A := STD.System.Thorlib.Node();
## Nodes

**STD.System.Thorlib.Nodes()**

<table>
<thead>
<tr>
<th>Return:</th>
<th>Nodes returns an UNSIGNED INTEGER4 value.</th>
</tr>
</thead>
</table>

The **Nodes** function returns the number of nodes in the Thor cluster (always returns 1 on hThor and Roxie). This number is the same as the CLUSTERSIZE compile time constant. The Nodes function is evaluated each time it is called, so the choice to use the function versus the constant depends upon the circumstances.

Example:

```plaintext
A := STD.System.Thorlib.Nodes();
```
LogicalToPhysical

STD.System.Thorlib.LogicalToPhysical ( filename [, createflag ] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename</td>
<td>A null-terminated string containing the logical name of the file.</td>
</tr>
<tr>
<td>createflag</td>
<td>A boolean value indicating whether to create the filename. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>

Return: LogicalToPhysical returns a VARSTRING value.

The LogicalToPhysical function (Logical to Physical) returns the physical name of the file represented by the logical filename.

Example:

A := STD.System.Thorlib.LogicalToPhysical('Fred');
DaliServer

STD.System.Thorlib.DaliServer ( )

Return: DaliServer returns a VARSTRING value.

The DaliServer function returns the IP and port of the system data store (Dali) server for the environment running the workunit.

Example:

```plaintext
IMPORT STD;
A := STD.System.Thorlib.Daliserver();
```
**Group**

STD.System.Thorlib.Group ()

| Return: | Group returns a VARSTRING value. |

The **Group** function returns the name of the node group running the workunit. This name is used in ECL code to specify the target CLUSTER for an OUTPUT action or a PERSISTed attribute.

Example:

```ecl
IMPORT STD;
A := STD.System.Thorlib.Group();
```
# GetExpandLogicalName

**ThorLib.GetExpandLogicalName**  

_{filename}_  

- **filename**: A null-terminated string containing the logical name of the file.  
- **Return**: GetExpandLogicalName returns a VARSTRING (null-terminated) value.  

The *GetExpandLogicalName* function returns a string containing the expanded logical filename (including the default scope, if the filename does not contain a leading tilde), all in lowercase. This is the same value as is used internally by DATASET and OUTPUT.

**Example:**

```plaintext
IMPORT STD;  
A := STD.System.ThorLib.GetExpandLogicalName('Fred');
```
Job Handling
WUID

STD.System.Job.WUID ( )

Return: WUID returns a VARSTRING value.

The WUID function returns the workunit identifier of the current job. This is the same as the WORKUNIT compile time constant.

Example:

A := STD.System.Job.WUID();
Target

STD.System.Job.Target( )

| Return: | Target returns a VARSTRING value. |

The **Target** function returns the name of the cluster running the workunit. Not supported on Roxie clusters. This name is used by #WORKUNIT, the ecl command line utility, or the eclplus command line utility to specify the target cluster for a workunit.

Example:

```plaintext
A := STD.System.Job.Target();
```
Name

STD.System.Job.Name ( )

Return: Name returns a VARSTRING value.

The **Name** function returns the name of the workunit.

Example:

```plaintext
A := STD.System.Job.Name();
```
User

STD.System.Job.User  ()

| Return:          | User returns a VARSTRING value. |

The **User** function returns the username of the person running the workunit.

Example:

```plaintext
```
OS

STD.System.Job.OS ( )

Return: OS returns a VARSTRING value.

The OS function returns the operating system (windows or Linux) of the cluster running the workunit.

Example:

A := STD.System.Job.OS();
Platform

STD.System.Job.Platform ( )

Return: Platform returns a VARSTRING value.

The Platform function returns the platform name (hthor, thor, or roxie) of the cluster running the workunit.

Example:

LogString

\texttt{STD.System.Job.LogString ( \textit{message} )}

<table>
<thead>
<tr>
<th>\textit{message}</th>
<th>A string expression containing the text to place in the log file.</th>
</tr>
</thead>
</table>

Return: LogString returns an INTEGER value.

The \texttt{LogString} function outputs "USER:" followed by the \textit{message} text to the eclagent or Roxie log file and returns the length of the text written to the file.

Example:

\begin{verbatim}
A := STD.System.Job.LogString('The text message to log');
//places USER:The text message to log
//in the log file
\end{verbatim}
**MonitorFile**

STD.File.MonitorFile(  
  event, [ ip ],  
  filename, [ ,subdirs ] [ ,shotcount ] [ ,espserverIPport ] )

\[ dfuwuid := STD.File.fMonitorFile(  
  event, [ ip ],  
  filename, [ ,subdirs ] [ ,shotcount ] [ ,espserverIPport ] ) ; \]

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>A null-terminated string containing the user-defined name of the event to fire when the <code>filename</code> appears. This value is used as the first parameter to the EVENT function.</td>
</tr>
<tr>
<td>ip</td>
<td>Optional. A null-terminated string containing the ip address for the file to monitor. This is typically a landing zone. This may be omitted only if the <code>filename</code> parameter contains a complete URL.</td>
</tr>
<tr>
<td>filename</td>
<td>A null-terminated string containing the full path to the file to monitor. This may contain wildcard characters (* and ?).</td>
</tr>
<tr>
<td>subdirs</td>
<td>Optional. A boolean value indicating whether to include files in sub-directories that match the wildcard mask when the <code>filename</code> contains wildcards. If omitted, the default is false.</td>
</tr>
<tr>
<td>shotcount</td>
<td>Optional. An integer value indicating the number of times to generate the event before the monitoring job completes. A negative one (-1) value indicates the monitoring job continues until manually aborted. If omitted, the default is 1.</td>
</tr>
<tr>
<td>espserverIPport</td>
<td>Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with &quot;/FileSpray&quot; appended.</td>
</tr>
</tbody>
</table>

**Return:** fMonitorFile returns a null-terminated string containing the DFU workunit ID (DFUWUID).

The `MonitorFile` function creates a file monitor job in the DFU Server. Once the job is received it goes into a ‘monitoring’ mode (which can be seen in the ECL Watch DFU Workunit display), which polls at a fixed interval. This interval is specified in the DFU Server’s `monitorinterval` configuration setting. The default interval is 900 seconds (15 minutes). 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).

This process continues until either:

1) The `shotcount` number of events have been generated.

2) The user aborts the DFU workunit.

The `STD.File.AbortDfuWorkunit` and `STD.File.WaitDfuWorkunit` functions can be used to abort or wait for the DFU job by passing them the returned `dfuwuid`.

**Note the following caveats and restrictions:**

1) Events are only generated when the monitor job starts or subsequently on the polling interval.

2) Note that the `event` is generated if the file has been created since the last polling interval. Therefore, the `event` may occur before the file is closed and the data all written. To ensure the file is not subsequently read before it is complete you should use a technique that will preclude this possibility, such as using a separate ‘flag’ file instead of the file, itself or renaming the file once it has been created and completely written.
3) The EVENT function's subtype parameter (its 2nd parameter) when monitoring physical files is the full URL of the file, with an absolute IP rather than DNS/netbios name of the file. This parameter cannot be retrieved but can only be used for matching a particular value.

Example:

```plaintext
EventName := 'MyFileEvent';
FileName := 'c:\test\myfile';
LZ := '10.150.50.14';
STD.File.MonitorFile(EventName, LZ, FileName);
OUTPUT('File Found') : WHEN(EVENT(EventName, '*'), COUNT(1));
```
MonitorLogicalFileName

```
STD.File.MonitorLogicalFileName( event, filename, [ , shotcount ][ , espserverIPport ] )
```

```
dfuwuid := STD.File.fMonitorLogicalFileName( event, filename, [ , shotcount ][ , espserverIPport ] );
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event</td>
<td>A null-terminated string containing 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.</td>
</tr>
<tr>
<td>filename</td>
<td>A null-terminated string containing the name of the logical file in the DFU to monitor.</td>
</tr>
<tr>
<td>shotcount</td>
<td>Optional. An integer value indicating the number of times to generate the event before the monitoring job completes. A negative one (-1) value indicates the monitoring job continues until manually aborted. If omitted, the default is 1.</td>
</tr>
<tr>
<td>espserverIPport</td>
<td>Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with &quot;/FileSpray&quot; appended.</td>
</tr>
</tbody>
</table>

| dfuwuid    | The attribute name to receive the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the monitoring job. |

**Return:**
- fMonitorLogicalFileName returns a null-terminated string containing the DFU workunit ID (DFUWUID).

The MonitorLogicalFileName function creates a file monitor job in the DFU Server. Once the job is received it goes into a 'monitoring' mode (which can be seen in the eclwatch DFU Workunit display), which polls at a fixed interval (default 15 mins). 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).

This function does not support wildcard characters. To monitor physical files or directories using wildcards, use the MonitorFile function.

This process continues until either:

1) The shotcount number of events have been generated.

2) The user aborts the DFU workunit.

The STD.File.AbortDfuWorkunit and STD.File.WaitDfuWorkunit functions can be used to abort or wait for the DFU job by passing them the returned dfuwuid.

**Note the following caveats and restrictions:**

1) If a matching file already exists when the DFU Monitoring job is started, that file will not generate an event. It will only generate an event once the file has been deleted and recreated.

2) If a file is created and then deleted (or deleted then re-created) between polling intervals, it will not be seen by the monitor and will not trigger an event.

3) Events are only generated on the polling interval.

**Example:**

```plaintext
EventName := 'MyFileEvent';
FileName := 'test::myfile';
```
IF (STD.File.FileExists(FileName),
   STD.File.DeleteLogicalFile(FileName));
STD.File.MonitorLogicalFileName(EventName,FileName);
OUTPUT('File Created') : WHEN(EVENT(EventName,'*'),COUNT(1));

rec := RECORD
   STRING10 key;
   STRING10 val;
END;
afile := DATASET([{'A', '0'}], rec);
OUTPUT(afile,,FileName);
Logging
The `dbglog` action writes the `text` string to the eclagent.log file for the workunit.

In a containerized platform deployment, this action writes the `text` string to the eclagent pod's log, accessible using this command:

```
kubectl logs <podname>
```

Example:

```
IMPORT STD;
STD.System.Log.dbglog('Got Here 1');  //write text to log
```
addWorkunitInformation

STD.System.Log.addWorkunitInformation ( text [, code ])

<table>
<thead>
<tr>
<th>text</th>
<th>A string containing the text to write.</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>Optional. The code number to associate with the text. If omitted, the default is zero (0).</td>
</tr>
</tbody>
</table>

Return: addWorkunitInformation does not return a value.

The `addWorkunitInformation` action writes the `text` string to the eclagent.log file for the workunit, and also displays the `code` and `text` in the Info section of the ECL Watch page for the workunit.

Example:

```plaintext
IMPORT STD;
STD.System.Log.addWorkunitInformation('Got Here',1);  //write text to log and display "1: Got Here" as Info
```
addWorkunitWarning

STD.System.Log.addWorkunitWarning ( text [, code ])

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>A string containing the text to write.</td>
</tr>
<tr>
<td>code</td>
<td>Optional. The code number to associate with the text. If omitted, the default is zero (0).</td>
</tr>
</tbody>
</table>

Return: addWorkunitWarning does not return a value.

The addWorkunitWarning action writes the text string to the eclagent.log file for the workunit, and also displays the code and text in the Syntax Errors toolbox along with the Warnings section of the ECL Watch page for the workunit.

Example:

```plaintext
IMPORT STD;
STD.System.Log.addWorkunitWarning('Got Here',1);
//write text to log and display "1: Got Here" in Warnings
```
addWorkunitError

STD.System.Log.addWorkunitError ( text [ , code ] )

<table>
<thead>
<tr>
<th>text</th>
<th>A string containing the text to write.</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>Optional. The code number to associate with the text. If omitted, the default is zero (0).</td>
</tr>
</tbody>
</table>

Return: addWorkunitError does not return a value.

The addWorkunitError action writes the text string to the eclagent.log file for the workunit, and also displays the code and text in the Syntax Errors toolbox along with the Errors section of the ECL Watch page for the workunit.

Example:

```ecl
IMPORT STD;
STD.System.Log.addWorkunitError('Got Here',1);
//write text to log and display "1: Got Here" in Errors
```
### getGlobalId

STD.System.Log.getGlobalId ()

| Return:       | getGlobalId returns the Global Id |

The `getGlobalId` gets the Global Id associated with the current query or workunit. Example:

```plaintext
IMPORT STD;
STD.System.Log.getGlobalId();
```
getLocalId

STD.System.Log.getLocalId()

Return: getLocalId returns the Local Id

The getLocalId gets the Local Id associated with the current query or workunit. Example:

```cpp
IMPORT STD;
STD.System.Log.getLocalId();
```
The `generateGloballyUniqueID` returns a globally unique identifier (GUID) with base58 encoding. Base58 encoding is similar to base64 encoding but avoids both non-alphanumeric characters and visually ambiguous letters. It is designed to avoid errors by human users who manually enter the data by copying from some visual source. It allows easy copy/paste because a double-click will usually select the entire string.

```
IMPORT STD;

value1 := std.system.log.generateGloballyUniqueId() : INDEPENDENT;
value2 := NOFOLD(std.system.log.generateGloballyUniqueId()) : INDEPENDENT;

OUTPUT(value1);
OUTPUT(value2);
OUTPUT(IF (value1 = value2, 'Values are not unique', 'Values are unique'));
```
getElapsedTimeMs

\[ \text{result} := \text{STD.System.Log.getElapsedTimeMs}(); \]

Return: getElapsedTimeMs returns the elapsed time in milliseconds.

The **getElapsedTimeMs** function returns the current elapsed query time (in ms) in Roxie. This is the elapsed time when STD.System.Log.getElapsedTimeMs() is called. Because ECL is a declarative language, code is not necessarily executed in sequence. You have to be careful when trying to get the elapsed time for a particular point in your code. You can look at the Workunit graphs to see the exact point at which the activity executes.

**For use in Roxie only.** An error is returned if you try to run on Thor or hThor.

Example:

```
IMPORT STD;
STD.System.Debug.Sleep (1054);    // pause processing for 1054 milliseconds.
OUTPUT(STD.System.Log.getElapsedTimeMs(), NAMED('Elapsed')); //returns total time elapsed
```
**Audit**

STD.Audit.Audit(  type, message  )

<table>
<thead>
<tr>
<th>type</th>
<th>A string constant containing the type of audit entry. Currently, only INFO is provided.</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>A string containing the audit entry text.</td>
</tr>
</tbody>
</table>

Return: Audit returns a BOOLEAN value indicating whether it was successful or not.

The Audit function writes the message into the Windows event log or Linux system log on the ECL Agent computer. The entries can be retrieved from the logs using standard operating system tools.

Example:

```
STD.Audit.Audit('INFO','Audit Message');
```

GetHostName

\[
result := \text{STD.System.Util.GetHostName} \left( \text{ip} \right);
\]

<table>
<thead>
<tr>
<th>\text{ip}</th>
<th>A null-terminated string containing the IP address of the remote machine.</th>
</tr>
</thead>
</table>

Return: GetHostName returns a VARSTRING (null-terminated) value.

The GetHostName function does a reverse DNS lookup to return the host name for the machine at the specified ip address.

Example:

```plaintext
IP := '10.150.254.6';

OUTPUT(STD.System.Util.GetHostName(IP));
```

© 2024 HPCC Systems®. All rights reserved
ResolveHostName

\[
\text{result} := \text{STD.System.Util.ResolveHostName ( host )};
\]

<table>
<thead>
<tr>
<th>\text{host}</th>
<th>A null-terminated string containing the DNS name of the remote machine.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{Return}</td>
<td>ResolveHostName returns returns a VARSTRING (null-terminated) value.</td>
</tr>
</tbody>
</table>

The \textbf{ResolveHostName} function does a DNS lookup to return the ip address for the specified \textit{host} name.

Example:

\[
\text{host} := \text{'dataland_dali.br.seisint.com'};
\]
\[
\text{OUTPUT(STD.System.Util.ResolveHostName(host))};
\]
GetUniqueInteger

\texttt{result := STD.System.Util.GetUniqueInteger ([ \ dali \ ]);}

\begin{tabular}{|l|l|}
\hline
\textit{dali} & Optional. A null-terminated string containing the ip address of the remote dali to provide the number. If omitted, the default is local. \\
\hline
\textbf{Return:} & GetUniqueInteger returns returns an UNSIGNED8 value. \\
\hline
\end{tabular}

The \texttt{GetUniqueInteger} function returns a number that is unique across all the worker nodes of the specified \textit{dali}.

Example:

\begin{verbatim}
IMPORT STD;

OUTPUT(STD.System.Util.GetUniqueInteger());
\end{verbatim}
GetEspUrl

result := STD.File.GetEspUrl ( [ username, userPW ]); 

<table>
<thead>
<tr>
<th>username</th>
<th>Optional. A STRING containing a username to use for authenticated access to the ESP process. If omitted, it indicates that no user authentication is required.</th>
</tr>
</thead>
<tbody>
<tr>
<td>userPW</td>
<td>Optional. A STRING containing the password to use with the user cited in the username argument. If username is empty then this is ignored</td>
</tr>
</tbody>
</table>

Return: GetEspUrl returns a STRING containing the full URL (including HTTP scheme and port) to an ESP server process. If more than one ESP process is defined then the first found process is returned. Returns an empty string if an ESP server process cannot be found in the environment.

The **GetEspUrl** function returns the full URL to an ESP server process.

Example:

```plaintext
IMPORT STD;
EspAddress := STD.File.GetEspUrl();
EspAddress;
```
# PlatformVersionCheck

\[ result := STD.System.Util.PlatformVersionCheck( v ); \]

| \( v \) | Required. The minimum platform version in either xx.xx.xx, xx.xx, or xx format (where xx is an integer and does not need to be zero-padded); extra trailing characters (such as the '-1' in the example below) are ignored. |
| Return: | TRUE if the platform's current version is equal to or higher than the argument, otherwise FALSE. |

The **PlatformVersionCheck** function tests a full version string against the individual platform version constants to determine if the platform's version is at least as high as the argument. This function is evaluated at compile-time if the argument is a constant. This makes it useful for embedding in #IF() declarations as shown in the example.

**Example:**

```plaintext
IMPORT STD;

#IF(STD.System.Util.PlatformVersionCheck('8.2.0-1'))

  OUTPUT('Platform check TRUE');

#ELSE

  OUTPUT('Platform check FALSE');

#END
```
GetParseTree

STD.System.Debug.GetParseTree ( )

Return: GetParseTree returns a STRING value.

The GetParseTree function returns a textual representation of the match that occurred, using square brackets (such as: a[b[c[d] ] ) to indicate nesting. This function is only used within the RECORD or TRANSFORM structure that defines the result of a PARSE operation. This function is useful for debugging PARSE operations.

Example:

IMPORT STD;

r := {string150 line};
d := dataset(
{"Ge 34:2 And when Shechem the son of Hamor the Hivite, '+
'prince of the country, saw her, he took her, and lay with her, '+
'and defiled her.'},
{"Ge 36:10 These are the names of Esau's sons; Eliphaz the son of '+
'Adah the wife of Esau, Reuel the son of Basemath the wife of '+
'Esau.'} ), r);

PATTERN ws := [ ' ', \t', ',']*;
PATTERN patStart := FIRST | ws;
PATTERN patEnd := LAST | ws;
PATTERN article := [ 'A', 'The', 'Thou', 'a', 'the', 'thou' ];
TOKEN patWord := PATTERN('[a-zA-Z]+');
TOKEN Name := PATTERN('^[A-Z][a-zA-Z]+');
RULE Namet := name OPT(ws 'the' ws name);
PATTERN produced_by := OPT(article ws) [ 'son of', 'daughter of' ];
PATTERN produces_with := OPT(article ws) [ 'wife of' ];
RULE progeny := namet ws ( produced_by | produces_with ) ws namet;
results := RECORD
  STRING LeftName   := MATCHTEXT(Namet[1]);
  STRING RightName  := MATCHTEXT(Namet[2]);
  STRING LinkPhrase := IF(MATCHTEXT(produced_by[1])<>'','
    MATCHTEXT(produced_by[1]),
    MATCHTEXT(produces_with[1]));
  STRING Tree := 'Tree: ' + STD.System.Debug.getParseTree();
END;

outfile1 := PARSE(d, line, progeny, results, SCAN ALL);
/* the Tree field output looks like this:
Tree: [namet[name"Shechem"] ws " produced_by"the son of" ws " namet[name"Hamor"]]
*/
GetXMLParseTree

STD.System.Debug.GetXMLParseTree()

Return: GetXMLParseTree returns a STRING value.

The GetXMLParseTree function returns a textual representation of the match that occurred, using XML tags to indicate nesting. This function is only used within the RECORD or TRANSFORM structure that defines the result of a PARSE operation. This function is useful for debugging PARSE operations.

Example:

```
IMPORT STD;

r := {string150 line};
d := dataset([{
  'Ge 34:2 And when Shechem the son of Hamor the Hivite, '+'
  'prince of the country, saw her, he took her, and lay with her, '+'
  'and defiled her.'},
  'Ge 36:10 These are the names of Esau’s sons; Eliphaz the son of '+'
  'Adah the wife of Esau, Reuel the son of Bashemath the wife of '+'
  'Esau.'}],r);

PATTERN ws := [׳ ,׳\‘,׳]*;
PATTERN patStart := FIRST | ws;
PATTERN patEnd := LAST | ws;
PATTERN article := [׳A’,׳The’,׳Thou’,׳a’,׳the’,׳thou’];
TOKEN patWord := PATTERN([a-zA-Z]+);
TOKEN Name := PATTERN([A-Z][a-zA-Z]+); RULE Namet := name OPT(ws ’the’ ws name);
PATTERN produced_by := OPT(article ws) [׳son of’,׳daughter of’];
PATTERN produces_with := OPT(article ws) [׳wife of’]; RULE progeny := namet ws ( produced_by | produces_with ) ws namet;
results := RECORD
  STRING LeftName := MATCHTEXT(Namet[1]);
  STRING RightName := MATCHTEXT(Namet[2]);
  STRING LinkPhrase := IF(MATCHTEXT(produced_by[1])<>'',
    MATCHTEXT(produced_by[1]),
    MATCHTEXT(produces_with[1]));
  STRING Tree := STD.System.Debug.getXMLParseTree();
END;
outfile1 := PARSE(d,line,progeny,results,SCAN ALL);
/* the Tree field output
looks like this:
  <namet>
    <name>Shechem</name>
  </namet>
  <ws> </ws>
  <produced_by>the son of</produced_by>
  <ws> </ws>
  <name>Hamor</name>
*/
```
Sleep

**STD.System.Debug.Sleep** ( *duration*)

<table>
<thead>
<tr>
<th><strong>duration</strong></th>
<th>An integer value specifying the length of the sleep period, in milliseconds.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Sleep does not return a value.</td>
</tr>
</tbody>
</table>

The **Sleep** function pauses processing for *duration* milliseconds.

Example:

```
IMPORT STD;
STD.System.Debug.Sleep(1000);  //pause for one second before continuing
```
msTick

STD.System.Debug.msTick ()

Return: msTick returns a 4-byte unsigned integer value.

The msTick function returns elapsed time since its start point, in milliseconds. The start point is undefined, making this function useful only for judging elapsed time between calls to the function by subtracting the latest return value from the earlier. When the return value reaches the maximum value of a 4-byte unsigned integer (2^32 or 4 Gb), it starts over again at zero (0). This occurs approximately every 49.71 days.

Example:

```plaintext
IMPORT STD;
t1 := STD.System.Debug.msTick() : STORED('StartTime');    //get start time

ds1 := DATASET([[0,0,0,0,0]],
    [UNSIGNED4 RecID,
     UNSIGNED4 Started,
     UNSIGNED4 ThisOne,
     UNSIGNED4 Elapsed,
     UNSIGNED4 RecsProcessed]);

RECORDOF(ds1) XF1(ds1 L, integer C) := TRANSFORM
    SELF.RecID := C;
    SELF := L;
END;
ds2 := NORMALIZE(ds1,100000,XF1(LEFT,COUNTER));

RECORDOF(ds1) XF(ds1 L) := TRANSFORM
    SELF.Started := T1;
    SELF.ThisOne := STD.System.Debug.msTick();
    SELF.Elapsed := SELF.ThisOne - SELF.Started;
    SELF := L;
END;

P := PROJECT(ds2,XF(LEFT)) : PERSIST('~RTTEST::TestTick');
R := ROLLUP(P,
    LEFT.Elapsed=RIGHT.Elapsed,
    TRANSFORM(RECORDOF(ds1),
        SELF.RecsProcessed := RIGHT.RecID - LEFT.RecID,
        SELF := LEFT));

paws := STD.System.Debug.Sleep(1000); //pause for one second before continuing

SEQUENTIAL(paws,OUTPUT(P, ALL),OUTPUT(R, ALL));
```
## SendEmail

**STD.System.Email.SendEmail** ( `to`, `subject`, `body`, `attachment`, `mimetype`, `filename`, `mailServer`, `port`, `sender`, `cc`, `bcc`, `highPriority` )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>to</code></td>
<td>A null-terminated string containing a comma-delimited list of the addresses of the intended recipients. The validity of the addresses is not checked, so it is the programmer's responsibility to ensure they are all valid.</td>
</tr>
<tr>
<td><code>subject</code></td>
<td>A null-terminated string containing the subject line.</td>
</tr>
<tr>
<td><code>body</code></td>
<td>A null-terminated string containing the text of the email to send. This must be character encoding &quot;ISO-8859-1 (latin1)&quot; (the ECL default character set). Text in any other character set must be sent as an attachment (see the STD.System.Email.SendEmailAttachText() function).</td>
</tr>
<tr>
<td><code>mailServer</code></td>
<td>Optional. A null-terminated string containing the name of the mail server. If omitted, defaults to the value in the SMTPserver environment variable.</td>
</tr>
<tr>
<td><code>port</code></td>
<td>Optional. An UNSIGNED4 integer value containing the port number. If omitted, defaults to the value in the SMTPport environment variable.</td>
</tr>
<tr>
<td><code>sender</code></td>
<td>Optional. A null-terminated string containing the address of the sender. If omitted, defaults to the value in the emailSenderAddress environment variable.</td>
</tr>
<tr>
<td><code>cc</code></td>
<td>Optional. comma-delimited addresses of carbon-copy recipients. Defaults to an empty string (none).</td>
</tr>
<tr>
<td><code>bcc</code></td>
<td>Optional. comma-delimited addresses of blind-carbon-copy recipients. Defaults to an empty string (none).</td>
</tr>
<tr>
<td><code>highPriority</code></td>
<td>Optional. If true, the message is sent with high priority. Defaults to false (normal priority).</td>
</tr>
</tbody>
</table>

The **SendEmail** function sends an email message.

Example:

```ecl
STD.System.Email.SendEmail( 'me@mydomain.com', 'testing 1,2,3', 'this is a test message');
```
**SendEmailAttachData**

*STD.System.Email.SendEmailAttachData* (  
  *to*, *subject*, *body*, *attachment*, *mimetype*, *filename*,  
  *mailServer*, *port*, *sendercc*, *bcc*, *highPriority* )

**to**
A null-terminated string containing a comma-delimited list of the addresses of the intended recipients. The validity of the addresses is not checked, so it is the programmer's responsibility to ensure they are all valid.

**subject**
A null-terminated string containing the subject line.

**body**
A null-terminated string containing the text of the email to send. This must be character encoding "ISO-8859-1 (latin1)" (the ECL default character set). Text in any other character set must be sent as an *attachment*.

**attachment**
A DATA value containing the binary data to attach.

**mimetype**
A null-terminated string containing the MIME-type of the *attachment*, which may include entrymeters (such as 'text/plain; charset=ISO-8859-3'). When attaching general binary data for which no specific MIME type exists, use 'application/octet-stream'.

**filename**
A null-terminated string containing the name of the *attachment* for the mail reader to display.

**mailServer**
Optional. A null-terminated string containing the name of the mail server. If omitted, defaults to the value in the SMTPserver environment variable.

**port**
Optional. An UNSIGNED4 integer value containing the port number. If omitted, defaults to the value in the SMTPport environment variable.

**sender**
Optional. A null-terminated string containing the address of the sender. If omitted, defaults to the value in the emailSenderAddress environment variable.

**cc**
Optional. comma-delimited addresses of carbon-copy recipients. Defaults to an empty string (none).

**bcc**
Optional. comma-delimited addresses of blind-carbon-copy recipients. Defaults to an empty string (none).

**highPriority**
Optional. If true, the message is sent with high priority. Defaults to false (normal priority).

The *SendEmailAttachData* function sends an email message with a binary *attachment*.

Example:

```ecl
DATA15 attachment := D'test attachment';
STD.System.Email.SendEmailAttachData( 'me@mydomain.com',  
  'testing 1,2,3',  
  'this is a test message',  
  attachment,  
  'application/octet-stream',  
  'attachment.txt');
```
## SendEmailAttachText

**STD.System.Email.SendEmailAttachText**  
(  
  to, subject, body, attachment, mimetype, filename,  
  mailServer, port, sender, cc, bcc, highPriority  
)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>to</strong></td>
<td>A null-terminated string containing a comma-delimited list of the addresses of the intended recipients. The validity of the addresses is not checked, so it is the programmer's responsibility to ensure they are all valid.</td>
</tr>
<tr>
<td><strong>subject</strong></td>
<td>A null-terminated string containing the subject line.</td>
</tr>
<tr>
<td><strong>body</strong></td>
<td>A null-terminated string containing the text of the email to send. This must be character encoding &quot;ISO-8859-1 (latin1)&quot; (the ECL default character set). Text in any other character set must be sent as an attachment.</td>
</tr>
<tr>
<td><strong>attachment</strong></td>
<td>A null-terminated string containing the text to attach.</td>
</tr>
<tr>
<td><strong>mimetype</strong></td>
<td>A null-terminated string containing the MIME-type of the attachment, which may include entrymeters (such as 'text/plain; charset=ISO-8859-3').</td>
</tr>
<tr>
<td><strong>filename</strong></td>
<td>A null-terminated string containing the name of the attachment for the mail reader to display.</td>
</tr>
<tr>
<td><strong>mailServer</strong></td>
<td>Optional. A null-terminated string containing the name of the mail server. If omitted, defaults to the value in the SMTPserver environment variable.</td>
</tr>
<tr>
<td><strong>port</strong></td>
<td>Optional. An UNSIGNED4 integer value containing the port number. If omitted, defaults to the value in the SMTPport environment variable.</td>
</tr>
<tr>
<td><strong>sender</strong></td>
<td>Optional. A null-terminated string containing the address of the sender. If omitted, defaults to the value in the emailSenderAddress environment variable.</td>
</tr>
<tr>
<td><strong>cc</strong></td>
<td>Optional. comma-delimited addresses of carbon-copy recipients. Defaults to an empty string (none).</td>
</tr>
<tr>
<td><strong>bcc</strong></td>
<td>Optional. comma-delimited addresses of blind-carbon-copy recipients. Defaults to an empty string (none).</td>
</tr>
<tr>
<td><strong>highPriority</strong></td>
<td>Optional. If true, the message is sent with high priority. Defaults to false (normal priority).</td>
</tr>
</tbody>
</table>

The **SendEmailAttachText** function sends an email message with a text attachment.

Example:

```ecl
STD.System.Email.SendEmailAttachText( 'me@mydomain.com', 'testing 1,2,3',  
  'this is a test message', 'this is a test attachment',  
  'text/plain; charset=ISO-8859-3', 'attachment.txt');
```
Workunit Services
## WorkunitExists

### STD.System.Workunit.WorkunitExists( wuid [, online ] [, archived ] )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wuid</td>
<td>A null-terminated string containing the WorkUnit IDentifier to locate.</td>
</tr>
<tr>
<td>online</td>
<td>Optional. A Boolean true/false value specifying whether the search is performed online. If omitted, the default is TRUE.</td>
</tr>
<tr>
<td>archived</td>
<td>Optional. A Boolean true/false value specifying whether the search is performed in the archives. If omitted, the default is FALSE.</td>
</tr>
</tbody>
</table>

**Return:** WorkunitExists returns a BOOLEAN value.

The **WorkunitExists** function returns whether the *wuid* exists.

**Example:**

```plaintext
OUTPUT(STD.System.Workunit.WorkunitExists('W20070308-164946'));
```
WorkunitList

STD.System.Workunit.WorkunitList ( lowwuid[, highwuid][, username][, cluster][, jobname]
[ , state ][ , priority ] [ , fileread ] [ , filewritten ] [ , roxiecluster ] [ , eclcontains ][ , online ][ ,
archived ][ , appvalues ] )

- **lowwuid**: A null-terminated string containing the lowest WorkUnit IDentifier to list. This may be an empty string.
- **highwuid**: Optional. A null-terminated string containing the highest WorkUnit IDentifier to list. If omitted, the default is an empty string.
- **cluster**: Optional. A null-terminated string containing the name of the cluster the workunit ran on. If omitted, the default is an empty string.
- **jobname**: Optional. A null-terminated string containing the name of the workunit. This may contain wildcard (? ?) characters. If omitted, the default is an empty string.
- **state**: Optional. A null-terminated string containing the state of the workunit. If omitted, the default is an empty string.
- **priority**: Optional. A null-terminated string containing the priority of the workunit. If omitted, the default is an empty string.
- **fileread**: Optional. A null-terminated string containing the name of a file read by the workunit. This may contain wildcard (? ?) characters. If omitted, the default is an empty string.
- **filewritten**: Optional. A null-terminated string containing the name of a file written by the workunit. This may contain wildcard (? ?) characters. If omitted, the default is an empty string.
- **roxiecluster**: Optional. A null-terminated string containing the name of the Roxie cluster. If omitted, the default is an empty string.
- **eclcontains**: Optional. A null-terminated string containing text to search for in the workunit's ECL code. This may contain wildcard (? ?) characters. If omitted, the default is an empty string.
- **online**: Optional. A Boolean true/false value specifying whether the search is performed online. If omitted, the default is TRUE.
- **archived**: Optional. A Boolean true/false value specifying whether the search is performed in the archives. If omitted, the default is FALSE.
- **appvalues**: Optional. A null-terminated string containing application values to search for. Use a string of the form appname/key=value or appname/*=value.

Return: WorkunitList returns a DATASET.

The **WorkunitList** function returns a dataset of all workunits that meet the search criteria specified by the parameters passed to the function. All the parameters are search values and all but the first are omittable, therefore the easiest way to pass a particular single search parameter would be to use the NAMED parameter passing technique.

The resulting DATASET is in this format:

```
WorkunitRecord := RECORD
    STRING24 wuid;
    STRING owner[MAXLENGTH(64)];
    STRING cluster[MAXLENGTH(64)];
    STRING roxiecluster[MAXLENGTH(64)];
    STRING job[MAXLENGTH(64)];
    STRING10 state;
    STRING7 priority;
    STRING20 created;
```
Example:

```c
OUTPUT(STD.System.Workunit.WorkunitList(''));
//list all current workunits
OUTPUT(STD.System.Workunit.WorkunitList(''
    NAMED eclcontains := 'COUNT')));
//list only those where the ECL code contains the word 'COUNT'
//this search is case insensitive and does include comments
STD.System.Workunit.SetWorkunitAppValue('MyApp', 'FirstName', 'Jim', TRUE);
OUTPUT(STD.System.Workunit.WorkunitList(appvalues := 'MyApp/FirstName=Jim'));
//returns a list of workunits with app values where FirstName='Jim'
```

See Also: SetWorkunitAppValue
SetWorkunitAppValue

STD.System.Workunit.SetWorkunitAppValue(app, key, value, [overwrite])

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>The application name to set.</td>
</tr>
<tr>
<td>key</td>
<td>The name of the value to set.</td>
</tr>
<tr>
<td>value</td>
<td>The value to set.</td>
</tr>
<tr>
<td>overwrite</td>
<td>A boolean TRUE or FALSE flag indicating whether to allow the value to overwrite an existing value. Default is TRUE.</td>
</tr>
</tbody>
</table>

Return: SetWorkunitAppValue returns TRUE if the value was set successfully.

The **SetWorkunitAppValue** function sets an application value in the current workunit. It returns TRUE if the value was set successfully.

Example:

```plaintext
IMPORT STD;
STD.System.Workunit.SetWorkunitAppValue('MyApp','FirstName','Jim',TRUE);
OUTPUT(STD.System.Workunit.WorkunitList(appvalues := 'MyApp/FirstName='Jim'));  //returns a list of workunits with app values where FirstName='Jim'
```

See Also: WorkunitList
WUIDonDate

STD.System.Workunit.WUIDonDate ( year, month, day, hour, minute )

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>year</td>
<td>An unsigned integer containing the year value.</td>
</tr>
<tr>
<td>month</td>
<td>An unsigned integer containing the month value.</td>
</tr>
<tr>
<td>day</td>
<td>An unsigned integer containing the day value.</td>
</tr>
<tr>
<td>hour</td>
<td>An unsigned integer containing the hour value.</td>
</tr>
<tr>
<td>minute</td>
<td>An unsigned integer containing the minute value.</td>
</tr>
</tbody>
</table>

Return: WUIDonDate returns a VARSTRING value.

The WUIDonDate function returns a valid WorkUnit IDentifier for a workunit that meets the passed parameters.

Example:

```plaintext
lowwuid := STD.System.Workunit.WUIDonDate(2008,02,13,13,00);
highwuid := STD.System.Workunit.WUIDonDate(2008,02,13,14,00);
OUTPUT(STD.System.Workunit.WorkunitList(lowwuid,highwuid));
//returns a list of workunits between 1 & 2 PM on 2/13/08
```
WUIDdaysAgo

STD.System.Workunit.WUIDdaysAgo ( daysago )

daysago  | An unsigned integer containing the number of days to go back.
Return:  | WUIDdaysAgo returns a VARSTRING value.

The WUIDdaysAgo function returns a valid WorkUnit IDentifier for a workunit that would have run within the last daysago days.

Example:

daysago := STD.System.Workunit.WUIDdaysAgo(3);
OUTPUT(STD.System.Workunit.WorkunitList(daysago));
  //returns a list of workunits run in the last 72 hours
WorkunitTimeStamps

STD.System.Workunit.WorkunitTimeStamps ( wuid )

<table>
<thead>
<tr>
<th>wuid</th>
<th>A null-terminated string containing the WorkUnit IDentifier.</th>
</tr>
</thead>
</table>

Return: WorkunitTimeStamps returns a DATASET value.

The WorkunitTimeStamps function returns a DATASET with this format:

```plaintext
EXPORT WsTimeStamp := RECORD
    STRING32 application;
    STRING16 id;
    STRING20 time;
    STRING16 instance;
END;
```

Each record in the returned dataset specifies a step in the workunit's execution process (creation, compilation, etc.).

Example:

```plaintext
OUTPUT(STD.System.Workunit.WorkunitTimeStamps('W20070308-164946'));
/* produces output like this:
'workunit     ','Created ','2008-02-13T18:28:20Z','              '
'workunit     ','Modified','2008-02-13T18:32:47Z','              '
'EclAgent     ','Started ','2008-02-13T18:32:35Z','training009003'
'Thor - graph1','Finished','2008-02-13T18:32:13Z','training009004'
'Thor - graph1','Started ','2008-02-13T18:32:13Z','training009004'
'EclAgent     ','Finished','2008-02-13T18:33:09Z','training009003'
*/
```
# WorkunitMessages

**STD.System.Workunit.WorkunitMessages** ( `wuid` )

<table>
<thead>
<tr>
<th><code>wuid</code></th>
<th>A null-terminated string containing the WorkUnit IDentifier.</th>
</tr>
</thead>
</table>

Return: WorkunitMessages returns a DATASET value.

The **WorkunitMessages** function returns a DATASET with this format:

```plaintext
EXPORT WsMessage := RECORD
    UNSIGNED4 severity;
    INTEGER4 code;
    STRING32 location;
    UNSIGNED4 row;
    UNSIGNED4 col;
    STRING16 source;
    STRING20 time;
    STRING message{MAXLENGTH(1024)};
END;
```

This function returns all messages in the workunit. Each record in the returned dataset specifies a message in the workunit.

The **severity** value can be 1 for Warning, 2 for Error, or 3 for Information.

Example:

```plaintext
IMPORT STD;
OUTPUT(STD.System.Workunit.WorkunitMessages('W20210602-164946'));
```
WorkunitFilesRead

STD.System.Workunit.WorkunitFilesRead ( wuid )

<table>
<thead>
<tr>
<th>wuid</th>
<th>A null-terminated string containing the WorkUnit IDentifier.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>WorkunitFilesRead returns a DATASET value.</td>
</tr>
</tbody>
</table>

The WorkunitFilesRead function returns a DATASET with this format:

```plaintext
EXPORT WsFileRead := RECORD
    STRING name{MAXLENGTH(256)};
    STRING cluster{MAXLENGTH(64)};
    BOOLEAN isSuper;
    UNSIGNED4 usage;
END;
```

Each record in the returned dataset specifies a file read by the workunit.

Example:

```plaintext
OUTPUT(STD.System.Workunit.WorkunitFilesRead('W20070308-164946'));
/* produces results that look like this
 'rttest::difftest::superfile','thor','true','1'
 'rttest::difftest::base1','thor','false','1'
 */
```
WorkunitFilesWritten

STD.System.Workunit.WorkunitFilesWritten ( wuid )

<table>
<thead>
<tr>
<th>wuid</th>
<th>A null-terminated string containing the WorkUnit IDentifier.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>WorkunitFilesWritten returns a DATASET value.</td>
</tr>
</tbody>
</table>

The **WorkunitFilesWritten** function returns a DATASET with this format:

```plaintext
EXPORT WsFileRead := RECORD
    STRING name{MAXLENGTH(256)};
    STRING10 graph;
    STRING cluster{MAXLENGTH(64)};
    UNSIGNED4 kind;
END;
```

Each record in the returned dataset specifies a file written by the workunit.

**Example:**

```plaintext
OUTPUT(STD.System.Workunit.WorkunitFilesWritten('W20070308-164946'));
/* produces results that look like this
 'rttest::testfetch','graph1','thor','0'
 */
```
## WorkunitTimings

**STD.System.Workunit.WorkunitTimings** ( `wuid` )

<table>
<thead>
<tr>
<th><code>wuid</code></th>
<th>A null-terminated string containing the WorkUnit IDentifier.</th>
</tr>
</thead>
</table>

Return: WorkunitTimings returns a DATASET value.

The **WorkunitTimings** function returns a DATASET with this format:

```
EXPORT WsTiming := RECORD
  UNSIGNED4 count;
  UNSIGNED4 duration;
  UNSIGNED4 max;
  STRING name{MAXLENGTH(64)};
END;
```

Each record in the returned dataset specifies a timing for the workunit.

Example:

```c
OUTPUT(STD.System.Workunit.WorkunitTimings('W20070308-164946'));
/* produces results that look like this
'1','4','4','EclServer: tree transform'
'1','0','0','EclServer: tree transform: normalize.scope'
'1','1','1','EclServer: tree transform: normalize.initial'
'1','18','18','EclServer: write c++'
'1','40','40','EclServer: generate code'
'1','1010','1010','EclServer: compile code'
'1','33288','33288','Graph graph1 - 1 (1)'
'1','33629','33629','Total thor time: '
'2','1','698000','WorkUnit_lockRemote'
'1','2','2679000','SDS.Initialize'
'1','0','439000','Environment.Initialize'
'1','33775','3710788928','Process'
'1','1','1942000','WorkUnit_unlockRemote'
*/
BLAS Support

This section provides support for Basic Linear Algebra Subprogram support.

The BLAS functions use the column major mapping for the storage of a matrix. This is the mapping used in Fortran, and has the entries of the first column followed by the entries of the second column. This is the transpose of the row major form commonly used in the C language where the entries of the first row are followed by the entries of the second row.
### Types

**STD.BLAS.Types**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value_t</td>
<td>REAL8</td>
</tr>
<tr>
<td>dimension_t</td>
<td>UNSIGNED4</td>
</tr>
<tr>
<td>matrix_t</td>
<td>SET OF REAL8</td>
</tr>
<tr>
<td>Triangle</td>
<td>ENUM(UNSIGNED1, Upper=1, Lower=2)</td>
</tr>
<tr>
<td>Diagonal</td>
<td>ENUM(UNSIGNED1, UnitTri=1, NotUnitTri=2)</td>
</tr>
<tr>
<td>Side</td>
<td>ENUM(UNSIGNED1, Ax=1, xA=2)</td>
</tr>
</tbody>
</table>

Types for the Block Basic Linear Algebra Sub-programs support
ICellFunc

STD.BLAS.ICellFunc( v, r, c );

<table>
<thead>
<tr>
<th>v</th>
<th>The value</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>The row ordinal</td>
</tr>
<tr>
<td>c</td>
<td>The column ordinal</td>
</tr>
<tr>
<td>Return</td>
<td>The updated value</td>
</tr>
</tbody>
</table>

ICellFunc is the function prototype for Apply2Cells.

Example:

```plaintext
IMPORT STD;
REAL8 my_func(STD.BLAS.Types.value_t v, STD.BLAS.Types.dimension_t x, STD.BLAS.Types.dimension_t y) := 1/v; //set element to the reciprocal value
```

See Also: Apply2Cells
Apply2Cells

STD.BLAS.Apply2Cells( m, n, x, f);

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Number of rows</td>
</tr>
<tr>
<td>n</td>
<td>Number of columns</td>
</tr>
<tr>
<td>x</td>
<td>Matrix</td>
</tr>
<tr>
<td>f</td>
<td>Function to apply</td>
</tr>
</tbody>
</table>

Return: The updated matrix

The **Apply2Cells** function iterates a matrix and applies a function to each cell.

Example:

```
IMPORT STD;
STD.LA.Types.value_t example_1(STD.LA.Types.value_t v, 
    STD.LA.Types.dimension_t x, 
    STD.LA.Types.dimension_t y) := FUNCTION 
    RETURN IF(x=y, 1.0, 1/v);
END;

init_mat := [1, 2, 4, 4, 5, 10, 2, 5, 2];
new_mat := STD.BLAS.Apply2Cells(3, 3, init_mat, example_1);

// The new_mat matrix will be [1, .5, .25, .25, 1, .1, .5, .2, 1]
```

See Also: ICellFunc
**dasum**

**STD.BLAS.dasum( m, x, incx, skipped);**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Number of entries</td>
</tr>
<tr>
<td>x</td>
<td>The column major matrix holding the vector</td>
</tr>
<tr>
<td>incxx</td>
<td>The increment for x, 1 in the case of an actual vector</td>
</tr>
<tr>
<td>skipped</td>
<td>The number of entries stepped over. Default is zero.</td>
</tr>
</tbody>
</table>

**Return:** The sum of the absolute values

The **dasum** function gets the absolute sum, the 1 norm of a vector.

**Example:**

```plaintext
IMPORT STD;
STD.BLAS.Types.matrix_t test_data := [2, -2, -3, 3, 1, 3, -1, -1, 1];
STD.BLAS.dasum(9, test_data, 1); //sums the absolute values of the matrix, and returns 17
```
**daxpy**

STD.BLAS.daxpy( \(N, \alpha, X, incX, Y, incY, x\_skipped, y\_skipped\));

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(N)</td>
<td>Number of entries</td>
</tr>
<tr>
<td>(\alpha)</td>
<td>The column major matrix holding the vector</td>
</tr>
<tr>
<td>(X)</td>
<td>The column major matrix holding the vector X</td>
</tr>
<tr>
<td>(incX)</td>
<td>The increment for (x), 1 in the case of an actual vector</td>
</tr>
<tr>
<td>(Y)</td>
<td>The column major matrix holding the vector Y</td>
</tr>
<tr>
<td>(incY)</td>
<td>The increment or stride of (Y)</td>
</tr>
<tr>
<td>(x_skipped)</td>
<td>The number of entries stepped over. to get to the first (X).</td>
</tr>
<tr>
<td>(y_skipped)</td>
<td>The number of entries stepped over. to get to the first (Y).</td>
</tr>
</tbody>
</table>

Return: The updated matrix

The **daxpy** function is used to sum two vectors or matrices with a scalar multiplier applied during the sum operation.

Example:

```plaintext
IMPORT STD;
STD.BLAS.Types.t_matrix term_1 := [1, 2, 3];
STD.BLAS.Types.t_matrix term_2 := [3, 2, 1].
STD.BLAS.daxpy(3, 2, term_1, 1, term_2, 1); // result is [5, 6, 7]
```
The `dgemm` function is used to multiply two matrices and optionally add that product to another matrix.

**Example:**

```plaintext
IMPORT STD;
STD.BLAS.Types.t_matrix term_a := [2, 4, 8];
STD.BLAS.Types.t_matrix term_b := [2, 1, 1];
STD.BLAS.dgemm(TRUE, FALSE, 3, 3, 1, 1, term_a, term_b);
// the outer product of the term_a and term_b vectors
// result is [4, 8, 16, 2, 4, 8, 2, 4, 8]
```
dgetf2

STD.BLAS.dgetf2( m, n, a);

<table>
<thead>
<tr>
<th>m</th>
<th>Number of rows of matrix a</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Number of columns of matrix a</td>
</tr>
<tr>
<td>a</td>
<td>Matrix a</td>
</tr>
</tbody>
</table>

Return: Composite matrix of factors, lower triangle has an implied diagonal of ones. Upper triangle has the diagonal of the composite.

The `dgetf2` function produces a combine lower and upper triangular factorization.

Example:

```hpcc
IMPORT STD;
STD.BLAS.Types.t_matrix test := [2,4,6,3,10,25, 9,34,100];
STD.BLAS.dgetf2(3, 3, test); //result is [2,2,3,3,4,4,9,16,25];
```
dpotf2

STD.BLAS.dpotf2( tri, r, A, clear);

<table>
<thead>
<tr>
<th>tri</th>
<th>Indicates whether upper or lower triangle is used</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Number of rows/columns in the square matrix</td>
</tr>
<tr>
<td>A</td>
<td>The square matrix A</td>
</tr>
<tr>
<td>clear</td>
<td>Clears the unused triangle</td>
</tr>
</tbody>
</table>

Return: The triangular matrix requested

The dpotf2 function computes the Cholesky factorization of a real symmetric positive definite matrix A. The factorization has the form $A = U^T U$ if the tri parameter is Triangle.Upper, or $A = L L^T$ if the tri parameter is Triangle.Lower. This is the unblocked version of the algorithm, calling Level 2 BLAS.

Example:

```plaintext
IMPORT STD;
STD.BLAS.Types.matrix_t  symmetric_pos_def := [4, 6, 8, 6, 13, 18, 8, 18, 29];
Lower_Triangle := BLAS.dpotf2(STD.BLAS.Types.Triangle.lower, 3, symmetric_pos_def);
```
**dscal**

STD.BLAS.dscal( $N$, $alpha$, $X$, $incX$, $skipped$);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N$</td>
<td>Number of elements in the vector</td>
</tr>
<tr>
<td>$alpha$</td>
<td>The scaling factor</td>
</tr>
<tr>
<td>$X$</td>
<td>The column major matrix holding the vector</td>
</tr>
<tr>
<td>$incX$</td>
<td>The stride to get to the next element in the vector</td>
</tr>
<tr>
<td>$skipped$</td>
<td>The number of elements skipped to get to the first element</td>
</tr>
</tbody>
</table>

Return: The updated matrix

The **dscal** function scales a vector $alpha$.

Example:

```
IMPORT STD;
STD.BLAS.Types.matrix_t test := [1, 1, 1, 2, 2, 2, 3, 3, 3];
result := STD.BLAS.dscal(9, 2.0, test, 1); // multiply each element by 2
```
## dsyrk

STD.BLAS.dsyrk( tri, transposeA, N, K, alpha, A, beta, C, clear);

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tri</td>
<td>Indicates whether upper or lower triangle is used</td>
</tr>
<tr>
<td>transposeA</td>
<td>Transpose the A matrix to be NxK</td>
</tr>
<tr>
<td>N</td>
<td>Number of rows</td>
</tr>
<tr>
<td>K</td>
<td>Number of columns in the update matrix or transpose</td>
</tr>
<tr>
<td>alpha</td>
<td>The alpha scalar</td>
</tr>
<tr>
<td>A</td>
<td>The update matrix, either NxK or KxN</td>
</tr>
<tr>
<td>beta</td>
<td>The beta scalar</td>
</tr>
<tr>
<td>C</td>
<td>The matrix to update</td>
</tr>
<tr>
<td>clear</td>
<td>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.</td>
</tr>
</tbody>
</table>

Return: The updated matrix

The **dsyrk** function implements a symmetric rank update $C \leftarrow \alpha A^{**T} A + \beta C$ or $c \leftarrow \alpha A A^{**T} + \beta C$. $C$ is $N \times N$.

Example:

```plaintext
IMPORT STD;
STD.BLAS.Types.matrix_t initC := [1, 1, 1, 2, 2, 2, 3, 3, 3];
STD.BLAS.Types.matrix_t initA := [1, 1, 1];
Test1_mat := STD.BLAS.dsyrk(STD.BLAS.Types.Triangle.upper, FALSE, 3, 1, 1, initA, 1, initC, TRUE)
```
The `dtrsm` function is a triangular matrix solver. \( \text{op}(A) \ X = \alpha \ B \) or \( X \ \text{op}(A) = \alpha \ B^* \) where \( \text{op} \) is Transpose, \( X \) and \( B \) is \( M \times N \).

Example:

```plaintext
IMPORT STD;
Side := STD.BLAS.Types.Side;
Diagonal := STD.BLAS.Types.Diagonal;
Triangle := STD.BLAS.Types.Triangle;
STD.BLAS.Types.matrix_t left_a0 := [2, 3, 4, 0, 2, 3, 0, 0, 2];
STD.BLAS.Types.matrix_t mat_b := [4, 6, 8, 6, 13, 18, 8, 18, 29];

Test1_mat := STD.BLAS.dtrsm(Side.Ax, Triangle.Lower, FALSE, Diagonal.NotUnitTri, 3, 3, 1.0, left_a0, mat_b);
```
extract_diag

STD.BLAS.extract_diag ( m.n.x);

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Number of rows</td>
</tr>
<tr>
<td>n</td>
<td>Number of columns</td>
</tr>
<tr>
<td>x</td>
<td>The matrix from which to extract the diagonal</td>
</tr>
</tbody>
</table>

Return: Diagonal matrix

The `extract_diag` function extracts the diagonal of the matrix.

Example:

```haskell
IMPORT STD;
STD.BLAS.Types.matrix_t x := [1.0, 2.0, 3.0, 2.0, 2.0, 2.0, 4.0, 4.0, 4.0];
diagonal_only := := STD.BLAS.extract_diag(3, 3, x);
```
**extract_tri**

STD.BLAS.extract_tri ( m, n, tri, dt, a );

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Number of rows</td>
</tr>
<tr>
<td>n</td>
<td>Number of columns</td>
</tr>
<tr>
<td>tri</td>
<td>Indicates whether upper or lower triangle is used</td>
</tr>
<tr>
<td>dt</td>
<td>Use Diagonal.NotUnitTri or Diagonal.UnitTri</td>
</tr>
<tr>
<td>a</td>
<td>The matrix, usually a composite from factoring</td>
</tr>
</tbody>
</table>

Return: Triangle

The `extract_tri` function extracts the upper or lower triangle. The diagonal can be the actual or implied unit diagonal.

Example:

```plaintext
IMPORT STD;
Diagonal := STD.BLAS.Types.Diagonal;
Triangle := STD.BLAS.Types.Triangle;
STD.BLAS.Types.matrix_t x := [1.0, 2.0, 3.0, 2.0, 2.0, 2.0, 4.0, 4.0, 4.0];
triangle := STD.BLAS.extract_tri(3, 3, Triangle.upper, Diagonal.NotUnitTri, x);
```
make_diag

STD.BLAS.make_diag ( m, v, X );

<table>
<thead>
<tr>
<th>m</th>
<th>Number of diagonal entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>Option value, default is 1</td>
</tr>
<tr>
<td>X</td>
<td>Optional input of diagonal values, multiplied by v</td>
</tr>
<tr>
<td>Return:</td>
<td>A diagonal matrix</td>
</tr>
</tbody>
</table>

The `make_diag` function generates a diagonal matrix.

Example:

```plaintext
IMPORT STD;
STD.BLAS.Types.matrix_t init1 := [1.0, 2.0, 3.0, 4.0];
Square := STD.BLAS.make_diag(4, 1, init1); // 4x4 with diagonal 1, 2, 3, 4
```
**make_vector**

STD.BLAS.make_vector ( \( m, v \) );

<table>
<thead>
<tr>
<th>( m )</th>
<th>Number of elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v )</td>
<td>The values, default is 1</td>
</tr>
</tbody>
</table>

**Return:** The vector

The `make_vector` function generates a vector of dimension \( n \)

Example:

```plaintext
IMPORT STD;
twos_vector := STD.BLAS.make_vector(4, 2); // a vector of [2, 2, 2, 2]
```
trace

STD.BLAS.trace ( m, n, x );

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Number of rows</td>
</tr>
<tr>
<td>n</td>
<td>Number of columns</td>
</tr>
<tr>
<td>x</td>
<td>The matrix</td>
</tr>
<tr>
<td>Return:</td>
<td>The trace (sum of the diagonal entries)</td>
</tr>
</tbody>
</table>

The trace function computes the trace of the input matrix

Example:

```
IMPORT STD;
STD.BLAS.Types.matrix_t x := [1.0, 2.0, 3.0, 2.0, 2.0, 2.0, 4.0, 4.0, 4.0];
trace_of_x := STD.BLAS.trace(3,3,x); // the trace is 7
```
This section covers the common math functions in the Standard Library.
**Infinity**

STD.Math.Infinity;

| Return: | Returns a REAL "infinity" value. |

**Infinity** returns an "infinity" value.

Example:

```pascal
IMPORT STD;
myValue := STD.Math.Infinity;
myValue;
```

See Also: isInfinite
NaN

STD.Math.NaN;

<table>
<thead>
<tr>
<th>Return:</th>
<th>Returns a non-signalling NaN (Not a Number) value.</th>
</tr>
</thead>
</table>

The NaN function returns a non-signalling NaN (Not a Number) value.

Example:

```pascal
IMPORT STD;
myValue := STD.Math.NaN;
myValue;
```

See Also: IsNan
isInfinite

STD.Math.isInfinite( val);

<table>
<thead>
<tr>
<th>val</th>
<th>The value to test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Returns a BOOLEAN indicating whether a real value is infinite (positive or negative).</td>
</tr>
</tbody>
</table>

The isInfinite function returns whether a real value is infinite (positive or negative).

Example:

```plaintext
IMPORT STD;
a := STD.Math.Infinity;
b := 42.1;
STD.Math.isInfinite(a); //true
STD.Math.isInfinite(b); //false
```

See Also: Infinity, isFinite
isNaN

STD.Math.isNaN(val);

<table>
<thead>
<tr>
<th>val</th>
<th>The value to test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Returns a BOOLEAN indicating whether a real value is a NaN (not a number) value.</td>
</tr>
</tbody>
</table>

The **isNaN** function returns whether a real value is a NaN (not a number) value.

**Example:**

```plaintext
IMPORT STD;
a := STD.Math.NaN;
b := 42.1;
STD.Math.isNaN(a); //true
STD.Math.isNaN(b); //false
```

See Also: NaN, isFinite
**isFinite**

STD.Math.isFinite( val);

<table>
<thead>
<tr>
<th>val</th>
<th>The value to test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>Returns a BOOLEAN indicating whether a real value is a valid value (neither infinite nor NaN).</td>
</tr>
</tbody>
</table>

The **isFinite** function returns whether a real value is a valid value (neither infinite nor NaN).

Example:

```plaintext
IMPORT STD;
a := STD.Math.Infinity;
b := STD.Math.NaN;
c := 42.1;
STD.Math.isFinite(a); //false
STD.Math.isFinite(b); //false
STD.Math.isFinite(c); //true
```

See Also: isNaN, isInfinite
## FMod

`STD.Math.FMod( numer, denom);`

<table>
<thead>
<tr>
<th>numer</th>
<th>The numerator</th>
</tr>
</thead>
<tbody>
<tr>
<td>denom</td>
<td>The denominator</td>
</tr>
<tr>
<td>Return:</td>
<td>Returns the floating-point remainder of numer/denom (rounded towards zero).</td>
</tr>
</tbody>
</table>

The **FMod** function returns the floating-point remainder of numer/denom (rounded towards zero).

If denom is zero, the result depends on the divideByZero flag:

- If set to 'zero' or unset: returns zero.
- If set to 'nan': returns a non-signalling NaN value.
- If set to 'fail': throws an exception.

Example:

```plaintext
#OPTION ('divideByZero', 'nan'); // divide by zero creates a quiet NaN
IMPORT STD;
STD.Math.FMod(5.1, 3.0); // 2.1
STD.Math.FMod(-5.1, 3.0); // -2.1
STD.Math.FMod(5.1, 0); // NaN
```
FMatch

STD.Math.FMatch( a, b, epsilon);

<table>
<thead>
<tr>
<th>a</th>
<th>The first value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>The second value.</td>
</tr>
<tr>
<td>epsilon</td>
<td>The allowable margin of error.</td>
</tr>
</tbody>
</table>

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

The FMatch function returns whether two floating point values are the same, within margin of error epsilon.

Example:

```plaintext
IMPORT STD;
STD.Math.FMatch(2.6, 2.2, 0.5); //true
STD.Math.FMatch(2.6, 2.2, 0.3); //false
```