Standard Library Reference

Boca Raton Documentation Team

Standard Library Reference

Boca Raton Documentation Team

We welcome your comments and feedback about this document via email to <docfeedback@hpccsystems.com>

Please include **Documentation Feedback** in the subject line and reference the document name, page numbers, and current Version Number in the text of the message.

2026 Version 10.4.20-1

Logical Files 9
CompareFiles 10
DeleteLogicalFile 11
LogicalFileList 12
LogicalFileListFiltered 13
GetNoCommonDefault 15
FileExists 16
ForeignLogicalFileName 17
GetFileDescription 18
GetLogicalFileAttribute 19
ProtectLogicalFile 20
RenameLogicalFile 21
SetFileDescription 22
SetReadOnly 23
VerifyFile 24
SuperFiles 25
CreateSuperFile 26
SuperFileExists 27
DeleteSuperFile 28
GetSuperFileSubCount 29
GetSuperFileSubName 30
LogicalFileSuperOwners 31
LogicalFileSuperSubList 32
SuperFileContents 33
FindSuperFileSubName 34
StartSuperFileTransaction 35
AddSuperFile 36
RemoveSuperFile 37
ClearSuperFile 38
RemoveOwnedSubFiles 39
SwapSuperFile 40
ReplaceSuperFile 41
PromoteSuperFileList 42
FinishSuperFileTransaction 43
External Files 44
ExternalLogicalFileName 45
PlaneLogicalFileName 46
MoveExternalFile 47
DeleteExternalFile 48
CreateExternalDirectory 49
RemoteDirectory 50
File Browsing 51
SetColumnMapping 52
GetColumnMapping 54
AddFileRelationship 55
FileRelationshipList 57
RemoveFileRelationship 58
File Movement 59
DfuPlusExec 60
AbortDfuWorkunit 61
Copy 62
DeSpray 64
RemotePull 65
Replicate 67

SprayFixed 68
SprayDelimited / SprayVariable 70
SprayXML 72
SprayJson 74
WaitDfuWorkunit 76
SetExpireDays 77
GetExpireDays 78
ClearExpireDays 79
String Handling 80
CleanAccents 81
CleanSpaces 82
CommonPrefix 83
CommonSuffix 84
CompareAtStrength 85
CompareIgnoreCase 86
Contains 87
CountWords 88
DecodeBase64 89
EditDistance 90
EditDistanceWithinRadius 91
EncodeBase64 92
EndsWith 93
EqualIgnoreCase 94
ExcludeFirstWord 95
ExcludeLastWord 96
ExcludeNthWord 97
Extract 98
ExtractMultiple 99
Filter 100
FilterOut 101
Find 102
FindCount 103
FindAtStrength 104
FindAtStrengthReplace 105
FindReplace 106
FindWord 107
FromHexPairs 108
GetNthWord 109
RemoveSuffix 110
Repeat 111
Reverse 112
SplitWords 113
SubstituteExcluded 114
SubstituteIncluded 115
StartsWith 116
ToHexPairs 117
ToLowerCase 118
ToTitleCase 119
ToUpperCase 120
Translate 121
Version 122
WildMatch 123
WordCount 124
Metaphone Support 125

Primary 126
Secondary 127
Double 128
Cryptography Support 129
Cryptographic Library Overview 130
SupportedHashAlgorithms 131
SupportedSymmetricCipherAlgorithms 132
SupportedPublicKeyAlgorithms 133
Hashing Module 134
Hash 135
SymmetricEncryption Module 136
Encrypt (Symmetric) 137
Decrypt (Symmetric) 138
PublicKeyEncryption Module 139
Encrypt (PKE) 140
Decrypt (PKE) 141
Sign (PKE) 142
VerifySignature (PKE) 143
PublicKeyEncryptionFromBuffer Module 144
Encrypt (PKE From Buffer) 146
Decrypt (PKE From Buffer) 147
Sign (PKE From Buffer) 148
VerifySignature (PKE From Buffer) 149
PublicKeyEncryptionFromLFN Module 150
Encrypt (PKE From LFN) 152
Decrypt (PKE From LFN) 154
Sign (PKE From LFN) 156
VerifySignature (PKE From LFN) 158
Date and Time Handling 160
Date Data Types 161
Time Data Types 162
Year 163
Month 164
Day 165
Hour 166
Minute 167
Second 168
DateFromParts 169
TimeFromParts 170
IsLeapYear 171
IsDateLeapYear 172
IsValidDate 173
IsValidTime 174
IsValidGregorianDate 175
FromGregorianYMD 176
ToGregorianYMD 177
FromStringToDate 178
Today 179
CurrentDate 180
CurrentTime 181
DayOfWeek 182
DayOfYear 183
DaysBetween 184
MonthsBetween 185

AdjustDate 186
AdjustCalendar 187
MonthWeekNumFromDate 188
YearWeekNumFromDate 189
TimestampToString 190
UniqueTZAbbreviations 191
UniqueTZLocations 192
TZDataForLocation 193
FindTZData 194
SecondsBetweenTZ 195
AdjustTimeTZ 196
ToLocalTime 197
ToUTCTime 198
AppendTZOffset 199
AppendTZAdjustedTime 201
ISODayOfWeekFromDate 203
ISOIsLongYear 204
ISOWeeksFromDate 205
ISORawWeekNumForDate 206
ISOWeekNumWeekDayAndYearFromDate 207
ISOWeekDate 208
Cluster Handling 209
Node 210
Nodes 211
LogicalToPhysical 212
DaliServer 213
Group 214
GetExpandLogicalName 215
Job Handling 216
WUID 217
Target 218
Name 219
User 220
OS 221
Platform 222
LogString 223
File Monitoring 224
MonitorFile 225
MonitorLogicalFileName 227
Logging 229
dbglog 230
addWorkunitInformation 231
addWorkunitWarning 232
addWorkunitError 233
getGlobalId 234
getLocalId 235
generateGloballyUniqueID 236
getElapsedMs 237
Auditing 238
Audit 239
Utilities 240
GetHostName 241
ResolveHostName 242
GetUniqueInteger 243

GetEspUrl 244
PlatformVersionCheck 245
Debugging 246
GetParseTree 247
GetXMLParseTree 248
Sleep 249
msTick 250
Email 251
SendEmail 252
SendEmailAttachData 253
SendEmailAttachText 254
Workunit Services 255
WorkunitExists 256
WorkunitList 257
SetWorkunitAppValue 259
WUIDonDate 260
WUIDdaysAgo 261
WorkunitTimeStamps 262
WorkunitMessages 263
WorkunitFilesRead 264
WorkunitFilesWritten 265
WorkunitTimings 266
WorkunitStatistics 267
BLAS Support 269
Types 270
ICellFunc 271
Apply2Cells 272
dasum 273
daxpy 274
dgemm 275
dgetf2 276
dpotf2 277
dscal 278
dsyrk 279
dtrsm 280
extract_diag 281
extract_tri 282
make_diag 283
make_vector 284
trace 285
Math Support 286
Infinity 287
NaN 288
isInfinite 289
isNaN 290
isFinite 291
FMod 292
FMatch 293
Key/Value Store Support 294
Key/Value Store Overview 295
Store Module 296
CreateStore 297
ListStores 298
ListNamespaces 299

WithNamespace Module 300
SetKeyValue 301
GetKeyValue 302
DeleteKeyValue 303
GetAllKeys 304
GetAllKeyValues 305
DeleteNamespace 306
Record Definitions 307

Logical Files

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 returns an INTEGER4 value.

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

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

Example:

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

DeleteLogicalFile

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

filename	A null-terminated string containing the logical name of the file.
ifexists	Optional. A boolean value indicating whether to post an error if the *filename* does not exist. If omitted, the default is FALSE.

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

Example:

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

LogicalFileList

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

pattern	Optional. A null-terminated string containing the mask of the files to list. If omitted,the default is '*' (all files).
includenormal	Optional. A boolean flag indicating whether to include "normal" files. If omitted, the default is TRUE.
includesuper	Optional. A boolean flag indicating whether to include SuperFiles. If omitted, the default is FALSE.
unknownszero	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.
foreigndali	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.
Return:	LogicalFileList returns a dataset in the following format:

Note: LogicalFileList is deprecated. For enhanced filtering capabilities and additional file information, use LogicalFileListFiltered instead. However, LogicalFileListFiltered cannot be used with remote environments and a foreign Dali is not supported. If you need that functionality, use STD.File.LogicalFileList and supply the foreign Dali IP.

```
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
```

See Also: LogicalFileListFiltered

LogicalFileListFiltered

STD.File.LogicalFileListFiltered([*namepattern* **] [,** *filters* **] [,** *fields* **] [,** *unknownszero* **] [,** *remoteDfs* **] [,** *maxFileLimit* **])**

namepattern	Optional. A null-terminated string containing the mask pattern of the files to list. If omitted, the default is '*' (all files).
filters	Optional. A comma-separated string of filter expressions for advanced filtering. Examples include 'size>100000000' for files larger than 100MB, 'owner:username' to filter by owner, 'is:superfile' to include only SuperFiles, 'is:normal' for normal files, 'has:description' for files with descriptions, and 'modified>=2026-01-01' for files modified on or after a specific date. If omitted, the default is '' (no filtering).
fields	Optional. A comma-separated string specifying which fields to return in the result set. For any non-empty value, the filename field ('name') is always included even if you do not request it explicitly. Examples include 'name' for just the filename, 'size' for filename and size, or 'superfile,owner' for filename, superfile, and owner. If omitted or empty, the default is the legacy field subset 'name,superfile,size,rowcount,modified,owner,cluster'. Request any additional fields you want explicitly.
unknownszero	Optional. A boolean flag indicating whether to set unknown numeric result values, including file sizes and row counts, to zero (0) instead of minus-one (-1). If omitted, the default is FALSE.
remoteDfs	Optional. The name of the remote DFS service used to resolve the file. If blank then the file is resolved locally. If omitted, the default is blank. (Not yet supported; specifying this will raise an error.)
maxFileLimit	Optional. Maximum number of files to return. Set to -1 to use server default limit (100,000). Maximum client-side limit is 1,000,000. If omitted, the default is -1.
Return:	LogicalFileListFiltered returns a record containing count, limitBreached, and a dataset of file information in the format shown below.

The **LogicalFileListFiltered** function returns a record containing the number of files returned, a flag indicating whether the result was limited by maxFileLimit, and a dataset of logical files in the environment with enhanced filtering and detailed file information.

Record formats:

```
EXPORT FsLogicalFileInfoRecordEx := RECORD(FsLogicalFileInfoRecord)
  VARSTRING description;
  VARSTRING workunit;
  VARSTRING job;
  VARSTRING directory;
  BOOLEAN compressed;
  INTEGER4 recordsize;
  VARSTRING kind;
  INTEGER8 compressedsize;
  BOOLEAN persistent;
  VARSTRING protect;
  STRING19 accessed;
  INTEGER4 expiredays;
  REAL8 readcost;
  REAL8 writecost;
END;

EXPORT FsLogicalFileListResult := RECORD
  UNSIGNED4 count;
```

```
  BOOLEAN limitBreached;
  DATASET(FsLogicalFileInfoRecordEx) files;
END;
```

Example:

```
IMPORT STD;

// List all files
result1 := STD.File.LogicalFileListFiltered();
OUTPUT(result1.count);
OUTPUT(result1.files);

// List with limit
result2 := STD.File.LogicalFileListFiltered(maxFileLimit := 100);
OUTPUT(result2.limitBreached);

// List files larger than 100MB
result3 := STD.File.LogicalFileListFiltered(filters := 'size>100000000');

// List superfiles owned by userXYZ
result4 := STD.File.LogicalFileListFiltered(filters := 'owner:userXYZ,is:superfile');

// List normal files without description, modified on or after 01 Jan 2026
result5 := STD.File.LogicalFileListFiltered(filters := 'is:normal,!has:description,modified>=2026-01-01');

// List only specific fields
result6 := STD.File.LogicalFileListFiltered(fields := 'name,size,modified,owner');
```

See Also: LogicalFileList

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* **])**

filename	A null-terminated string containing the logical name of the file.
physicalcheck	Optional. A boolean TRUE/FALSE to indicate whether to check for the physical existence the *filename* on disk. If omitted, the default is FALSE.
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

STD.File.ForeignLogicalFileName(*filename* **[,** *foreigndali* **] [,** *absolutepath* **] [,** *omitClusterPrefix* **])**

filename	A null-terminated string containing the logical name of the file.
foreigndali	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.
absolutepath	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.
omitCluster-Preifx	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.
Return:	ForeignLogicalFileName returns 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:

```
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* **)**

filename	A null-terminated string containing the logical name of the file.
Return:	GetFileDescription returns a VARSTRING (null-terminated) value.

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* **)**

logicalfilename	A null-terminated string containing the name of the logical file as it is known by the DFU.
attrname	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.
Return:	GetLogicalFileAttribute returns returns a VARSTRING (null-terminated) value.

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

Note: If the *logicalfilename* argument references a file that does not exist, the workunit will fail with a runtime error.

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* **])**

logicalfilename	A null-terminated string containing the name of the logical file as it is known by the DFU.
value	Optional. A boolean flag indicating whether to protect or un-protect the file. If omitted, the default is TRUE.

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

Example:

```
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] **)**

filename	A null-terminated string containing the current logical name of the file.
newname	A null-terminated string containing the new logical name for the file.
allowOverwrite	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.

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

Example:

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

SetFileDescription

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

filename	A null-terminated string containing the logical name of the file.
value	A null-terminated string containing the description to place on the file.

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:

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

SetReadOnly

STD.File.SetReadOnly(*filename* **,** *flag* **)**

filename	A null-terminated string containing the logical name of the file.
flag	A boolean value indicating which way to set the read-only attribute of the *filename*.

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

Example:

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

VerifyFile

STD.File.VerifyFile(*file, usecrcs* **)**

file	A null-terminated string containing the logical name of the file.
usecrcs	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.
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:

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

Example:

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

SuperFiles

CreateSuperFile

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

superfile	A null-terminated string containing the logical name of the superfile.
sequentialparts	Optional. A boolean value indicating whether the sub-files must be sequentially ordered. If omitted, the default is FALSE.
allowExist	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.
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 subfiles 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:

```
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* **)**

filename	A null-terminated string containing the logical name of the superfile.
Return:	SuperFileExists returns a BOOLEAN value.

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:

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

See Also: FileExists

DeleteSuperFile

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

superName	A null-terminated string containing the logical name of the superfile.
deletesub	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.**
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* **)**

superfile	A null-terminated string containing the logical name of the superfile.
Return:	GetSuperFileSubCount returns an INTEGER4 value.

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

This function is not included in a superfile transaction.

Example:

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

GetSuperFileSubName

STD.File.GetSuperFileSubName(*superfile, subfile* **[,** *absolutepath* **])**

superfile	A null-terminated string containing the logical name of the superfile.
subfile	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.
absolutepath	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.
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:

```
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 withinthe 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* **)**

superfile	A null-terminated string containing the logical name of the superfile.
subfile	A null-terminated string containing the logical name of the sub-file.
Return:	FindSuperFileSubName returns an INTEGER4 value.

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'
```

StartSuperFileTransaction

STD.File.StartSuperFileTransaction( )

Return: Null.

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:

```
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* **])**

superfile	A null-terminated string containing the logical name of the superfile.
subfile	A null-terminated string containing the logical name of the sub-file. This may be another superfile.
atpos	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*.
addcontents	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).
strict	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.
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:

```
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

STD.File.RemoveSuperFile(*superfile, subfile* **[** , *del* **] [** , *removecontents* **])**

superfile	A null-terminated string containing the logical name of the superfile.
subfile	A null-terminated string containing the logical name of the sub-file. This may be another superfile or a foreign file or superfile.
del	A boolean flag specifying whether to delete the *subfile* from disk or just remove it from the *superfile* list of files. If omitted, the default is to just remove it from the *superfile* list of files. **This option should not be used if the subfile is a foreign file or foreign superfile.**
removecontents	A boolean flag specifying whether the contents of a *subfile* that is itself a superfile are recursively removed.
Return:	Null.

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

This function may be included in a superfile transaction.

Example:

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

ClearSuperFile

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

superfile	A null-terminated string containing the logical name of the superfile.
delete	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.
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:

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

RemoveOwnedSubFiles

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

superfile	A null-terminated string containing the logical name of the superfile.
delete	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.
Return:	Null.

The **RemoveOwnedSubFiles** function removes all owned sub-files from the specified superfile. These are only removed if they are soley 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:

```
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:

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

ReplaceSuperFile

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

superfile	A null-terminated string containing the logical name of the superfile.
subfile1	A null-terminated string containing the logical name of the sub-file. This may be another superfile.
subfile2	A null-terminated string containing the logical name of the sub-file. This may be another superfile.
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:

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

PromoteSuperFileList

STD.File.PromoteSuperFileList(*supernames* **[** , *addhead* **]** **[** , *deltail* **]** **[** , *createjustone* **]** **[** , *reverse* **]** **)**

oldlist **:= STD.File.fPromoteSuperFileList(** *supernames* **[** , *addhead* **]** **[** , *deltail* **]** **[** , *createjustone* **]** **[** , *reverse* **]** **);**

supernames	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).
addhead	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.
deltail	Optional. A boolean value specifying whether to physically delete the contents moved out of the last superfile. If omitted, the default is FALSE.
createjustone	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.
reverse	Optional. A boolean value specifying whether to reverse the order of processing the *supernames* list, effectively "demoting" instead of "promoting" the sub-files. If omitted, the default is FALSE.
oldlist	The name of the attribute that receives the returned string containing the list of the previous subfile contents of the emptied superfile.
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* **])**

rollback	Optional. A boolean flag that indicates whether to commit (FALSE) or roll back (TRUE) the transaction. If omitted, the default is FALSE.
Return:	Null.

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:

```
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* **)**

machineIP	A null-terminated string containing the IP address of the remote machine.
filename	A null-terminated string containing the path/name of the file.
Return:	ExternalLogicalFileName returns 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.

For storage-plane based path mapping, see STD.File.PlaneLogicalFileName.

Example:

```
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
```

PlaneLogicalFileName

STD.File.PlaneLogicalFileName(*planeName, path, abspath=TRUE* **)**

planeName	A null-terminated string containing the storage plane name. The plane must exist and currently must be an *lz* category plane.
path	A null-terminated string containing a relative or absolute path. Absolute paths must be within the selected plane prefix.
abspath	A BOOLEAN value that specifies whether to prepend ~ to the returned logical filename. Defaults to TRUE.
Return:	PlaneLogicalFileName returns a VARSTRING (null-terminated) value.

The **PlaneLogicalFileName** function returns an encoded plane-scoped logical filename using *~plane::<planeName>::...* syntax. Use this when constructing logical names for files addressed by storage plane and path instead of host/IP addressing.

Example:

```
OUTPUT(STD.File.PlaneLogicalFileName('mylz1', 'subdir1/subdir2/somefile.txt'));
//returns:
//~plane::mylz1::subdir1::subdir2::somefile.txt

OUTPUT(STD.File.PlaneLogicalFileName('mylz2', '/var/lib/HPCCSystems/thelz2prefix/subdir2/subdir3/somefile.txt
//returns:
//~plane::mylz2::subdir2::subdir3::somefile.txt
```

MoveExternalFile

STD.File.MoveExternalFile(*location, frompath, topath* ***[*** , planename ***]*** **)**

location	A null-terminated string containing the IP address of the remote machine. Optional if *planename* is provided.
frompath	A null-terminated string containing the path/name of the file to move.
topath	A null-terminated string containing the path/name of the target file.
planeName	A null-terminated string containing name of the data plane containing the file. Optional if *location* is provided, but *planename* is preferred.

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:

```
IMPORT STD;
IP      := '';
infile  := '/var/lib/HPCCSystems/dropzone/originalperson';
outfile := '/var/lib/HPCCSystems/dropzone/originalperson_bak';
planename := 'mydropzone';
STD.File.MoveExternalFile(IP,infile,outfile,planename);
```

DeleteExternalFile

STD.File.DeleteExternalFile(*location, path* **[** , planename **]** **)**

location	A null-terminated string containing the IP address of the remote machine. Optional if *planename* is provided.
path	A null-terminated string containing the path/name of the file to remove.
planename	A null-terminated string containing name of the data plane containing the file. Optional if *location* is provided, but *planename* is preferred.

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

Example:

```
IMPORT STD;
IP   := '';
infile := '/var/lib/HPCCSystems/dropzone/originalperson';
planename := 'mydropzone';
STD.File.DeleteExternalFile(IP,infile,planename);
```

CreateExternalDirectory

STD.File.CreateExternalDirectory(*location, path* ***[*** , planename ***]*** **)**

location	A null-terminated string containing the IP address of the remote machine. Optional if *planename* is provided.
path	A null-terminated string containing the directory path to create.
planename	A null-terminated string containing name of the data plane containing the file. Optional if *location* is provided, but *planename* is preferred.

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:

```
IMPORT STD;
IP   := '';
path := '/var/lib/HPCCSystems/dropzone/advancedtraining/';
planename := 'mydropzone';
STD.File.CreateExternalDirectory(IP,path,planename);
```

RemoteDirectory

STD.File.RemoteDirectory(*machineIP, dir* **[** *, mask* **][** *, recurse* **][** *, planeName* **])**

machineIP	A null-terminated string containing the IP address of the remote machine. Optional if *planeName* is provided.
dir	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.
mask	Optional. A null-terminated string containing the filemask specifying which files to include in the result. If omitted,the default is '*' (all files).
recurse	Optional. A boolean flag indicating whether to include files from sub-directories under the *directory*. If omitted, the default is FALSE.
planeName	A null-terminated string containing name of the data plane containing the file. Optional if *machineIP* is provided, but *planeName* is preferred.
Return:	RemoteDirectory returns a dataset in the following format:

```
EXPORT FsFilenameRecord := RECORD
  STRING name;       //filename
  UNSIGNED8 size;    //filesize
  STRING19 modified; //date-time stamp
END;
```

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

```
IMPORT STD;
machineIP   := '';
dir := '/var/lib/HPCCSystems/dropzone/training';
recurse:= FALSE;
planeName := 'mydropzone';
OUTPUT(STD.File.RemoteDirectory(machineIP,dir,'*.csv',recurse,planeName));
```

File Browsing

SetColumnMapping

STD.File.SetColumnMapping(*file, mapping* **);**

file	A null-terminated string containing the logical filename.
mapping	A null-terminated string containing a comma-delimited list of field mappings.

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:

<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:

```
// 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

result **:= STD.File.GetColumnMapping(** *file* **);**

file	A null-terminated string containing the logical filename.
Return:	GetColumnMapping returns a null-terminated string containing the comma-delimited list of field mappings for the *file*.

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

Example:

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

AddFileRelationship

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

primary	A null-terminated string containing the logical filename of the primary file.
secondary	A null-terminated string containing the logical filename of the secondary file.
primaryfields	A null-terminated string containing the name of the primary key field for the *primary* file. The value "__fileposition__" indicates the *secondary* is an INDEX that must use FETCH to access non-keyed fields.
secondaryfields	A null-terminated string containing the name of the foreign key field relating to the *primary* file.
relationship	A null-terminated string containing either "link" or "view" indicating the type of relationship between the *primary* and *secondary* files. If omitted, the default is "link."
cardinality	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 "1" or 'M'.
payload	A BOOLEAN value indicating whether the *primary* or *secondary* are payload INDEXes.
description	A null-terminated string containing the relationship description.

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:

```
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.

```
STD.File.AddFileRelationship('fetch_file',
                             'new_index',
                             '__fileposition__',
                             'index_filepos_field',
                             'link',
```

```
                              '1:1',
                              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.

```
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 subfiles.

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):

```
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:

```
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:

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

FileRelationshipList

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

primary	A null-terminated string containing the logical filename of the primary file.
secondary	A null-terminated string containing the logical filename of the secondary file.
primaryfields	A null-terminated string containing the name of the primary key field for the *primary* file. The value "__fileposition__" indicates the *secondary* is an INDEX that must use FETCH to access non-keyed fields. If omitted, the default is an empty string.
secondaryfields	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.
relationship	A null-terminated string containing either "link" or "view" indicating the type of relationship between the *primary* and *secondary* files. If omitted, the default is "link."
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:

```
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:

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

See Also: AddFileRelationship

RemoveFileRelationship

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

primary	A null-terminated string containing the logical filename of the primary file.
secondary	A null-terminated string containing the logical filename of the secondary file.
primaryfields	A null-terminated string containing the name of the primary key field for the *primary* file. The value "__fileposition__" indicates the *secondary* is an INDEX that must use FETCH to access non-keyed fields. If omitted, the default is an empty string.
secondaryfields	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.
relationship	A null-terminated string containing either "link" or "view" indicating the type of relationship between the *primary* and *secondary* files. If omitted, the default is "link."

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

Example:

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

See Also: AddFileRelationship

File Movement

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:

```
IMPORT STD;
usr := 'username=emilyd ';

pwd := 'password=password ';
ovr := 'overwrite=1 ';
repl := 'replicate=1 ';
action := 'action=spray ';
srcplane := 'srcplane=mydropzone ';
srcfile := 'srcfile=originalperson ';
dstname := 'dstname=EmilyTutorial::originalperson ';
dstcluster := 'dstcluster=data ';
fmt := 'format=fixed ';
recsize := 'recordsize=124 ';
cmd := usr + pwd + ovr + repl + action + srcplane
       + srcfile + 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:

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

Copy

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

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

sourceLogical-Name	A null-terminated string containing the logical name of the file.
destination-Group	A null-terminated string containing the destination cluster for the file.
destinationLogi-calName	A null-terminated string containing the new logical name of the file.
sourceDali	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.
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.
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. If FALSE and the file to copy is a superfile containing INDEXes, then the operation is not valid and will produce an error.
compress	Optional. A boolean TRUE or FALSE flag indicating whether to LZW compress the new file. If omitted, the default is FALSE.
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. This option is only valid within the same environment. If omitted, the default is FALSE.

transferBuffer-Size	Optional. An integer value to override the DFU Server's buffer size value (default is 64k)
preserveCom-pression	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.
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.
ensure	Optional. Copies logical file, but does not copy file parts if they already exist. Default is FALSE.
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.
dfuwuid	The attribute name to receive the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job.
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:

Note: In containerized HPCC Systems deployments, the *destinationGroup* parameter should be a named data plane (for example, **'data'**) rather than **STD.System.Thorlib.Group()**.

```
// Recommended: explicit data plane name (containerized and bare-metal)
STD.File.Copy('OUT::MyFile', 'data', 'OUT::MyNewFile');

// Also valid in containerized: empty string uses the default data plane
STD.File.Copy('OUT::MyFile', '', 'OUT::MyNewFile');

// Bare-metal: specify a cluster/group name or use STD.System.Thorlib.Group()
 STD.File.Copy('OUT::MyFile', STD.System.Thorlib.Group(), 'OUT::MyNewFile');
```

DeSpray

STD.File.DeSpray(*logicalname, destinationIP* **,** *destinationpath* , **[** *timeout* **]** , **[** *espserverIPport* **]** , **[** *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.
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.
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:

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

RemotePull

STD.File.RemotePull(*remoteURL, sourcelogicalname, destinationGroup* **,** *destinationlogicalname,* **[** *,timeout* **] [** *,maxConnections* **] [** *,allowoverwrite* **] [** *,replicate* **] [** *,asSuperfile* **] [***,forcePush* **] [***,transferBuffer-Size* **] [***,wrap* **] [***,compress* **] [** *,noSplit* **] [** *,expireDays***])**

dfuwuid **:= STD.File.fRemotePull(** *remoteURL, sourcelogicalname, destinationGroup* **,** *destinationlogicalname,* **[** *,timeout* **] [** *,maxConnections* **] [** *,allowoverwrite* **] [** *,replicate* **] [** *,asSuperfile* **] [***,forcePush* **] [***,transferBufferSize* **] [***,wrap* **] [***,compress* **] [** *,noSplit* **] [** *,expireDays***]);**

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.
sourcelogical-name	A null-terminated string containing the local logical name of the file.
destination-Group	A null-terminated string containing the name of the destination cluster.
destinationlogi-calname	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.
transferBuffer-Size	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 *espserverIPport* parameter. Since the DFU workunit executes on the remote DFU server, the user name authentication must be the same on both systems, and the use must have rights to copy files on both systems.

Example:

```
STD.File.RemotePull('http://10.150.50.14:8010/FileSpray',
    '~THOR::LOCAL::MyFile',
    'RemoteThor',
    '~REMOTETHOR::LOCAL::MyFile');
```

Replicate

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

dfuwuid **:= STD.File.fReplicate(** *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

STD.File.SprayFixed(*sourceIP* **,** *sourcepath, recordsize, destinationgroup, destinationlogicalname* **, [** *timeout* **]** , **[** *espserverIPport* **]** , **[** *maxConnections* **]** , **[** *allowoverwrite* **]** , **[** *replicate* **]** , **[** *compress* **]** , **[***failIfNoSourceFile***]** ,**[***expireDays***] , [** *dfuServerQueue* **] , [** *noSplit* **] , [** *noCommon* **],[** *sourcePlane* **],[destinationNumParts])**

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

sourceIP	A null-terminated string containing the IP address or hostname of the Drop-zone where the file is located.
sourcepath	A null-terminated string containing the path and name of the file.
recordsize	An integer containing the size of the records in the file.
destinationgroup	A null-terminated string containing the name of the specific supercomputer within the target cluster.
destinationlogicalname	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.
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 replicate the new file. If omitted, the default is FALSE.
compress	Optional. A boolean TRUE or FALSE flag indicating whether to compress the new file. If omitted, the default is TRUE in a containerized deployment and FALSE in a bare-metal deployment.
failIfNoSourceFile	Optional. A boolean TRUE or FALSE flag indicating whether a missing file triggers a failure. If omitted, the 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.
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:	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:

```
STD.File.SprayFixed('10.150.50.14','c:\\InputData\\MyFile.txt',
    255,'400way','IN::MyFile',-1,
    'http://10.150.50.12:8010/FileSpray');
```

SprayDelimited / SprayVariable

STD.File.SprayDelimited(*sourceIP* **,** *sourcePath* **,** **[** *sourceMaxRecordSize* **]** **,** **[** *sourceCsvSeparate* **]** **,** **[** *sourceCsvTerminate* **]** **,** **[** *sourceCsvQuote* **]** **,** *destinationGroup, destinationLogicalName* **,** **[** *timeout* **]** **,** **[** *espServerIpPort* **]** **,** **[** *maxConnections* **]** **,** **[** *allowOverwrite* **]** **,** **[** *replicate* **]** **,** **[** *compress* **]** **,** **[** *sourceCsvEscape* **],** **[** *failIfNoSourceFile* **],** **[** *recordStructurePresent* **],** **[** *quotedTerminator* **],** **[** *encoding* **]** **,** **[** *expireDays* **]** **,** **[** *dfuServerQueue***]** **,** **[** *noSplit***]** **,** **[***noCommon* **],[** *sourcePlane* **],[destinationNumParts])**

dfuwuid **:= STD.File.fSprayDelimited(** *sourceIP* **,** *sourcePath* **,** **[** *sourceMaxRecordSize* **]** **,** **[** *sourceCsvSeparate* **]** **,** **[** *sourceCsvTerminate* **]** **,** **[** *sourceCsvQuote* **]** *, destinationGroup, destinationLogicalName* **,** **[** *timeout* **]** **,** **[** *espServerIpPort* **]** **,** **[** *maxConnections* **]** **,** **[** *allowOverwrite* **]** **,** **[** *replicate* **]** **,** **[** *compress* **]** **,** **[** *sourceCsvEscape* **],** **[** *failIfNoSourceFile* **],** **[** *recordStructurePresent* **],** **[** *quotedTerminator* **]** **,** **[** *encoding* **]** **,** **[** *expireDays* **]** **,** **[** *dfuServerQueue***]** **,** **[** *noSplit***]** **,** **[***noCommon* **],[** *sourcePlane* **],[destinationNumParts])**

sourceIP	A null-terminated string containing the IP address or hostname of the Drop-zone where the file is located.
sourcePath	A null-terminated string containing the path and name of the file.
sourceMaxRecordSize	Optional. An integer containing the maximum size of the records in the file. If omitted, the default is 4096.
sourceCsvSeparate	Optional. A null-terminated string containing the CSV field separator. If omitted, the default is '\\,'
sourceCSVterminate	Optional. A null-terminated string containing the CSV record separator. If omitted, the default is '\\n,\\r\\n'
sourceCSVquote	Optional. A null-terminated string containing the CSV quoted field delimiter. If omitted, the default is '\'"
destinationGroup	A null-terminated string containing the name of the specific supercomputer within the target cluster.
destinationLogicalName	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.
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 replicate the new file. If omitted, the default is FALSE.
compress	Optional. A boolean TRUE or FALSE flag indicating whether to compress the new file. If omitted, the default is TRUE in a containerized deployment and FALSE in a bare-metal deployment.

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, utf16le, 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:

```
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])**

sourceIP	A null-terminated string containing the IP address or hostname of the Drop-zone where the file is located.
sourcepath	A null-terminated string containing the path and name of the file.
maxrecordsize	Optional. An integer containing the maximum size of the records in the file. If omitted, the default is 8192.
srcRowTag	A null-terminated string containing the row delimiting XML tag. Required.
srcEncoding	Optional. A null-terminated string containing the encoding. If omitted, the default is 'utf8'
destinationgroup	A null-terminated string containing the name of the specific supercomputer within the target cluster.
destinationlogicalname	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.
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 replicate the new file. If omitted, the default is FALSE.
compress	Optional. A boolean TRUE or FALSE flag indicating whether to compress the new file. If omitted, the default is TRUE in a containerized deployment and FALSE in a bare-metal deployment.
failIfNoSourceFile	Optional. A boolean TRUE or FALSE flag indicating whether a missing file triggers a failure. If omitted, the 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.

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 recieve 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).

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

Example:

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

SprayJson

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

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

sourceIP	A null-terminated string containing the IP address or hostname of the Drop-zone where the file is located.
sourcepath	A null-terminated string containing the path and name of the file.
maxrecordsize	Optional. An integer containing the maximum size of the records in the file. If omitted, the default is 8192.
sourceRowPath	The JSON path that is used to delimit records in the source file. Required.
srcEncoding	Optional. A null-terminated string containing the encoding (utf8,utf8n,utf16be,utf16le,utf32be,utf32le). If omitted, the default is 'utf8'
destinationgroup	A null-terminated string containing the name of the group to distribute the file across.
destinationlogicalname	A null-terminated string containing the logical name of the file to create.
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.
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 replicate the new file. If omitted, the default is FALSE.
compress	Optional. A boolean TRUE or FALSE flag indicating whether to compress the new file. If omitted, the default is TRUE in a containerized deployment and FALSE in a bare-metal deployment.
failIfNoSourceFile	Optional. A boolean TRUE or FALSE flag indicating whether a missing file triggers a failure. If omitted, the 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.

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.
username	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.
userPw:	Optional. String containing the password to be used with the user cited in the *username* argument; if *username* is empty then this is ignored. If omitted, the default is an empty string.
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:

```
STD.File.SprayJson('10.150.50.14','/var/lib/HPCCSystems/mydropzone/colors.json',,
      '/',,'mythor','examples::colors.json',-1,
      'http://10.150.50.12:8010/FileSpray');
```

WaitDfuWorkunit

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

dfuwuid	A null-terminated string containing the DFU workunit ID (DFUWUID) for the job to wait for. This value is returned by the "leading-f" versions of the Copy, DKC, SprayFixed, SprayVariable, SprayXML, and Despray FileServices functions.
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.
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:

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

SetExpireDays

STD.File.SetExpireDays(*lfn***,** *expireDays* **)**

lfn	A string containing the logical name of the file.
expireDays	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).

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(*lfn***)**

lfn	A string containing the logical name of the file.

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

Example:

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

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:

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

See Also: GetExpireDays, SetExpireDays

String Handling

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:

```
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* **)**

source	A string containing the data to clean.
Return:	CleanSpaces returns either a STRING or UNICODE value, as appropriate.

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'
```

CommonPrefix

STD.Str.CommonPrefix(*s1, s2* **[** *,nocase* **])**

STD.Uni.CommonPrefix(*s1, s2* **[** *,nocase* **])**

s1	A string to compare.
s2	A string to compare.
nocase	Optional. If TRUE, the comparison is case-insensitive. If omitted, the default is FALSE.
Return:	CommonPrefix returns either a STRING or UNICODE value, as appropriate. It contains the longest prefix common to both strings, as copied from the first argument. The result is empty if the strings have no common prefix or if either argument is empty.

The **CommonPrefix** function returns the longest prefix common to both strings. This can be used for identifying shared prefixes between strings, which can be helpful in text processing tasks such as pattern matching, data normalization, or linguistic analysis.

Example:

```
IMPORT Std;
Std.Str.CommonPrefix('DANIEL', 'DANNY',nocase:=FALSE); // DAN
Std.Str.CommonPrefix('DANIEL', 'Danny',FALSE);         // D
Std.Str.CommonPrefix('DANIEL', 'Danny',TRUE);          // DAN
Std.Str.CommonPrefix('APPLES', 'ORANGES',FALSE);       // empty
```

See Also: CommonSuffix

CommonSuffix

STD.Str.CommonSuffix(*s1, s2* **[** *,nocase* **])**

STD.Uni.CommonSuffix(*s1, s2* **[** *,nocase* **])**

s1	A string to compare.
s2	A string to compare.
nocase	Optional. If TRUE, the comparison is case-insensitive. If omitted, the default is FALSE.
Return:	CommonSuffix returns either a STRING or UNICODE value, as appropriate. It contains the longest suffix common to both strings, as copied from the first argument. The result is empty if the strings have no common suffix or if either argument is empty.

The **CommonSuffix** function returns the longest suffix common to both strings. This can be useful for identifying shared endings between strings, which can be helpful in text processing tasks such as pattern matching, data normalization, or linguistic analysis.

Example:

```
IMPORT Std;
Std.Str.CommonSuffix('SUNLIGHT', 'MOONLIGHT',nocase:=FALSE); // NLIGHT
Std.Str.CommonSuffix('TABLETOP', 'Laptop',FALSE);            // empty
Std.Str.CommonSuffix('TABLETOP', 'Laptop',TRUE);             // TOP
Std.Str.CommonSuffix('APPLES', 'ORANGES',FALSE);             // ES
```

See Also: CommonPrefix

CompareAtStrength

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

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

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

```
base := u'caf\u00E9';   // 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\u00C9';   // 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* **)**

source1	A string containing the data to compare.
source2	A string containing the data to compare.
locale	A null-terminated string containing the language and country code to use to determine correct sort order and other operations.
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* **)**

source	A string containing the data to search.
pattern	A string containing the characters to compare. An empty string ('') always returns true.
nocase	A boolean true or false indicating whether to ignore the case.
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
```

See Also: Find

CountWords

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

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

source	A string containing the words to count.
separator	A string containing the word delimiter to use.
allow_blank	Optional, A BOOLEAN value indicating if empty/blank string items are included in the results. Defaults to FALSE
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* **)**

value	A STRING value containing the data to decode.
Return:	DecodeBase64 returns a DATA value.

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

Example:

```
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* **)**

string1	The first of a pair of strings to compare.
string2	The second of a pair of strings to compare.
locale	A null-terminated string containing the language and country code to use to determine correct sort order and other operations.
radius	Optional. The maximum acceptable edit distance, or 0 for no limit. Defaults to 0.
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 1
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* **)**

string1	The first of a pair of strings to compare.
string2	The second of a pair of strings to compare.
radius	An integer specifying the maximum acceptable edit distance.
locale	A null-terminated string containing the language and country code to use to determine correct sort order and other operations.
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* **])**

value	A DATA value containing the data to encode.
insertLF	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.
Return:	EncodeBase64 returns a STRING value.

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

Example:

```
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* **)**

src	The string to search.
suffix	The string to find.
form	The type of Unicode normalization to be employed. (NFC, NFD, NFKC, or NFKD)
Return:	EndsWith returns a BOOLEAN value.

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:

```
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* **)**

source1	A string containing the data to compare.
source2	A string containing the data to compare.
Return:	EqualIgnoreCase returns a BOOLEAN value.

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

Example:

```
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* **)**

text	A string containing words separated by whitespace.
localename	Optional. The locale to use for the break semantics. Defaults to ''
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:

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

ExcludeLastWord

STD.Str.ExcludeLastWord(*text* **)**

STD.Uni.ExcludeLastWord(*text, localename* **)**

text	A string containing words separated by whitespace.
localename	Optional. The locale to use for the break semantics. Defaults to ''
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:

```
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* **)**

text	A string containing words separated by whitespace.
n	A integer containing the ordinal position of the word to remove.
localename	Optional. The locale to use for the break semantics. Defaults to ''
Return:	ExcludeNthWord returns a STRING or UNICODE value, as appropriate.

The **ExcludeNthWord** function returns the *text* string with the *n*th 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:

```
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* **)**

source	A string containing a comma-delimited list of data.
instance	An integer specifying the ordinal position of the data item within the *source* to return.
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* **)**

source	A string containing a comma-delimited list of data.
mask	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..
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* **)**

source	A string containing the data to filter.
filterstring	A string containing the characters to use as the filter.
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:

```
//all these examples result in 'Success'

A := IF(STD.Str.Filter('ADCBE', 'BD') = 'DB',
   'Success',
   'Failure - 1');

B := IF(STD.Str.Filter('ADCBEREBD', 'BDG') = 'DBBD',
   'Success',
   'Failure - 2');

C := IF(STD.Str.Filter('ADCBE', '') = '',
   '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* **)**

source	A string containing the data to filter.
filterstring	A string containing the characters to use as the filter.
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:

```
//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* **)**

source	A string containing the data to search.
target	A string containing the substring to search for.
instance	An integer specifying which occurrence of the *target* to find.
locale	A null-terminated string containing the language and country code to use to determine correct sort order and other operations.
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
```

See Also: Contains

FindCount

STD.Str.FindCount(*src, sought* **)**

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

src	A string containing the data to search.
sought	A string containing the substring to search for.
form	The type of Unicode normalization to be employed. (NFC, NFD, NFKC, or NFKD)
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* **)**

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

```
base := u'caf\u00E9';   // 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\u00C9';   // 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\u00E9 vert',
         u'cafe', 1, 'fr', 2) = 4;
  // however, an accent on the source,
STD.Uni.LocaleFindAtStrength(u'le caf\u00E9 vert',
         u'cafe', 1, 'fr', 3) = 4;
 // rather than on the pattern,
STD.Uni.LocaleFindAtStrength(u'le caf\u00E9 vert',
         u'cafe', 1, 'fr', 4) = 4;
 // is ignored at strengths up to 4,
STD.Uni.LocaleFindAtStrength(u'le caf\u00E9 vert',
         u'cafe', 1, 'fr', 5) = 0;
 // and only counts at strength 5
```

FindAtStrengthReplace

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

source	A string containing the data to search.
target	A string containing the substring to search for.
replacement	A string containing the replacement data.
locale	A null-terminated string containing the language and country code to use to determine correct sort order and other operations.
strength	An integer value indicating how to compare. Valid values are:
	1 ignores accents and case, differentiating only between letters.
	2 ignores case but differentiates between accents.
	3 differentiates between accents and case but ignores e.g. differences between Hiragana and Katakana
	4 differentiates between accents and case and e.g. Hiragana/Katakana, but ignores e.g. Hebrew cantellation marks
	5 differentiates between all strings whose canonically decomposed forms (NFD--Normalization Form D) are non-identical
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'e\u00E8E\u00C9eE',
   u'e\u00E9', u'xyz', 'fr', 1) = u'xyzxyzxyz';
STD.Uni.LocaleFindAtStrengthReplace(u'e\u00E8E\u00C9eE',
   u'e\u00E9', u'xyz', 'fr', 2) = u'e\u00E8xyzeE';
STD.Uni.LocaleFindAtStrengthReplace(u'e\u00E8E\u00C9eE',
   u'e\u00E9', u'xyz', 'fr', 3) = u'e\u00E8E\u00C9eE';
```

FindReplace

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

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

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

source	A string containing the data to search.
target	A string containing the substring to search for.
replacement	A string containing the replacement data.
locale	A null-terminated string containing the language and country code to use to determine correct sort order and other operations.
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:

```
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'gh\u0131klm', 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'
```

FindWord

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

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

src	A string containing the data to search.
word	A string containing the substring to search for.
ignore_case	A boolean true or false to indicate whether to ignore the case.
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* **)**

source	The string containing the hex pairs to process.
Return:	FromHexPairs returns a data value with each byte created from a pair of hex digits.

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

Example:

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

GetNthWord

STD.Str.GetNthWord(*source, instance* **)**

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

source	A string containing the space-delimited words.
instance	An integer specifying the word to return.
locale	A null-terminated string containing the language and country code to use to determine correct sort order and other operations.
Return:	GetNthWord returns a string value.

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

Example:

```
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* **)**

src	The string to search.
suffix	The ending string to remove.
form	The type of Unicode normalization to be employed. (NFC, NFD, NFKC, or NFKD)
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:

```
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

STD.Str.Repeat(*text*, *n* **)**

STD.Uni.Repeat(*text*, *n* **)**

text	The string to be repeated (maximum length is 255 characters).
n	The number of repetitions.
Return:	Repeat returns a STRING containing *n* concatenations of the string text..

The **Repeat** function returns the *text* string repeated n times.

Example:

```
A := STD.Str.Repeat('ABC',3); //A contains 'ABCABCABC'
```

Reverse

STD.Str.Reverse(*source* **)**

STD.Uni.Reverse(*source* **)**

source	A string containing the data to reverse.
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* **])**

src	A string containing the words to extract.
separator	A string containing the word delimiter to use.
allow_blank	Optional. If TRUE, specifies allowing blank items in the result. If omitted, the default is FALSE.
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:

```
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* **)**

source	A string containing the data to search.
target	A string containing the characters to search for.
replacement	A string containing the replacement character as its first character.
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'xxcdxxxcdxc';
```

SubstituteIncluded

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

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

source	A string containing the data to search.
target	A string containing the characters to search for.
replacement	A string containing the replacement character as its first character.
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* **)**

src	The string to search.
prefix	The string to find.
form	The type of Unicode normalization to be employed. (NFC, NFD, NFKC, or NFKD)
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:

```
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* **)**

source	The data value that should be expanded as a sequence of hex pairs.
Return:	ToHexPairs returns a string containing a sequence of hex pairs.

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* **)**

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:	ToLowerCase returns a STRING or UNICODE value, as appropriate.

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* **)**

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:	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* **)**

src	A string containing the characters to search.
search	A string containing the characters to be replaced by characters in the *replacement* string.
replacement	A string containing the characters to act as replacements.
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:

```
IMPORT STD;
UniTest:= IF ((INTEGER)std.Uni.Version() < 50,'Not Supported',(STRING)std.Uni.WordCount(U'我是電腦'));
                                        //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* **])**

source	A string containing the words to count. Words are delimited by spaces.
locale	A null-terminated string containing the language and country code to use to determine correct sort order and other operations.
Return:	WordCount returns an integer value.

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

Example:

```
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 **)**

source	The string to process.
Return:	Primary returns a STRING value.

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:

```
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 **)**

source	The string to process.
Return:	Secondary returns a STRING value.

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:

```
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 **)**

source	The string to process.
Return:	Double returns a STRING value.

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:

```
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(*);*

Return:	SET OF STRINGs containing all supported Public Key Algorithms

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

Example:

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

Hashing Module

myHashModule := **STD.Crypto.Hashing(***hashAlgorithm***);**

myHashModule	The name of the Hashing module structure
hashAlgorithm	The hashing algorithm to use, as returned by SupportedHashAlgorithms()

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

Example:

```
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***);**

myHashModule	The name of the Hashing module structure
inputData	The data to hash in DATA format
Return:	Hashed contents in DATA format

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

Example:

```
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

mySymEncModule := **STD.Crypto.SymmetricEncryption(***algorithm, passphrase);*

mySymEncModule	The name of the Symmetric Encryption module structure
algorithm	The algorithm to use, as returned by SupportedSymmetricCipherAlgorithms()
passphrase	The passphrase to use for encryption/decryption

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

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);
```

Encrypt (Symmetric)

mySymEncModule.**Encrypt(***inputData***)**;

mySymEncModule	The name of the Symmetric 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 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);
```

Decrypt (Symmetric)

mySymEncModule.**Decrypt(***encryptedData***);**

mySymEncModule	The name of the Symmetric Encryption 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 defined in the Symmetric Encryption module. You can only decrypt data that was encrypted by the Standard Library's Encrypt method.

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);
```

PublicKeyEncryption Module

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

myPKEModule	The name of the Public Key Encryption module structure
pkAlgorithm	The algorithm to use, as returned by SupportedPublicKeyAlgorithms()
publicKeyFile	The File Specification of the PEM formatted Public Key file
privateKeyFile	The File Specification of the PEM formatted Private Key file
passphrase	The passphrase to use for encryption, decryption, signing, verifying

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:

```
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) );
```

Encrypt (PKE)

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 module definition.

Example:

```
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)

myPKEModule.**Decrypt(***encryptedData***);**

myPKEModule	The name of the Public Key Encryption 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 module definition. You can only decrypt data that was encrypted by the Standard Library's Encrypt method.

Example:

```
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) );
```

Sign (PKE)

mySymEncModule.**Sign(***encryptedData***);**

myPKEModule	The name of the Public Key Encryption module structure
inputData	The data to sign in DATA format
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:

```
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***);**

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 module definition.

Example:

```
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'));
```

PublicKeyEncryptionFromBuffer Module

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:

```
IMPORT STD;

STRING publicKey := '-----BEGIN PUBLIC KEY-----' + '\n' +
'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KMnWRAof' + '\n' +
'od+3AUS/IDngT39j3Iovv9aI2N8g4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeE' + '\n' +
'BHqlMDydw9aHOQG17CB30GYsw3Lf8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoy' + '\n' +
'hIR9MexCldF+3WM/X0IX0ApSs7kuVPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv' + '\n' +
'/oKj6q7kInEIvhLiGfcm3bpTzWQ66zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3' + '\n' +
'J6Tk4NY3NySWzE/2/ZOWxZdR79XC+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0' + '\n' +
'bwIDAQAB' + '\n' +
'-----END PUBLIC KEY-----';

STRING privateKey := '-----BEGIN RSA PRIVATE KEY-----' + '\n' +
'MIIEowIBAAKCAQEAr64RncTp5pV0KMnWRAofod+3AUS/IDngT39j3Iovv9aI2N8g' + '\n' +
'4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeEBHqlMDydw9aHOQG17CB30GYsw3Lf' + '\n' +
'8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoyhIR9MexCldF+3WM/X0IX0ApSs7ku' + '\n' +
'VPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv/oKj6q7kInEIvhLiGfcm3bpTzWQ6' + '\n' +
'6zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3J6Tk4NY3NySWzE/2/ZOWxZdR79XC' + '\n' +
'+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0bwIDAQABAoIBAQCnGAtNYkOOu8wW' + '\n' +
'F5Oid3aKwnwPytF211WQh3v2AcFU17qle+SMRi+ykBL6+u5RU5qH+HSc9Jm31AjW' + '\n' +
'V1yPrdYVZInFjYIJCPzorcXY5zDOmMAuzg5PBVV7VhUA0a5GZck6FC8AilDUcEom' + '\n' +
'GCK6Ul8mR9XELBFQ6keeTo2yDu0TQ4oBXrPBMN61uMHCxh2tDb2yvl8Zz+EllADG' + '\n' +
'70pztRWNOrCzrC+ARlmmDfYOUgVFtZin53jq6O6ullPLzhkm3/+QFRGYWsFgQB6J' + '\n' +
'Z9HJtW5YB47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AaNAUzc/G' + '\n' +
'sB0cNeURAoGBAOKtaVFa6z2F4Q+koMBXCt4m7dCJnaC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jhhvuV0OcN8lYbR/ZlYRJDUs6mHh/2BYSkdeaLKojXTxKR2bA4xQk5dtJCdoPf' + '\n' +
'0c15AlTgOYk2oNXP/azDICJYT/cdvIdUL9P4IoZthu1FjwG266GacEnNAoGBAMZn' + '\n' +
'1wRUXS1dbqemoc+g48wj5r3/qsIG8PsZ2Y8W+oYW7diNA5o6acc8YPEWE2RbJDbX' + '\n' +
'YEADBnRSdzzOdo0JEj4VbNZEtx6nQhBOOrtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
'eCzO1VcEeZtsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAI+fOD+nz6znirYSpRe5D' + '\n' +
'tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DjXrT09QmVAMciNEnc/sZBjiNzFf525wv' + '\n' +
'wZP/bPZMVYKtbsaVkdlcNJranHGUrkzswbxSRzmBQ5/YmCWrDAuYcnhEqmMWcuU9' + '\n' +
'8jiS13JP9hOXlHDyIBYDhV0CgYBV6TznuQgnzp9NpQ/H8ijxilItz3lHTu4mLMlR' + '\n' +
'9mdAjMkszdLTg5uuE+z+N8rp17VUseoRjb3LvLG4+MXIyDbH/0sDdPm+IjqvCNDR' + '\n' +
'spmh9MgBh0JbsbWaZK0s9/qrI/FcSLZ04JLsfRmTPU/Y5y8/dHjYO6fDQhp44RZF' + '\n' +
'iCqNxQKBgHf7KZIOKgV4YNyphk1UYWHNz8YY5o7WtaQ51Q+kIbU8PRd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjxi8GKQzE/IdkQqiFmB1yEjjRHQ81WS+K5NnjN1t0IEscJqOAwv9s' + '\n' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWxRHEYdegqU48f+qxUcJj9R' + '\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)

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:

```
IMPORT STD;

STRING publicKey := '-----BEGIN PUBLIC KEY-----' + '\n' +
'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KMnWRAof' + '\n' +
'od+3AUS/IDngT39j3Iovv9aI2N8g4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeE' + '\n' +
'BHqlMDydw9aHOQG17CB30GYsw3Lf8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoy' + '\n' +
'hIR9MexCldF+3WM/X0IX0ApSs7kuVPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv' + '\n' +
'/oKj6q7kInEIvhLiGfcm3bpTzWQ66zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3' + '\n' +
'J6Tk4NY3NySWzE/2/ZOWxZdR79XC+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0' + '\n' +
'bwIDAQAB' + '\n' +
'-----END PUBLIC KEY-----';

STRING privateKey := '-----BEGIN RSA PRIVATE KEY-----' + '\n' +
'MIIEowIBAAKCAQEAr64RncTp5pV0KMnWRAofod+3AUS/IDngT39j3Iovv9aI2N8g' + '\n' +
'4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeEBHqlMDydw9aHOQG17CB30GYsw3Lf' + '\n' +
'8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoyhIR9MexCldF+3WM/X0IX0ApSs7ku' + '\n' +
'VPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv/oKj6q7kInEIvhLiGfcm3bpTzWQ6' + '\n' +
'6zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3J6Tk4NY3NySWzE/2/ZOWxZdR79XC' + '\n' +
'+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0bwIDAQABAoIBAQCnGAtNYkOOu8wW' + '\n' +
'F5Oid3aKwnwPytF211WQh3v2AcFU17qle+SMRi+ykBL6+u5RU5qH+HSc9Jm31AjW' + '\n' +
'V1yPrdYVZInFjYIJCPzorcXY5zDOmMAuzg5PBVV7VhUA0a5GZck6FC8AilDUcEom' + '\n' +
'GCK6Ul8mR9XELBFQ6keeTo2yDu0TQ4oBXrPBMN61uMHCxh2tDb2yvl8Zz+EllADG' + '\n' +
'70pztRWNOrCzrC+ARlmmDfYOUgVFtZin53jq6O6ullPLzhkm3/+QFRGYWsFgQB6J' + '\n' +
'Z9HJtW5YB47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AaNAUzc/G' + '\n' +
'sB0cNeURAoGBAOKtaVFa6z2F4Q+koMBXCt4m7dCJnaC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jhhvuV0OcN8lYbR/ZlYRJDUs6mHh/2BYSkdeaLKojXTxKR2bA4xQk5dtJCdoPf' + '\n' +
'0c15AlTgOYk2oNXP/azDICJYT/cdvIdUL9P4IoZthu1FjwG266GacEnNAoGBAMZn' + '\n' +
'1wRUXS1dbqemoc+g48wj5r3/qsIG8PsZ2Y8W+oYW7diNA5o6acc8YPEWE2RbJDbX' + '\n' +
'YEADBnRSdzzOdo0JEj4VbNZEtx6nQhBOOrtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
'eCzO1VcEeZtsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAI+fOD+nz6znirYSpRe5D' + '\n' +
'tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DjXrT09QmVAMciNEnc/sZBjiNzFf525wv' + '\n' +
'wZP/bPZMVYKtbsaVkdlcNJranHGUrkzswbxSRzmBQ5/YmCWrDAuYcnhEqmMWcuU9' + '\n' +
'8jiS13JP9hOXlHDyIBYDhV0CgYBV6TznuQgnzp9NpQ/H8ijxilItz3lHTu4mLMlR' + '\n' +
'9mdAjMkszdLTg5uuE+z+N8rp17VUseoRjb3LvLG4+MXIyDbH/0sDdPm+IjqvCNDR' + '\n' +
'spmh9MgBh0JbsbWaZK0s9/qrI/FcSLZ04JLsfRmTPU/Y5y8/dHjYO6fDQhp44RZF' + '\n' +
'iCqNxQKBgHf7KZIOKgV4YNyphk1UYWHNz8YY5o7WtaQ51Q+kIbU8PRd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjxi8GKQzE/IdkQqiFmB1yEjjRHQ81WS+K5NnjN1t0IEscJqOAwv9s' + '\n' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWxRHEYdegqU48f+qxUcJj9R' + '\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));
```

Decrypt (PKE From Buffer)

myPKEModule.**Decrypt(***encryptedData***);**

myPKEModule	The name of the Public Key Encryption 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 Buffer module definition. You can only decrypt data that was encrypted by the Standard Library's Encrypt method.

Example:

```
IMPORT STD;

STRING publicKey := '-----BEGIN PUBLIC KEY-----' + '\n' +
'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KMnWRAof' + '\n' +
'od+3AUS/IDngT39j3Iovv9aI2N8g4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeE' + '\n' +
'BHqlMDydw9aHOQG17CB30GYsw3Lf8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoy' + '\n' +
'hIR9MexCldF+3WM/X0IX0ApSs7kuVPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv' + '\n' +
'/oKj6q7kInEIvhLiGfcm3bpTzWQ66zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3' + '\n' +
'J6Tk4NY3NySWzE/2/ZOWxZdR79XC+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0' + '\n' +
'bwIDAQAB' + '\n' +
'-----END PUBLIC KEY-----';

STRING privateKey := '-----BEGIN RSA PRIVATE KEY-----' + '\n' +
'MIIEowIBAAKCAQEAr64RncTp5pV0KMnWRAofod+3AUS/IDngT39j3Iovv9aI2N8g' + '\n' +
'4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeEBHqlMDydw9aHOQG17CB30GYsw3Lf' + '\n' +
'8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoyhIR9MexCldF+3WM/X0IX0ApSs7ku' + '\n' +
'VPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv/oKj6q7kInEIvhLiGfcm3bpTzWQ6' + '\n' +
'6zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3J6Tk4NY3NySWzE/2/ZOWxZdR79XC' + '\n' +
'+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0bwIDAQABAoIBAQCnGAtNYkOOu8wW' + '\n' +
'F5Oid3aKwnwPytF211WQh3v2AcFU17qle+SMRi+ykBL6+u5RU5qH+HSc9Jm31AjW' + '\n' +
'V1yPrdYVZInFjYIJCPzorcXY5zDOmMAuzg5PBVV7VhUA0a5GZck6FC8AilDUcEom' + '\n' +
'GCK6Ul8mR9XELBFQ6keeTo2yDu0TQ4oBXrPBMN61uMHCxh2tDb2yvl8Zz+EllADG' + '\n' +
'70pztRWNOrCzrC+ARlmmDfYOUgVFtZin53jq6O6ullPLzhkm3/+QFRGYWsFgQB6J' + '\n' +
'Z9HJtW5YB47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AaNAUzc/G' + '\n' +
'sB0cNeURAoGBAOKtaVFa6z2F4Q+koMBXCt4m7dCJnaC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jhhvuV0OcN8lYbR/ZlYRJDUs6mHh/2BYSkdeaLKojXTxKR2bA4xQk5dtJCdoPf' + '\n' +
'0c15AlTgOYk2oNXP/azDICJYT/cdvIdUL9P4IoZthu1FjwG266GacEnNAoGBAMZn' + '\n' +
'1wRUXS1dbqemoc+g48wj5r3/qsIG8PsZ2Y8W+oYW7diNA5o6acc8YPEWE2RbJDbX' + '\n' +
'YEADBnRSdzzOdo0JEj4VbNZEtx6nQhBOOrtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
'eCzO1VcEeZtsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAI+fOD+nz6znirYSpRe5D' + '\n' +
'tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DjXrT09QmVAMciNEnc/sZBjiNzFf525wv' + '\n' +
'wZP/bPZMVYKtbsaVkdlcNJranHGUrkzswbxSRzmBQ5/YmCWrDAuYcnhEqmMWcuU9' + '\n' +
'8jiS13JP9hOXlHDyIBYDhV0CgYBV6TznuQgnzp9NpQ/H8ijxilItz3lHTu4mLMlR' + '\n' +
'9mdAjMkszdLTg5uuE+z+N8rp17VUseoRjb3LvLG4+MXIyDbH/0sDdPm+IjqvCNDR' + '\n' +
'spmh9MgBh0JbsbWaZK0s9/qrI/FcSLZ04JLsfRmTPU/Y5y8/dHjYO6fDQhp44RZF' + '\n' +
'iCqNxQKBgHf7KZIOKgV4YNyphk1UYWHNz8YY5o7WtaQ51Q+kIbU8PRd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjxi8GKQzE/IdkQqiFmB1yEjjRHQ81WS+K5NnjN1t0IEscJqOAwv9s' + '\n' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWxRHEYdegqU48f+qxUcJj9R' + '\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));
```

Sign (PKE From Buffer)

mySymEncModule.**Sign(***encryptedData***);**

myPKEModule	The name of the Public Key Encryption module structure
inputData	The data to sign in DATA format
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-----' + '\n' +
'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KMnWRAof' + '\n' +
'od+3AUS/IDngT39j3Iovv9aI2N8g4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeE' + '\n' +
'BHqlMDydw9aHOQG17CB30GYsw3Lf8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoy' + '\n' +
'hIR9MexCldF+3WM/X0IX0ApSs7kuVPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv' + '\n' +
'/oKj6q7kInEIvhLiGfcm3bpTzWQ66zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3' + '\n' +
'J6Tk4NY3NySWzE/2/ZOWxZdR79XC+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0' + '\n' +
'bwIDAQAB' + '\n' +
'-----END PUBLIC KEY-----';

STRING privateKey := '-----BEGIN RSA PRIVATE KEY-----' + '\n' +
'MIIEowIBAAKCAQEAr64RncTp5pV0KMnWRAofod+3AUS/IDngT39j3Iovv9aI2N8g' + '\n' +
'4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeEBHqlMDydw9aHOQG17CB30GYsw3Lf' + '\n' +
'8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoyhIR9MexCldF+3WM/X0IX0ApSs7ku' + '\n' +
'VPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv/oKj6q7kInEIvhLiGfcm3bpTzWQ6' + '\n' +
'6zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3J6Tk4NY3NySWzE/2/ZOWxZdR79XC' + '\n' +
'+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0bwIDAQABAoIBAQCnGAtNYkOOu8wW' + '\n' +
'F5Oid3aKwnwPytF211WQh3v2AcFU17qle+SMRi+ykBL6+u5RU5qH+HSc9Jm31AjW' + '\n' +
'V1yPrdYVZInFjYIJCPzorcXY5zDOmMAuzg5PBVV7VhUA0a5GZck6FC8AilDUcEom' + '\n' +
'GCK6Ul8mR9XELBFQ6keeTo2yDu0TQ4oBXrPBMN61uMHCxh2tDb2yvl8Zz+EllADG' + '\n' +
'70pztRWNOrCzrC+ARlmmDfYOUgVFtZin53jq6O6ullPLzhkm3/+QFRGYWsFgQB6J' + '\n' +
'Z9HJtW5YB47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AaNAUzc/G' + '\n' +
'sB0cNeURAoGBAOKtaVFa6z2F4Q+koMBXCt4m7dCJnaC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jhhvuV0OcN8lYbR/ZlYRJDUs6mHh/2BYSkdeaLKojXTxKR2bA4xQk5dtJCdoPf' + '\n' +
'0c15AlTgOYk2oNXP/azDICJYT/cdvIdUL9P4IoZthu1FjwG266GacEnNAoGBAMZn' + '\n' +
'1wRUXS1dbqemoc+g48wj5r3/qsIG8PsZ2Y8W+oYW7diNA5o6acc8YPEWE2RbJDbX' + '\n' +
'YEADBnRSdzzOdo0JEj4VbNZEtx6nQhBOOrtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
'eCzO1VcEeZtsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAI+fOD+nz6znirYSpRe5D' + '\n' +
'tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DjXrT09QmVAMciNEnc/sZBjiNzFf525wv' + '\n' +
'wZP/bPZMVYKtbsaVkdlcNJranHGUrkzswbxSRzmBQ5/YmCWrDAuYcnhEqmMWcuU9' + '\n' +
'8jiS13JP9hOXlHDyIBYDhV0CgYBV6TznuQgnzp9NpQ/H8ijxilItz3lHTu4mLMlR' + '\n' +
'9mdAjMkszdLTg5uuE+z+N8rp17VUseoRjb3LvLG4+MXIyDbH/0sDdPm+IjqvCNDR' + '\n' +
'spmh9MgBh0JbsbWaZK0s9/qrI/FcSLZ04JLsfRmTPU/Y5y8/dHjYO6fDQhp44RZF' + '\n' +
'iCqNxQKBgHf7KZIOKgV4YNyphk1UYWHNz8YY5o7WtaQ51Q+kIbU8PRd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjxi8GKQzE/IdkQqiFmB1yEjjRHQ81WS+K5NnjN1t0IEscJqOAwv9s' + '\n' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWxRHEYdegqU48f+qxUcJj9R' + '\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'));
```

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-----' + '\n' +
'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KMnWRAof' + '\n' +
'od+3AUS/IDngT39j3Iovv9aI2N8g4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeE' + '\n' +
'BHqlMDydw9aHOQG17CB30GYsw3Lf8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoy' + '\n' +
'hIR9MexCldF+3WM/X0IX0ApSs7kuVPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv' + '\n' +
'/oKj6q7kInEIvhLiGfcm3bpTzWQ66zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3' + '\n' +
'J6Tk4NY3NySWzE/2/ZOWxZdR79XC+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0' + '\n' +
'bwIDAQAB' + '\n' +
'-----END PUBLIC KEY-----';

STRING privateKey := '-----BEGIN RSA PRIVATE KEY-----' + '\n' +
'MIIEowIBAAKCAQEAr64RncTp5pV0KMnWRAofod+3AUS/IDngT39j3Iovv9aI2N8g' + '\n' +
'4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeEBHqlMDydw9aHOQG17CB30GYsw3Lf' + '\n' +
'8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoyhIR9MexCldF+3WM/X0IX0ApSs7ku' + '\n' +
'VPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv/oKj6q7kInEIvhLiGfcm3bpTzWQ6' + '\n' +
'6zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3J6Tk4NY3NySWzE/2/ZOWxZdR79XC' + '\n' +
'+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0bwIDAQABAoIBAQCnGAtNYkOOu8wW' + '\n' +
'F5Oid3aKwnwPytF211WQh3v2AcFU17qle+SMRi+ykBL6+u5RU5qH+HSc9Jm31AjW' + '\n' +
'V1yPrdYVZInFjYIJCPzorcXY5zDOmMAuzg5PBVV7VhUA0a5GZck6FC8AilDUcEom' + '\n' +
'GCK6Ul8mR9XELBFQ6keeTo2yDu0TQ4oBXrPBMN61uMHCxh2tDb2yvl8Zz+EllADG' + '\n' +
'70pztRWNOrCzrC+ARlmmDfYOUgVFtZin53jq6O6ullPLzhkm3/+QFRGYWsFgQB6J' + '\n' +
'Z9HJtW5YB47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AaNAUzc/G' + '\n' +
'sB0cNeURAoGBAOKtaVFa6z2F4Q+koMBXCt4m7dCJnaC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jhhvuV0OcN8lYbR/ZlYRJDUs6mHh/2BYSkdeaLKojXTxKR2bA4xQk5dtJCdoPf' + '\n' +
'0c15AlTgOYk2oNXP/azDICJYT/cdvIdUL9P4IoZthu1FjwG266GacEnNAoGBAMZn' + '\n' +
'1wRUXS1dbqemoc+g48wj5r3/qsIG8PsZ2Y8W+oYW7diNA5o6acc8YPEWE2RbJDbX' + '\n' +
'YEADBnRSdzzOdo0JEj4VbNZEtx6nQhBOOrtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
'eCzO1VcEeZtsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAI+fOD+nz6znirYSpRe5D' + '\n' +
'tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DjXrT09QmVAMciNEnc/sZBjiNzFf525wv' + '\n' +
'wZP/bPZMVYKtbsaVkdlcNJranHGUrkzswbxSRzmBQ5/YmCWrDAuYcnhEqmMWcuU9' + '\n' +
'8jiS13JP9hOXlHDyIBYDhV0CgYBV6TznuQgnzp9NpQ/H8ijxilItz3lHTu4mLMlR' + '\n' +
'9mdAjMkszdLTg5uuE+z+N8rp17VUseoRjb3LvLG4+MXIyDbH/0sDdPm+IjqvCNDR' + '\n' +
'spmh9MgBh0JbsbWaZK0s9/qrI/FcSLZ04JLsfRmTPU/Y5y8/dHjYO6fDQhp44RZF' + '\n' +
'iCqNxQKBgHf7KZIOKgV4YNyphk1UYWHNz8YY5o7WtaQ51Q+kIbU8PRd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjxi8GKQzE/IdkQqiFmB1yEjjRHQ81WS+K5NnjN1t0IEscJqOAwv9s' + '\n' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWxRHEYdegqU48f+qxUcJj9R' + '\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'));
```

PublicKeyEncryptionFromLFN 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:

```
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' +
'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KMnWRAof' + '\n' +
'od+3AUS/IDngT39j3Iovv9aI2N8g4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeE' + '\n' +
'BHqlMDydw9aHOQG17CB30GYsw3Lf8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoy' + '\n' +
'hIR9MexCldF+3WM/X0IX0ApSs7kuVPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv' + '\n' +
'/oKj6q7kInEIvhLiGfcm3bpTzWQ66zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3' + '\n' +
'J6Tk4NY3NySWzE/2/ZOWxZdR79XC+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0' + '\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' +
'MIIEowIBAAKCAQEAr64RncTp5pV0KMnWRAofod+3AUS/IDngT39j3Iovv9aI2N8g' + '\n' +
'4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeEBHqlMDydw9aHOQG17CB30GYsw3Lf' + '\n' +
'8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoyhIR9MexCldF+3WM/X0IX0ApSs7ku' + '\n' +
'VPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv/oKj6q7kInEIvhLiGfcm3bpTzWQ6' + '\n' +
'6zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3J6Tk4NY3NySWzE/2/ZOWxZdR79XC' + '\n' +
'+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0bwIDAQABAoIBAQCnGAtNYkOOu8wW' + '\n' +
'F5Oid3aKwnwPytF211WQh3v2AcFU17qle+SMRi+ykBL6+u5RU5qH+HSc9Jm31AjW' + '\n' +
'V1yPrdYVZInFjYIJCPzorcXY5zDOmMAuzg5PBVV7VhUA0a5GZck6FC8AilDUcEom' + '\n' +
'GCK6Ul8mR9XELBFQ6keeTo2yDu0TQ4oBXrPBMN61uMHCxh2tDb2yvl8Zz+EllADG' + '\n' +
'70pztRWNOrCzrC+ARlmmDfYOUgVFtZin53jq6O6ullPLzhkm3/+QFRGYWsFgQB6J' + '\n' +
```

```
'Z9HJtW5YB47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AaNAUzc/G' + '\n' +
'sB0cNeURAoGBAOKtaVFa6z2F4Q+koMBXCt4m7dCJnaC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jhhvuV0OcN8lYbR/ZlYRJDUs6mHh/2BYSkdeaLKojXTxKR2bA4xQk5dtJCdoPf' + '\n' +
'0c15AlTgOYk2oNXP/azDICJYT/cdvIdUL9P4IoZthu1FjwG266GacEnNAoGBAMZn' + '\n' +
'1wRUXS1dbqemoc+g48wj5r3/qsIG8PsZ2Y8W+oYW7diNA5o6acc8YPEWE2RbJDbX' + '\n' +
'YEADBnRSdzzOdo0JEj4VbNZEtx6nQhBOOrtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
'eCzO1VcEeZtsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAI+fOD+nz6znirYSpRe5D' + '\n' +
'tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DjXrT09QmVAMciNEnc/sZBjiNzFf525wv' + '\n' +
'wZP/bPZMVYKtbsaVkdlcNJranHGUrkzswbxSRzmBQ5/YmCWrDAuYcnhEqmMWcuU9' + '\n' +
'8jiS13JP9hOXlHDyIBYDhV0CgYBV6TznuQgnzp9NpQ/H8ijxilItz3lHTu4mLMlR' + '\n' +
'9mdAjMkszdLTg5uuE+z+N8rp17VUseoRjb3LvLG4+MXIyDbH/0sDdPm+IjqvCNDR' + '\n' +
'spmh9MgBh0JbsbWaZK0s9/qrI/FcSLZ04JLsfRmTPU/Y5y8/dHjYO6fDQhp44RZF' + '\n' +
'iCqNxQKBgHf7KZIOKgV4YNyphk1UYWHNz8YY5o7WtaQ51Q+kIbU8PRd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjxi8GKQzE/IdkQqiFmB1yEjjRHQ81WS+K5NnjN1t0IEscJqOAwv9s' + '\n' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWxRHEYdegqU48f+qxUcJj9R' + '\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 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***);**

myPKEModule	The name of the Public Key Encryption From LFN (Logical FileName) 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 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::private scope

pubKey :=  RECORD
  STRING  Key;
END;

dPubKey :=  DATASET([{
'-----BEGIN PUBLIC KEY-----' + '\n' +
'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KMnWRAof' + '\n' +
'od+3AUS/IDngT39j3Iovv9aI2N8g4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeE' + '\n' +
'BHqlMDydw9aHOQG17CB30GYsw3Lf8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoy' + '\n' +
'hIR9MexCldF+3WM/X0IX0ApSs7kuVPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv' + '\n' +
'/oKj6q7kInEIvhLiGfcm3bpTzWQ66zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3' + '\n' +
'J6Tk4NY3NySWzE/2/ZOWxZdR79XC+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0' + '\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' +
'MIIEowIBAAKCAQEAr64RncTp5pV0KMnWRAofod+3AUS/IDngT39j3Iovv9aI2N8g' + '\n' +
'4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeEBHqlMDydw9aHOQG17CB30GYsw3Lf' + '\n' +
'8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoyhIR9MexCldF+3WM/X0IX0ApSs7ku' + '\n' +
'VPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv/oKj6q7kInEIvhLiGfcm3bpTzWQ6' + '\n' +
'6zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3J6Tk4NY3NySWzE/2/ZOWxZdR79XC' + '\n' +
'+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0bwIDAQABAoIBAQCnGAtNYkOOu8wW' + '\n' +
'F5Oid3aKwnwPytF211WQh3v2AcFU17qle+SMRi+ykBL6+u5RU5qH+HSc9Jm31AjW' + '\n' +
'V1yPrdYVZInFjYIJCPzorcXY5zDOmMAuzg5PBVV7VhUA0a5GZck6FC8AilDUcEom' + '\n' +
'GCK6Ul8mR9XELBFQ6keeTo2yDu0TQ4oBXrPBMN61uMHCxh2tDb2yvl8Zz+EllADG' + '\n' +
'70pztRWNOrCzrC+ARlmmDfYOUgVFtZin53jq6O6ullPLzhkm3/+QFRGYWsFgQB6J' + '\n' +
'Z9HJtW5YB47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AaNAUzc/G' + '\n' +
'sB0cNeURAoGBAOKtaVFa6z2F4Q+koMBXCt4m7dCJnaC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jhhvuV0OcN8lYbR/ZlYRJDUs6mHh/2BYSkdeaLKojXTxKR2bA4xQk5dtJCdoPf' + '\n' +
'0c15AlTgOYk2oNXP/azDICJYT/cdvIdUL9P4IoZthu1FjwG266GacEnNAoGBAMZn' + '\n' +
'1wRUXS1dbqemoc+g48wj5r3/qsIG8PsZ2Y8W+oYW7diNA5o6acc8YPEWE2RbJDbX' + '\n' +
'YEADBnRSdzzOdo0JEj4VbNZEtx6nQhBOOrtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
```

```
'eCzO1VcEeZtsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAI+fOD+nz6znirYSpRe5D' + '\n' +
'tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DjXrT09QmVAMciNEnc/sZBjiNzFf525wv' + '\n' +
'wZP/bPZMVYKtbsaVkdlcNJranHGUrkzswbxSRzmBQ5/YmCWrDAuYcnhEqmMWcuU9' + '\n' +
'8jiS13JP9hOXlHDyIBYDhV0CgYBV6TznuQgnzp9NpQ/H8ijxilItz3lHTu4mLMlR' + '\n' +
'9mdAjMkszdLTg5uuE+z+N8rp17VUseoRjb3LvLG4+MXIyDbH/0sDdPm+IjqvCNDR' + '\n' +
'spmh9MgBh0JbsbWaZK0s9/qrI/FcSLZ04JLsfRmTPU/Y5y8/dHjYO6fDQhp44RZF' + '\n' +
'iCqNxQKBgHf7KZIOKgV4YNyphk1UYWHNz8YY5o7WtaQ51Q+kIbU8PRd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjxi8GKQzE/IdkQqiFmB1yEjjRHQ81WS+K5NnjN1t0IEscJqOAwv9s' + '\n' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWxRHEYdegqU48f+qxUcJj9R' + '\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 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 FileName) 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-----' + '\n' +
'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KMnWRAof' + '\n' +
'od+3AUS/IDngT39j3Iovv9aI2N8g4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeE' + '\n' +
'BHqlMDydw9aHOQG17CB30GYsw3Lf8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoy' + '\n' +
'hIR9MexCldF+3WM/X0IX0ApSs7kuVPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv' + '\n' +
'/oKj6q7kInEIvhLiGfcm3bpTzWQ66zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3' + '\n' +
'J6Tk4NY3NySWzE/2/ZOWxZdR79XC+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0' + '\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' +
'MIIEowIBAAKCAQEAr64RncTp5pV0KMnWRAofod+3AUS/IDngT39j3Iovv9aI2N8g' + '\n' +
'4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeEBHqlMDydw9aHOQG17CB30GYsw3Lf' + '\n' +
'8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoyhIR9MexCldF+3WM/X0IX0ApSs7ku' + '\n' +
'VPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv/oKj6q7kInEIvhLiGfcm3bpTzWQ6' + '\n' +
'6zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3J6Tk4NY3NySWzE/2/ZOWxZdR79XC' + '\n' +
'+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0bwIDAQABAoIBAQCnGAtNYkOOu8wW' + '\n' +
'F5Oid3aKwnwPytF211WQh3v2AcFU17qle+SMRi+ykBL6+u5RU5qH+HSc9Jm31AjW' + '\n' +
'V1yPrdYVZInFjYIJCPzorcXY5zDOmMAuzg5PBVV7VhUA0a5GZck6FC8AilDUcEom' + '\n' +
'GCK6Ul8mR9XELBFQ6keeTo2yDu0TQ4oBXrPBMN61uMHCxh2tDb2yvl8Zz+EllADG' + '\n' +
'70pztRWNOrCzrC+ARlmmDfYOUgVFtZin53jq6O6ullPLzhkm3/+QFRGYWsFgQB6J' + '\n' +
'Z9HJtW5YB47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AaNAUzc/G' + '\n' +
'sB0cNeURAoGBAOKtaVFa6z2F4Q+koMBXCt4m7dCJnaC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jhhvuV0OcN8lYbR/ZlYRJDUs6mHh/2BYSkdeaLKojXTxKR2bA4xQk5dtJCdoPf' + '\n' +
'0c15AlTgOYk2oNXP/azDICJYT/cdvIdUL9P4IoZthu1FjwG266GacEnNAoGBAMZn' + '\n' +
'1wRUXS1dbqemoc+g48wj5r3/qsIG8PsZ2Y8W+oYW7diNA5o6acc8YPEWE2RbJDbX' + '\n' +
```

```
'YEADBnRSdzzOdo0JEj4VbNZEtx6nQhBOOrtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
'eCzO1VcEeZtsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAI+fOD+nz6znirYSpRe5D' + '\n' +
'tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DjXrT09QmVAMciNEnc/sZBjiNzFf525wv' + '\n' +
'wZP/bPZMVYKtbsaVkdlcNJranHGUrkzswbxSRzmBQ5/YmCWrDAuYcnhEqmMWcuU9' + '\n' +
'8jiS13JP9hOXlHDyIBYDhV0CgYBV6TznuQgnzp9NpQ/H8ijxilItz3lHTu4mLMlR' + '\n' +
'9mdAjMkszdLTg5uuE+z+N8rp17VUseoRjb3LvLG4+MXIyDbH/0sDdPm+IjqvCNDR' + '\n' +
'spmh9MgBh0JbsbWaZK0s9/qrI/FcSLZ04JLsfRmTPU/Y5y8/dHjYO6fDQhp44RZF' + '\n' +
'iCqNxQKBgHf7KZIOKgV4YNyphk1UYWHNz8YY5o7WtaQ51Q+kIbU8PRd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjxi8GKQzE/IdkQqiFmB1yEjjRHQ81WS+K5NnjN1t0IEscJqOAwv9s' + '\n' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWxRHEYdegqU48f+qxUcJj9R' + '\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 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***);**

myPKEModule	The name of the Public Key Encryption From LFN (Logical FileName) module structure
inputData	The data to sign in DATA format
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 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::private scope

pubKey :=  RECORD
  STRING  Key;
END;

dPubKey :=  DATASET([{
'-----BEGIN PUBLIC KEY-----' + '\n' +
'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KMnWRAof' + '\n' +
'od+3AUS/IDngT39j3Iovv9aI2N8g4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeE' + '\n' +
'BHqlMDydw9aHOQG17CB30GYsw3Lf8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoy' + '\n' +
'hIR9MexCldF+3WM/X0IX0ApSs7kuVPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv' + '\n' +
'/oKj6q7kInEIvhLiGfcm3bpTzWQ66zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3' + '\n' +
'J6Tk4NY3NySWzE/2/ZOWxZdR79XC+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0' + '\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' +
'MIIEowIBAAKCAQEAr64RncTp5pV0KMnWRAofod+3AUS/IDngT39j3Iovv9aI2N8g' + '\n' +
'4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeEBHqlMDydw9aHOQG17CB30GYsw3Lf' + '\n' +
'8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoyhIR9MexCldF+3WM/X0IX0ApSs7ku' + '\n' +
'VPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv/oKj6q7kInEIvhLiGfcm3bpTzWQ6' + '\n' +
'6zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3J6Tk4NY3NySWzE/2/ZOWxZdR79XC' + '\n' +
'+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0bwIDAQABAoIBAQCnGAtNYkOOu8wW' + '\n' +
'F5Oid3aKwnwPytF211WQh3v2AcFU17qle+SMRi+ykBL6+u5RU5qH+HSc9Jm31AjW' + '\n' +
'V1yPrdYVZInFjYIJCPzorcXY5zDOmMAuzg5PBVV7VhUA0a5GZck6FC8AilDUcEom' + '\n' +
'GCK6Ul8mR9XELBFQ6keeTo2yDu0TQ4oBXrPBMN61uMHCxh2tDb2yvl8Zz+EllADG' + '\n' +
'70pztRWNOrCzrC+ARlmmDfYOUgVFtZin53jq6O6ullPLzhkm3/+QFRGYWsFgQB6J' + '\n' +
'Z9HJtW5YB47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AaNAUzc/G' + '\n' +
'sB0cNeURAoGBAOKtaVFa6z2F4Q+koMBXCt4m7dCJnaC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jhhvuV0OcN8lYbR/ZlYRJDUs6mHh/2BYSkdeaLKojXTxKR2bA4xQk5dtJCdoPf' + '\n' +
'0c15AlTgOYk2oNXP/azDICJYT/cdvIdUL9P4IoZthu1FjwG266GacEnNAoGBAMZn' + '\n' +
'1wRUXS1dbqemoc+g48wj5r3/qsIG8PsZ2Y8W+oYW7diNA5o6acc8YPEWE2RbJDbX' + '\n' +
'YEADBnRSdzzOdo0JEj4VbNZEtx6nQhBOOrtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
```

```
'eCzO1VcEeZtsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAI+fOD+nz6znirYSpRe5D' + '\n' +
'tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DjXrT09QmVAMciNEnc/sZBjiNzFf525wv' + '\n' +
'wZP/bPZMVYKtbsaVkdlcNJranHGUrkzswbxSRzmBQ5/YmCWrDAuYcnhEqmMWcuU9' + '\n' +
'8jiS13JP9hOXlHDyIBYDhV0CgYBV6TznuQgnzp9NpQ/H8ijxilItz3lHTu4mLMlR' + '\n' +
'9mdAjMkszdLTg5uuE+z+N8rp17VUseoRjb3LvLG4+MXIyDbH/0sDdPm+IjqvCNDR' + '\n' +
'spmh9MgBh0JbsbWaZK0s9/qrI/FcSLZ04JLsfRmTPU/Y5y8/dHjYO6fDQhp44RZF' + '\n' +
'iCqNxQKBgHf7KZIOKgV4YNyphk1UYWHNz8YY5o7WtaQ51Q+kIbU8PRd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjxi8GKQzE/IdkQqiFmB1yEjjRHQ81WS+K5NnjN1t0IEscJqOAwv9s' + '\n' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWxRHEYdegqU48f+qxUcJj9R' + '\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'));
```

VerifySignature (PKE From LFN)

myPKEModule.**VerifySignature(***signature, signedData***);**

myPKEModule	The name of the Public Key Encryption From LFN (Logical FileName) 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 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::private scope

pubKey :=  RECORD
  STRING  Key;
END;

dPubKey :=  DATASET([{
'-----BEGIN PUBLIC KEY-----' + '\n' +
'MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAr64RncTp5pV0KMnWRAof' + '\n' +
'od+3AUS/IDngT39j3Iovv9aI2N8g4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeE' + '\n' +
'BHqlMDydw9aHOQG17CB30GYsw3Lf8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoy' + '\n' +
'hIR9MexCldF+3WM/X0IX0ApSs7kuVPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv' + '\n' +
'/oKj6q7kInEIvhLiGfcm3bpTzWQ66zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3' + '\n' +
'J6Tk4NY3NySWzE/2/ZOWxZdR79XC+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0' + '\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' +
'MIIEowIBAAKCAQEAr64RncTp5pV0KMnWRAofod+3AUS/IDngT39j3Iovv9aI2N8g' + '\n' +
'4W5ipqhKftRESmzQ6I/TiUQcmi42soUXmCeEBHqlMDydw9aHOQG17CB30GYsw3Lf' + '\n' +
'8iZo7RC7ocQE3OcRzH0eBkOryW6X3efWnMoyhIR9MexCldF+3WM/X0IX0ApSs7ku' + '\n' +
'VPVG4Yj202+1FVO/XNwjMukJG5ASuxpYAQvv/oKj6q7kInEIvhLiGfcm3bpTzWQ6' + '\n' +
'6zVz3z/huLbEXEy5oj2fQaC5E3s5mdpk/CW3J6Tk4NY3NySWzE/2/ZOWxZdR79XC' + '\n' +
'+goNL6v/5gPI8B/a3Z8OeM2PfSZwPMnVuvU0bwIDAQABAoIBAQCnGAtNYkOOu8wW' + '\n' +
'F5Oid3aKwnwPytF211WQh3v2AcFU17qle+SMRi+ykBL6+u5RU5qH+HSc9Jm31AjW' + '\n' +
'V1yPrdYVZInFjYIJCPzorcXY5zDOmMAuzg5PBVV7VhUA0a5GZck6FC8AilDUcEom' + '\n' +
'GCK6Ul8mR9XELBFQ6keeTo2yDu0TQ4oBXrPBMN61uMHCxh2tDb2yvl8Zz+EllADG' + '\n' +
'70pztRWNOrCzrC+ARlmmDfYOUgVFtZin53jq6O6ullPLzhkm3/+QFRGYWsFgQB6J' + '\n' +
'Z9HJtW5YB47RT5RbLHKXeMc6IJW+d+5HrzgTdK79P7wAZk8JCIDyHe2AaNAUzc/G' + '\n' +
'sB0cNeURAoGBAOKtaVFa6z2F4Q+koMBXCt4m7dCJnaC+qthF249uEOIBeF3ds9Fq' + '\n' +
'f0jhhvuV0OcN8lYbR/ZlYRJDUs6mHh/2BYSkdeaLKojXTxKR2bA4xQk5dtJCdoPf' + '\n' +
'0c15AlTgOYk2oNXP/azDICJYT/cdvIdUL9P4IoZthu1FjwG266GacEnNAoGBAMZn' + '\n' +
'1wRUXS1dbqemoc+g48wj5r3/qsIG8PsZ2Y8W+oYW7diNA5o6acc8YPEWE2RbJDbX' + '\n' +
```

```
'YEADBnRSdzzOdo0JEj4VbNZEtx6nQhBOOrtYKnnqHVI/XOz3VVu6kedUKdBR87KC' + '\n' +
'eCzO1VcEeZtsTHuLO4t7NmdHGqNxTV+jLvzBoQsrAoGAI+fOD+nz6znirYSpRe5D' + '\n' +
'tW67KtYxlr28+CcQoUaQ/Au5kjzE9/4DjXrT09QmVAMciNEnc/sZBjiNzFf525wv' + '\n' +
'wZP/bPZMVYKtbsaVkdlcNJranHGUrkzswbxSRzmBQ5/YmCWrDAuYcnhEqmMWcuU9' + '\n' +
'8jiS13JP9hOXlHDyIBYDhV0CgYBV6TznuQgnzp9NpQ/H8ijxilItz3lHTu4mLMlR' + '\n' +
'9mdAjMkszdLTg5uuE+z+N8rp17VUseoRjb3LvLG4+MXIyDbH/0sDdPm+IjqvCNDR' + '\n' +
'spmh9MgBh0JbsbWaZK0s9/qrI/FcSLZ04JLsfRmTPU/Y5y8/dHjYO6fDQhp44RZF' + '\n' +
'iCqNxQKBgHf7KZIOKgV4YNyphk1UYWHNz8YY5o7WtaQ51Q+kIbU8PRd9rqJLZyk2' + '\n' +
'tKf8e6z+wtKjxi8GKQzE/IdkQqiFmB1yEjjRHQ81WS+K5NnjN1t0IEscJqOAwv9s' + '\n' +
'iIhG5ueb6xoj/N0LuXa8loUT5aChKWxRHEYdegqU48f+qxUcJj9R' + '\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

Date_rec	A RECORD structure containing three fields, and INTEGER2 year, an UNSIGNED1 month, and an UNSIGNED1 day.
Date_t	An UNSIGNED4 containing a date value in YYYYMMDD format.
Days_t	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).

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

```
    // A record stucture 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

Time_rec	A RECORD structure containing three fields, and INTEGER1 hour, an UNSIGNED1 minute, and an UNSIGNED1 second.
Time_t	An UNSIGNED3 holding a time of day in the decimal form HHMMDD.
Seconds_t	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).
DateTime_rec	A RECORD structure containing both a Date_rec and a Time_rec
Timestamp_t	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).

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

```
// 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* **)**

date	A date value in the Date_t format.
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:

```
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* **)**

time	A time value in the Time_ format.
Return:	Hour returns an INTEGER value representing the hour in the range of 0-23.

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

Example:

```
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* **)**

time	A time value in the Time_ format.
Return:	Minute returns an INTEGER value representing the minute in the range of 0-59.

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

Example:

```
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* **)**

time	A time value in the Time_ format.
Return:	Second returns an INTEGER value representing the second in the range of 0-59.

The **Second** function returns the second from the *time* value.

Example:

```
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:

```
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* **)**

hour	An INTEGER1 hour value in the range 0 to 23.
minute	An UNSIGNED1 minute value in the range 0 to 59.
second	An UNSIGNED1 second value in the range 0 to 59.
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:

```
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* **)**

year	A year value in the INTEGER2 format.
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:

```
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*] **)**

date	A date value in the Date_t format.
yearLowerBound	The minimum acceptable year. Optional; defaults to 1800.
yearUpperBound	The maximum acceptable year. Optional; defaults to 2100.
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:

```
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* **)**

time	A time value in the Time_t format.
Return:	IsValidTime returns a BOOLEAN value.

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:

```
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* **)**

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:	FromGregorianYMD returns an UNSIGNED4 value.

The **FromGregorianYMD** function returns a Days_t value from the *year*, *month*, and *day* parameters representing the number 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:

```
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* **)**

days	A year value in the Days_t format.
Return:	ToGregorianYMD returns separate values for Year, Month, and Day.

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

Example:

```
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.Date.FromStringToDate(*date_text, format* **)**

date_text	The string to be converted
format	The format of the input string. See strftime documentation for details (http://strftime.org/)
return	The date that was matched in the string. Returns 0 if failed to match or if the date components match but the result is an invalid date.

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:

```
IMPORT STD;

D1 := STD.Date.FromStringToDate('19720607', '%Y%m%d');
    //D1 contains 19720607
D2 := STD.Date.FromStringToDate('19720007', '%Y%m%d');
    //D2 contains 0
D3 := STD.Date.FromStringToDate('4/29/1974', '%m/%d/%Y');
    //D3 contains 19740429
D4:= STD.Date.FromStringToDate('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 YYYYMMDD 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***])**

in_local_time	TRUE if the returned value should be local to the cluster computing the date, FALSE for UTC. Optional, defaults to FALSE.
Return:	Today returns a Date_t representing the current date.

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:

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

CurrentTime

STD.Date.CurrentTime ([*in_local_time***])**

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

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:

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

DayOfWeek

STD.Date.DayOfWeek(date)

date	A date value in the Date_t format.
Return:	DayofWeek returns an INTEGER value representing the day of the week, where 1 = Sunday.

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:

```
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)

fromDate	The first date value in Date_t format.
toDate	The last date value in Date_t format.
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:

```
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)

fromDate	The first date value in Date_t format.
toDate	The last date value in Date_t format.
month_ends_equal	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
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:

```
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*] **)**

date	A date value in the Date_t format.
year_delta	The minimum acceptable year. Optional; defaults to zero.
month_delta	The minimum acceptable year. Optional; defaults to zero.
day_delta	The maximum acceptable year. Optional; defaults to zero.
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:

```
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*] **)**

date	A date value in the Date_t format.
year_delta	The minimum acceptable year. Optional; defaults to zero.
month_delta	The minimum acceptable year. Optional; defaults to zero.
day_delta	The maximum acceptable year. Optional; defaults to zero.
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.

As an example, Jan. 31, 2014 + 1 month results in Feb. 28, 2014; Jan. 31, 2014 + 1 month + 1 day results in Mar. 1, 2014.

Example:

```
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.
startingDay-OfWeek	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:

```
IMPORT STD;
startDate := STD.Date.Today();
weekNum := STD.Date.MonthWeekNumFromDate(startDate,2);
weekNum;
```

See Also: YearWeekNumFromDate

YearWeekNumFromDate

STD.Date.YearWeekNumFromDate(date, startingDayOfWeek)

date	The date (in Date_t format) for which to compute the week number.
startingDay-OfWeek	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 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:

```
IMPORT STD;
startDate := STD.Date.Today();
weekNum := STD.Date.YearWeekNumFromDate(startDate,2);
weekNum;
```

See Also: MonthWeekNumFromDate

TimestampToString

STD.Date.TimestampToString (*timestamp, format***)**

timestamp	An INTEGER8 holding the number of microseconds since epoch (January 1, 1970 UTC)
format	OPTIONAL. The format of the string to return. See strftime documentation for details (http://strftime.org/). If omitted, it defaults to '%Y-%m-%dT%H:%M:%S.%@' which is YYYY-MM-DDTHH:MM:SS.ssssss.
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)

Millisecond fractions are truncated from microseconds when necessary.

The maximum length of the resulting string is 255 characters.

Example:

```
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* **)**

location	REQUIRED. The name of the location to search for; must be a non-empty uppercase string.
Returns:	A new DATASET(STRING5 tzAbbrev, INTEGER4 secondsOffset) containing the records found for the given location.

The **STD.Date.TimeZone.TZDataForLocation** function returns the time zone records for a given location.

Example:

```
IMPORT STD;
STD.Date.TimeZone.TZDataForLocation('ASIA');
```

See Also: FindTZData

FindTZData

STD.Date.TimeZone.FindTZData(*timeZoneAbbrev,* [*location*]**)**

timeZoneAbbrev	REQUIRED. The time zone abbreviation to search for; must be a non-empty uppercase string.
location	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.

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`
	`  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*] **)**

fromTimeZone-Abbrev	REQUIRED. The time zone abbreviation designated as the starting point; must be a non-empty uppercase string.
toTimeZoneAb-brev	REQUIRED. The time zone abbreviation designated as the ending point; 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 fromTimeZone-Abbrev is used.
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.
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:

```
MPORT STD;
STD.Date.TimeZone.SecondsBetweenTZ('CST','IST','NORTH AMERICA','');
```

See Also: AdjustTimeTZ

AdjustTimeTZ

STD.Date.TimeZone.AdjustTimeTZ(*time,fromTimeZoneAbbrev, toTimeZoneAbbrev,* [*fromLocation*,] [*toLocation*] **)**

time	REQUIRED. The time value (in Time_t format) to adjust.
fromTimeZone-Abbrev	REQUIRED. The time zone abbreviation that the *time* value is assumed to be within; must be a non-empty uppercase string.
toTimeZoneAb-brev	REQUIRED. The time zone abbreviation designated as the ending point; 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 fromTimeZone-Abbrev is used.
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.
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:

```
IMPORT STD;
STD.Date.TimeZone.AdjustTimeTZ(205246,'CST','IST','NORTH AMERICA','');
```

See Also: SecondsBetweenTZ

ToLocalTime

STD.Date.TimeZone.ToLocalTime(*utcTime, toTimeZoneAbbrev,* [*toLocation*] **)**

utcTime	REQUIRED. The UTC time value (in Time_t format) to adjust.
toTimeZoneAbbrev	REQUIRED. The time zone abbreviation designated as the ending point; must be a non-empty uppercase string.
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.
Returns:	The given UTC time value (in Time_t format) adjusted to the time zone defined by toTimeZoneAbbrev and toLocation; 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:

```
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.
fromTimeZone-Abbrev	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 fromTimeZone-Abbrev 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(*infile, timeZoneAbbrevField, newOffsetField,* [*fromLocationField*,] [*toTimeZoneAbbrev*,] [*toLocation*] **)**

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 // If <newOffsetField>_is_valid is FALSE // then <newOffsetField> will be zero.`
	`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:

```
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,
                                                toTimeZoneAbbrev := '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`
	`                                  // If <newOffsetField>_is_valid is FALSE`
	`                                  // then <newOffsetField> will have the same`
	`                                  // value as timeField.`
	`STRING5 <newOffsetField>_tz       // The value of toTimeZoneAbbrev`
	`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:

```
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

ISODayOfWeekFromDate

STD.Date.ISODayOfWeekFromDate(*date***)**

date	A date value in the Date_t format.
Return:	ISODayOfWeekFromDate returns an INTEGER value representing the ISO day of the week in the range 1–7, where 1 = Monday.

The **ISODayOfWeekFromDate** function returns a number representing the ISO-8601 day of the week for the given date. The date must be in the Gregorian calendar after the year 1600.

Example:

```
IMPORT STD;
DOW1 := STD.Date.ISODayOfWeekFromDate(STD.Date.Today());
      // DOW1 contains the ISO day of the week for today's date (where Monday = 1)
OUTPUT(DOW1);
```

ISOIsLongYear

STD.Date.ISOIsLongYear(*year* **)**

year	An INTEGER2 year value.
Return:	ISOIsLongYear returns TRUE if the year has 53 ISO weeks, FALSE otherwise.

The **ISOIsLongYear** function returns TRUE if the specified year is a long year (i.e., has 53 ISO weeks), or FALSE if it has 52 weeks.

Example:

```
IMPORT STD;
Y := 2020;
IsLong := STD.Date.ISOIsLongYear(Y);
         // IsLong is TRUE for 2020 (it is a long year)
OUTPUT(IsLong);
```

ISOWeeksFromDate

STD.Date.ISOWeeksFromDate(*date* **)**

date	A date value in the Date_t format.
Return:	An INTEGER value in the range 52-53: 52 for short years and 53 for long years.

The **ISOWeeksFromDate** function returns the number of ISO-8601 weeks in the ISO week-numbering year associated with the given date (52 or 53).

Example:

```
IMPORT STD;
D := 20201231;
Weeks := STD.Date.ISOWeeksFromDate(D);
      // Weeks is 53 for 2020-12-31
OUTPUT(Weeks);
```

ISORawWeekNumForDate

STD.Date.ISORawWeekNumForDate(*date* **)**

date	A date value in the Date_t format.
Return:	ISORawWeekNumForDate returns an INTEGER value from 1 to 53 representing the raw ISO week number for the date.

The **ISORawWeekNumForDate** function returns the raw ISO-8601 week number (1–53) for the given date.

Example:

```
IMPORT STD;
D := 20200615;
RawWeek := STD.Date.ISORawWeekNumForDate(D);
      // RawWeek is the ISO week number for June 15, 2020 which is 25
OUTPUT(RawWeek);
```

ISOWeekNumWeekDayAndYearFrom-Date

STD.Date.ISOWeekNumWeekDayAndYearFromDate(*date* **)**

date	A date value in the Date_t format.
Return:	ISOWeekNumWeekDayAndYearFromDate returns a module with *weekNumber* (1–53), *year* (the ISO week-numbering year), and *weekDay* (1–7, where 1 = Monday).

The **ISOWeekNumWeekDayAndYearFromDate** function returns a module containing the ISO week number, the ISO week-numbering year, and the ISO day of week for the given date. This is an ISO-8601 implementation of computing week numbers ("week dates").

Example:

```
IMPORT STD;
D := 20210101;
W := STD.Date.ISOWeekNumWeekDayAndYearFromDate(D);
      // W.weekNumber, W.year, W.weekDay for January 1, 2021
OUTPUT(W);
```

ISOWeekDate

STD.Date.ISOWeekDate(*date***[,** *extended***])**

date	A date value in the Date_t format.
extended	Optional BOOLEAN. If TRUE, returns the ISO week date in extended form (e.g., 2018-W23-7). If FALSE (default), returns compact form (e.g., 2018W237).
Return:	ISOWeekDate returns a STRING representing the ISO-8601 week date for the given date.

The **ISOWeekDate** function returns the ISO-8601 week date for the given date, in either extended or compact form. This is an ISO-8601 implementation of computing week numbers ("week dates").

Example:

```
IMPORT STD;
D := 20211231;
WeekDate := STD.Date.ISOWeekDate(D, TRUE);
      // WeekDate is "2021-W52-5" for December 31, 2021
OUTPUT(WeekDate);
```

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( )

Return:	Nodes returns an UNSIGNED INTEGER4 value.

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:

```
A := STD.System.Thorlib.Nodes();
```

LogicalToPhysical

STD.System.Thorlib.LogicalToPhysical (*filename* [, *createflag*])

filename	A null-terminated string containing the logical name of the file.
createflag	A boolean value indicating whether to create the *filename*. If omitted, the default is FALSE.
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:

```
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.

Deprecated in containerized environments. In containerized HPCC Systems deployments, compute nodes are ephemeral and the node group name has no stable meaning. Storage is decoupled from compute and is referenced via data planes (for example, **'data'**). In containerized deployments this function returns a blank string (**''**), which is interpreted as the default data plane. New ECL code targeting containerized HPCC should use an explicit data plane name (for example, **'data'**) rather than calling this function.

Example:

```
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:

```
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 the target cluster for a workunit.

Example:

```
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:

```
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:

```
A := STD.System.Job.User();
```

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:

```
A := STD.System.Job.Platform();
```

LogString

STD.System.Job.LogString (*message*)

message	A string expression containing the text to place in the log file.
Return:	LogString returns an INTEGER value.

The **LogString** function outputs "USER:" followed by the *message* text to the eclagent or Roxie log file and returns the length of the text written to the file.

Example:

```
A := STD.System.Job.LogString('The text message to log');
 //places USER:The text message to log
 //in the log file
```

File Monitoring

MonitorFile

STD.File.MonitorFile(*event,* **[** *ip* **] ,** *filename,* **[** *,subdirs* **] [** *,shotcount* **] [** *,espserverIPport* **])**

dfuwuid **:= STD.File.fMonitorFile(** *event,* **[** *ip* **] ,** *filename,* **[** *,subdirs* **] [** *,shotcount* **] [** *,espserverIPport* **]);**

event	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.
ip	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 *filename*parameter contains a complete URL.
filename	A null-terminated string containing the full path to the file to monitor. This may contain wildcard characters (* and ?).
subdirs	Optional. A boolean value indicating whether to include files in sub-directories that match the wildcard mask when the *filename* contains wildcards. If omitted, the default is false.
shotcount	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.
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 recieve the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the monitoring job.
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:

```
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* **]);**

event	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.
filename	A null-terminated string containing the name of the logical file in the DFU to monitor.
shotcount	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.
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 recieve 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:

```
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

dbglog

STD.System.Log.dbglog (*text*)

text	A string containing the text to write.
Return:	dbglog does not return a value.

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*])

text	A string containing the text to write.
code	Optional. The code number to associate with the *text*. If omitted, the default is zero (0).
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:

```
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*])

text	A string containing the text to write.
code	Optional. The code number to associate with the *text*. If omitted, the default is zero (0).
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:

```
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*])

text	A string containing the text to write.
code	Optional. The code number to associate with the *text*. If omitted, the default is zero (0).
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:

```
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:

```
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:

```
IMPORT STD;
STD.System.Log.getLocalId();
```

generateGloballyUniqueID

STD.System.Log.generateGloballyUniqueID ()

Return:	generateGloballyUniqueID returns a globally unique identifier.

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'));
```

getElapsedMs

result **:= STD.System.Log.getElapsedMs ();**

Return:	getElapsedMs returns returns the elapsed time in milliseconds.

The **getElapsedMs** function returns the current elapsed query time (in ms) in Roxie.

This is the elapsed time when STD.System.Log.getElapsedMs() 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.getElapsedMs(), NAMED('Elapsed')); //returns total time elapsed
```

Auditing

Audit

STD.Audit.Audit(*type, message* **)**

type	A string constant containing the type of audit entry. Currently, only INFO is provided.
message	A string containing the audit entry text.
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');
```

Utilities

GetHostName

result **:= STD.System.Util.GetHostName (*ip*);**

ip	A null-terminated string containing the IP address of the remote machine.
Return:	GetHostName returns 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:

```
IP   := '10.150.254.6';

OUTPUT(STD.System.Util.GetHostName(IP));
```

ResolveHostName

result **:= STD.System.Util.ResolveHostName (*host*);**

host	A null-terminated string containing the DNS name of the remote machine.
Return:	ResolveHostName returns returns a VARSTRING (null-terminated) value.

The **ResolveHostName** function does a DNS lookup to return the ip address for the specified *host* name.

Example:

```
host := 'dataland_dali.br.seisint.com';
OUTPUT(STD.System.Util.ResolveHostName(host));
```

GetUniqueInteger

result **:= STD.System.Util.GetUniqueInteger ([** *dali* **]);**

dali	Optional. A null-terminated string containing the ip address of the remote dali to provide the number. If omitted, the default is local.
Return:	GetUniqueInteger returns returns an UNSIGNED8 value.

The **GetUniqueInteger** function returns a number that is unique across all the worker nodes of the specified *dali*.

Example:

```
IMPORT STD;

OUTPUT(STD.System.Util.GetUniqueInteger());
```

GetEspUrl

result **:= STD.File.GetEspUrl ([** *username, userPW* **]);**

username	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.
userPW	Optional. A STRING containing the password to use with the user cited in the *username* argument. If *username* is empty then this is ignored
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:

```
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:

```
IMPORT STD;
#IF(STD.System.Util.PlatformVersionCheck('8.2.0-1'))
  OUTPUT('Platform check TRUE');
#ELSE
  OUTPUT('Platform check FALSE');
#END
```

Debugging

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 Esaus sons; Eliphaz the son of '+
 'Adah the wife of Esau, Reuel the son of Bashemath 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 Esaus sons; Eliphaz the son of '+
 'Adah the wife of Esau, Reuel the son of Bashemath 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       := 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>
<namet>
 <name>Hamor</name>
</namet>
*/
```

Sleep

STD.System.Debug.Sleep (*duration*)

duration	An integer value specifying the length of the sleep period, in milliseconds.
Return:	Sleep does not return a value.

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:

```
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));
```

Email

SendEmail

STD.System.Email.SendEmail (*to, subject, body, attachment, mimietype, filename, mailServer, port, sender, cc, 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 (see the STD.System.Email.SendEmailAttachText() function).
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 **SendEmail** function sends an email message.

Example:

```
STD.System.Email.SendEmail( 'me@example.com', 'testing 1,2,3', 'this is a test message');
```

SendEmailAttachData

STD.System.Email.SendEmailAttachData (*to, subject, body, attachment, mimietype, 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:

```
DATA15 attachment :=  D'test attachment';

STD.System.Email.SendEmailAttachData( 'me@example.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, mimietype, filename, mailServer, port, sender, cc, 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 null-terminated string containing the text 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').
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 **SendEmailAttachText** function sends an email message with a text *attachment*.

Example:

```
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* **])**

wuid	A null-terminated string containing the WorkUnit IDentifier to locate.
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.
Return:	WorkunitExists returns a BOOLEAN value.

The **WorkunitExists** function returns whether the *wuid* exists.

Example:

```
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(256)};
  STRING10 state;
  STRING7 priority;
  STRING20 created;
```

```
  STRING20 modified;
  BOOLEAN online;
  BOOLEAN protected;
END;
```

Example:

```
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* **])**

app	The application name to set.
key	The name of the value to set.
value	The value to set.
overwrite	A boolean TRUE or FALSE flag indicating whether to allow the value to overwrite an existing value. Default is TRUE..
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:

```
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*)

year	An unsigned integer containing the year value.
month	An unsigned integer containing the month value.
day	An unsigned integer containing the day value.
hour	An unsigned integer containing the hour value.
minute	An unsigned integer containing the minute value.
Return:	WUIDonDate returns a VARSTRING value.

The **WUIDonDate** function returns a valid WorkUnit IDentifier for a workunit that meets the passed parameters.

Example:

```
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*)

wuid	A null-terminated string containing the WorkUnit IDentifier.
Return:	WorkunitTimeStamps returns a DATASET value.

The **WorkunitTimeStamps** function returns a DATASET with this format:

```
EXPORT TimeStampRecord := 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:

```
OUTPUT(STD.System.Workunit.WorkunitTimeStamps('W20240801-122755'));

/* produces output like this:
'workunit     ','Created ','2024-08-01T16:28:20Z','              '
'workunit     ','Modified','2024-08-01T16:32:47Z','              '
'EclServer    ','Compiled','2024-08-01T16:28:20Z','172.31.4.17'
'EclAgent     ','Started ','2024-08-01T16:32:35Z','172.31.4.17'
'Thor - graph1','Finished','2024-08-01T16:32:47Z','172.31.4.17'
'Thor - graph1','Started ','2024-08-01T16:32:13Z','172.31.4.17'
'EclAgent     ','Finished','2024-08-01T16:33:09Z','172.31.4.17'
*/
```

WorkunitMessages

STD.System.Workunit.WorkunitMessages (*wuid*)

wuid	A null-terminated string containing the WorkUnit IDentifier.
Return:	WorkunitMessages returns a DATASET value.

The **WorkunitMessages** function returns a DATASET with this format:

```
EXPORT WsMessage_v2 := RECORD
  UNSIGNED4 severity;
  INTEGER4  code;
  STRING32  location;
  UNSIGNED4 row;
  UNSIGNED4 col;
  STRING16  source;
  STRING20  time;
  UNSIGNED4 priority;
  REAL8     cost;
  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.

Note: The DATASET structure returned by WorkunitMessages added two fields (`priority` and `cost`) in version 9.10.0.

Example:

```
IMPORT STD;
OUTPUT(STD.System.Workunit.WorkunitMessages('W20250602-164946'));
```

WorkunitFilesRead

STD.System.Workunit.WorkunitFilesRead (*wuid*)

wuid	A null-terminated string containing the WorkUnit IDentifier.
Return:	WorkunitFilesRead returns a DATASET value.

The **WorkunitFilesRead** function returns a DATASET with this format:

```
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:

```
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*)

wuid	A null-terminated string containing the WorkUnit IDentifier.
Return:	WorkunitFilesWritten returns a DATASET value.

The **WorkunitFilesWritten** function returns a DATASET with this format:

```
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:

```
OUTPUT(STD.System.Workunit.WorkunitFilesWritten('W20070308-164946'));
/* produces results that look like this
'rttest::testfetch','graph1','thor','0'
*/
```

WorkunitTimings

STD.System.Workunit.WorkunitTimings (*wuid*)

wuid	A null-terminated string containing the WorkUnit IDentifier.
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:

```
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'
*/
```

WorkunitStatistics

STD.System.Workunit.WorkunitStatistics (*wuid* [, *includeActivities*] [, *_filter*])

wuid	A null-terminated string containing the WorkUnit IDentifier.
includeActivities	Optional. A Boolean TRUE or FALSE flag specifying whether activity-level statistics are included. If omitted, the default is FALSE.
_filter	Optional. A null-terminated string containing a statistics filter expression. If omitted, the default is an empty string.
Return:	WorkunitStatistics returns a DATASET value.

The **WorkunitStatistics** function returns a DATASET with this format:

```
EXPORT WsStatistic := RECORD
  UNSIGNED8 value;
  UNSIGNED8 count;
  UNSIGNED8 maxValue;
  STRING creatorType;
  STRING creator;
  STRING scopeType;
  STRING scope;
  STRING name;
  STRING description;
  STRING unit;
END;
```

Each record in the returned dataset specifies a statistic for the workunit.

The optional `_filter` parameter is a comma-separated list of `option[value]` clauses that can be used to limit which scopes are searched and which statistics are returned.

The supported filter clauses are:

- `scope[<scope-id>]`, `stype[<scope-type>]`, or `id[<id>]` to select matching scopes. These clauses may be repeated, but `scope`, `stype`, and `id` should not be mixed in the same filter.
- `depth[n]`, `depth[low,high]`, or `depth[low..high]` to restrict the depth searched for matches.
- `source[global|stats|statistics|graph|workflow|exception|all]` to choose which workunit sources are searched.
- `where[<stat>]`, `where[<stat>(=|<|<=|>|>=)<value>]`, or `where[<stat>=<low>..<high>]` to filter by statistic existence or value range.
- `matched[true|false]`, `nested[<depth>|all]`, and `include[<scope-type>]` or `include-type[<scope-type>]` to control which matching scopes are returned.
- `properties[statistics|hints|attributes|notes|scope|all]` or `props[...]` to choose categories of properties to return.
- `statistic[<statistic-kind>|none|all]` or `stat[...]`, `attribute[<attribute-name>|none|all]` or `attr[...]`, `hint[<hint-name>]`, `property[<name>]` or `prop[...]`, `measure[<measure>]`, and `version[<version>]` to control which details are returned for each scope.

Common scope type values include `global`, `graph`, `subgraph`, `activity`, `operation`, `workflow`, `file`, and `child`. Statistic names use the internal statistic kind names, for example `TimeElapsed` and `NumRowsProcessed`.

Example:

```
OUTPUT(STD.System.Workunit.WorkunitStatistics('W20260330-164946'));
OUTPUT(STD.System.Workunit.WorkunitStatistics('W20260330-164946', TRUE));
OUTPUT(STD.System.Workunit.WorkunitStatistics('W20260330-164946', FALSE,
       'source[stats],where[TimeElapsed>1000000]'));
OUTPUT(STD.System.Workunit.WorkunitStatistics('W20260330-164946', TRUE,
       'stype[activity],where[NumRowsProcessed>0],stat[NumRowsProcessed],
       stat[TimeElapsed]'));
```

BLAS Support

This section provides support tor 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

value_t	REAL8
dimension_t	UNSIGNED4
matrix_t	SET OF REAL8
Triangle	ENUM(UNSIGNED1, Upper=1, Lower=2)
Diagonal	ENUM(UNSIGNED1, UnitTri=1, NotUnitTri=2)
Side	ENUM(UNSIGNED1, Ax=1, xA=2)

Types for the Block Basic Linear Algebra Sub-programs support

ICellFunc

STD.BLAS.ICellFunc(*v, r* **,** *c);*

v	The value
r	The row ordinal
c	The column ordinal
Return:	The updated value

ICellFunc is the function prototype for Apply2Cells.

Example:

```
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);*

m	Number of rows
n	Number of columns
x	Matrix
f	Function to apply
Return:	The updated matrix

The **Apply2Cells** function iterates a matrix and applies a function to each cell.

Example:

```
IMPORT STD;
STD.BLAS.Types.value_t example_1(STD.BLAS.Types.value_t v,
                                 STD.BLAS.Types.dimensiopn_t x,
                                 STD.BLAS.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);*

m	Number of entries
x	The column major matrix holding the vector
incxx	The increment for x, 1 in the case of an actual vector
skipped	The number of entries stepped over. Default is zero.
Return:	The sum of the absolute values

The **dasum** function gets the absolute sum, the 1 norm of a vector.

Example:

```
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);*

N	Number of entries
alpha	The column major matrix holding the vector
X	The column major matrix holding the vector X
incX	The increment for x, 1 in the case of an actual vector
Y	The column major matrix holding the vector Y
incY	The increment or stride of Y
x_skipped	The number of entries stepped over. to get to the first X .
y_skipped	The number of entries stepped over. to get to the first Y .
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:

```
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]
```

dgemm

STD.BLAS.dgemm(*transposeA, transposeB, M, N, K, alpha, A, B, beta, C);*

transposeA	True when transpose of A is used
transposeB	True when transpose of B is used
M	Number of rows in product
N	Number of columns in product
K	Number of columns/rows for the multiplier/multiplicand
alpha	Scalar used on A
A	Matrix A
B	Matrix B
beta	Scalar for matirx C
C	Matrix C (or empty)
Return:	The updated matrix

The **dgemm** function is used to multiply two matrices and optionally add that product to another matrix.

Example:

```
IMPORT STD;
STD.BLAS.Types.t_matrix term_a := [2, 4, 8];
STD.BLAS.Types.t_matrix term_c := [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);*

m	Number of rows of matrix a
n	Number of columns of matrix a
a	Matrix a
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:

```
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);*

tri	Indicates whether upper or lower triangle is used
r	Number of rows/columns in the square matrix
A	The square matrix A
clear	Clears the unused triangle
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:

```
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);*

N	Number of elements in the vector
alpha	The scaling factor
X	The column major matrix holding the vector
incX	The stride to get to the next element in the vector
skipped	The number of elements skipped to get to the first element
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);*

tri	Indicates whether upper or lower triangle is used
transposeA	Transpose the A matrix to be NxK
N	Number of rows
K	Number of columns in the update matrix or transpose
alpha	The alpha scalar
A	The update matrix, either NxK or KxN
beta	The beta scalar
C	The matrix to update
clear	Clear the triangle that is not updated. BLAS assumes that symmetric matrices have only one of the triangles and this option lets you make that true.
Return:	The updated matrix

The **dsyrk** function implements a symmetric rank update C <- alpha A**T A + beta C or c <- alpha A A**T + beta C. C is N x N.

Example:

```
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)
```

dtrsm

STD.BLAS.dtrsm(*side, tri, transposeA, diag, M, N, lda, alpha, A, B);*

side	Side for A, Side.Ax is op(A) X = alpha B
tri	Indicates whether upper or lower triangle is used
transposeA	Is op(A) the transpose of A
diag	The diagonal (an implied unit diagonal or supplied)
M	Number of rows
N	Number of columns
lda	The leading dimension of the A matrix, either M or N
alpha	The scalar multiplier for B
A	A triangular matrix
B	The matrix of values for the solve
Return:	The matrix of coefficients to get B

The **dtrsm** function is a triangular matrix solver. op(A) X = alpha B or X op(A) = alpha B * where op is Transpose, X and B is MxN

Example:

```
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, 3, 1.0, left_a0, mat_b);
```

extract_diag

STD.BLAS.extract_diag (*m.n.x);*

m	Number of rows
n	Number of columns
x	The matrix from which to extract the diagonal
Return:	Diagonal matrix

The **extract_diag** function extracts the diagonal of he 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];
diagonal_only := := STD.BLAS.extract_diag(3, 3, x);
```

extract_tri

STD.BLAS.extract_tri (*m, n, tri, dt, a);*

m	Number of rows
n	Number of columns
tri	Indicates whether upper or lower triangle is used
dt	Use Diagonal.NotUnitTri or Diagonal.UnitTri
a	The matrix, usually a composite from factoring
Return:	Triangle

The **extract_tri** function extracts the upper or lower triangle. The diagonal can be the actual or implied unit diagonal.

Example:

```
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);*

m	Number of diagonal entries
v	Option value, default is 1
X	Optional input of diagonal values, multiplied by v
Return:	A diagonal matrix

The **make_diag** function generates a diagonal matrix.

Example:

```
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);*

m	Number of elements
v	The values, default is 1
Return:	The vector

The **make_vector** function generates a vector of dimension n

Example:

```
IMPORT STD;
twos_vector := STD.BLAS.make_vector(4, 2); // a vector of [2, 2, 2, 2]
```

trace

STD.BLAS.trace (*m, n, x);*

m	Number of rows
n	Number of columns
x	The matrix
Return:	The trace (sum of the diagonal entries)

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
```

Math Support

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:

```
IMPORT STD;
myValue := STD.Math.Infinity;
myValue;
```

See Also: isInfinite

NaN

STD.Math.NaN;

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

The **NaN** function returns a non-signalling NaN (Not a Number) value..

Example:

```
IMPORT STD;
myValue := STD.Math.NaN;
myValue;
```

See Also: IsNan

isInfinite

STD.Math.isInfinite(*val);*

val	The value to test
Return:	Returns a BOOLEAN indicating whether a real value is infinite (positive or negative).

The **isInfinite** function returns whether a real value is infinite (positive or negative).

Example:

```
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);*

val	The value to test
Return:	Returns a BOOLEAN indicating whether a real value is a NaN (not a number) value.

The **isNaN** function returns whether a real value is a NaN (not a number) value.

Example:

```
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);*

val	The value to test
Return:	Returns a BOOLEAN indicating whether a real value is a valid value (neither infinite not NaN).

The **isFinite** function returns whether a real value is a valid value (neither infinite not NaN).

Example:

```
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);*

numer	The numerator
denom	The denominator
Return:	Returns the floating-point remainder of numer/denom (rounded towards zero).

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:

```
#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);*

a	The first value.
b	The second value.
epsilon	The allowable margin of error.
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:

```
IMPORT STD;
STD.Math.FMatch(2.6,2.2,0.5); //true
STD.Math.FMatch(2.6,2.2,0.3); //false
```

Key/Value Store Support

This section provides support for accessing the key/value store functionality on HPCC Systems clusters. The Store module allows you to create stores, manage namespaces, and perform key/value operations for persisting data across workunit executions.

Key/Value Store Overview

The Store module provides a persistent key/value storage mechanism that allows data to be stored and retrieved across multiple workunit executions. The key/value store is organized hierarchically:

- **Stores**: Top-level containers that hold namespaces. Stores can be global (accessible to all users) or user-specific (private to an individual user).
- **Namespaces**: Logical partitions within stores that group related keys together.
- **Keys**: Named identifiers that map to stored values. Keys can be global or user-specific.
- **Values**: String data associated with keys.

The Store module requires authentication credentials and can operate against a specific ESP URL or use the default ESP process on the cluster.

Store Module

myStoreModule := **STD.System.Store([***username***],[** *userPW***],[***espURL***]);**

myStoreModule	The name of the Store module instance
username	The username of the user requesting access to the key value store; this is typically the same username used to login to ECL Watch; set to an empty string if authentication is not required; OPTIONAL, defaults to an empty string
userPW	The password of the user requesting access to the key value store; this is typically the same password used to login to ECL Watch; set to an empty string if authentication is not required; OPTIONAL, defaults to an empty string
espURL	The full URL for accessing the esp process running on the HPCC Systems cluster (this is typically the same URL as used to access ECL Watch); set to an empty string to use the URL of the current esp process as found via Std.File.GetEspURL(); OPTIONAL, defaults to an empty string
Return:	A reference to the module, correctly initialized with the given access parameters

A Store module instance is defined in ECL with the necessary authentication credentials and ESP endpoint. Subsequent function calls use the module instance to access the key/value store.

Example:

```
IMPORT STD;

// Store module definition with credentials
myStore := STD.System.Store('myusername', 'mypassword', 'http://localhost:8010');

// Store module definition using default ESP URL
myDefaultStore := STD.System.Store();
```

CreateStore

myStoreModule.**CreateStore(***storeName***[,** *description***][,** *maxValueSize***][,** *isUserSpecific***][,** *timeoutInSeconds***]);**

myStoreModule	The name of the Store module instance
storeName	A STRING naming the store to create; this cannot be an empty string; REQUIRED
description	A STRING describing the purpose of the store; may be an empty string; OPTIONAL, defaults to an empty string
maxValueSize	The maximum size of any value stored within this store, in bytes; use a value of zero to indicate an unlimited maximum size; OPTIONAL, defaults to 1024
isUserSpecific	If TRUE, this store will be visible only to the user indicated by the (username, userPW) arguments provided when the module was defined; if FALSE, the store will be global and visible to all users; OPTIONAL, defaults to FALSE
timeoutInSeconds	The number of seconds to wait for the underlying SOAPCALL to complete; set to zero to wait forever; OPTIONAL, defaults to zero
Return:	A *CreateStoreResponseRec* RECORD. If the store already exists then the result will show succeeded = FALSE and already_present = TRUE.

The **CreateStore** function creates a key/value store if it has not been created before. If the store has already been created then this function has no effect.

Example:

```
IMPORT STD;

myStore := STD.System.Store('myusername', 'mypassword');

// Create a global store
result := myStore.CreateStore('MyApplicationStore', 'Store for application data', 2048, FALSE);

OUTPUT(result.succeeded);
OUTPUT(result.already_present);

// Create a user-specific store
privateResult := myStore.CreateStore('MyPrivateStore', 'My private data', 4096, TRUE);

OUTPUT(privateResult.succeeded);
```

See Also: CreateStoreResponseRec

ListStores

myStoreModule.**ListStores([***nameFilter***][,** *ownerFilter***][,** *timeoutInSeconds***]);**

myStoreModule	The name of the Store module instance
nameFilter	A STRING defining a filter to be applied to the store's name; the filter accepts the '*' wildcard character to indicate 'match anything' and '?' to match any single character; string comparisons are case-insensitive; an empty string is equivalent to '*'; OPTIONAL, defaults to '*'
ownerFilter	A STRING defining a filter to be applied to the store's owner; the filter accepts the '*' wildcard character to indicate 'match anything' and '?' to match any single character; string comparisons are case-insensitive; an empty string is equivalent to '*'; OPTIONAL, defaults to '*'
timeoutInSeconds	The number of seconds to wait for the underlying SOAPCALL to complete; set to zero to wait forever; OPTIONAL, defaults to zero
Return:	A *ListStoresResponseRec* RECORD

The **ListStores** function gets a list of available stores based on the provided filters.

Example:

```
IMPORT STD;

myStore := STD.System.Store();

// List all stores
allStores := myStore.ListStores();
OUTPUT(allStores.stores);

// List stores starting with 'App'
appStores := myStore.ListStores('App*', '*');
OUTPUT(appStores.stores);

// List stores owned by specific user
userStores := myStore.ListStores('*', 'myusername');
OUTPUT(userStores.stores);
```

See Also: ListStoresResponseRec

ListNamespaces

myStoreModule.**ListNamespaces(***storeName***[,** *isUserSpecific***][,** *timeoutInSeconds***]);**

myStoreModule	The name of the Store module instance
storeName	A STRING naming the store containing the namespaces you are interested in; set this to an empty string to reference the default store in the cluster, if one has been defined; REQUIRED
isUserSpecific	If TRUE, the system will look only for private keys; if FALSE then the system will look for global keys; OPTIONAL, defaults to FALSE
timeoutInSeconds	The number of seconds to wait for the underlying SOAPCALL to complete; set to zero to wait forever; OPTIONAL, defaults to zero
Return:	A *ListNamespacesResponseRec* RECORD

The **ListNamespaces** function gets a list of namespaces defined in the specified store.

Example:

```
IMPORT STD;

myStore := STD.System.Store();

// List all namespaces in the default store
namespaces := myStore.ListNamespaces('');
OUTPUT(namespaces.namespaces);

// List namespaces in a specific store
appNamespaces := myStore.ListNamespaces('MyApplicationStore');
OUTPUT(appNamespaces.namespaces);

// List user-specific namespaces
privateNamespaces := myStore.ListNamespaces('MyPrivateStore', TRUE);
OUTPUT(privateNamespaces.namespaces);
```

See Also: ListNamespacesResponseRec

WithNamespace Module

myNamespaceModule :=**myStoreModule.WithNamespace(***namespace***[,** *storeName***]);**

myStoreModule	The name of the parent Store module instance
myNamespaceModule	The name of the namespace-specific module instance
namespace	A STRING naming the namespace partition that will be used for this module; cannot be an empty string; REQUIRED
storeName	A STRING naming the store that this module will access; set this to an empty string to reference the default store in the cluster, if one has been defined; OPTIONAL, defaults to an empty string
Return:	A reference to the module, correctly initialized with the given namespace

The **WithNamespace** submodule nails down a specific namespace to access within a key/value store on the cluster. All subsequent operations will be scoped to this namespace.

Example:

```
IMPORT STD;

myStore := STD.System.Store();

// Define a namespace module
myNamespace := myStore.WithNamespace('ApplicationConfig', 'MyApplicationStore');

// Use default store with a specific namespace
defaultNamespace := myStore.WithNamespace('Settings', '');
```

SetKeyValue

myNamespaceModule.**SetKeyValue(***keyName*, *keyValue*[, *isUserSpecific*][, *timeoutInSeconds*]);

myNamespaceModule	The name of the namespace-specific module instance
keyName	A STRING naming the key; may not be an empty string; REQUIRED
keyValue	A STRING representing the value to store for the key; may be an empty string; REQUIRED
isUserSpecific	If TRUE, this key will be visible only to the user indicated by the (username, userPW) arguments provided when the module was defined; if FALSE, the key will be global and visible to all users; OPTIONAL, defaults to FALSE
timeoutInSeconds	The number of seconds to wait for the underlying SOAPCALL to complete; set to zero to wait forever; OPTIONAL, defaults to zero
Return:	A *SetKeyValueResponseRec* RECORD

The **SetKeyValue** function sets a value for a key within a namespace. If the key already exists then its value is overridden. The namespace will be created if it has not already been defined.

Example:

```
IMPORT STD;

myStore := STD.System.Store();
myNamespace := myStore.WithNamespace('Config', 'MyStore');

// Set a global key
result1 := myNamespace.SetKeyValue('ApiEndpoint', 'https://api.example.com/v1');
OUTPUT(result1.succeeded);

// Set a user-specific key
result2 := myNamespace.SetKeyValue('UserPreference', 'darkmode', TRUE);
OUTPUT(result2.succeeded);

// Update an existing key
result3 := myNamespace.SetKeyValue('ApiEndpoint', 'https://api.example.com/v2');
OUTPUT(result3.succeeded);
```

See Also: SetKeyValueResponseRec

GetKeyValue

myNamespaceModule.**GetKeyValue(***keyName***[,** *isUserSpecific***][,** *timeoutInSeconds***]);**

myNamespaceModule	The name of the namespace-specific module instance
keyName	A STRING naming the key; may not be an empty string; REQUIRED
isUserSpecific	If TRUE, the system will look only for private keys; if FALSE then the system will look for global keys; OPTIONAL, defaults to FALSE
timeoutInSeconds	The number of seconds to wait for the underlying SOAPCALL to complete; set to zero to wait forever; OPTIONAL, defaults to zero
Return:	A *GetKeyValueResponseRec* RECORD. Note that the record will have was_found set to TRUE or FALSE, depending on whether the key was actually found in the key/value store.

The **GetKeyValue** function gets a previously-set value for a key within a namespace.

Example:

```
IMPORT STD;

myStore := STD.System.Store();
myNamespace := myStore.WithNamespace('Config', 'MyStore');

// Get a global key
result1 := myNamespace.GetKeyValue('ApiEndpoint');
OUTPUT(result1.was_found);
OUTPUT(result1.value);

// Get a user-specific key
result2 := myNamespace.GetKeyValue('UserPreference', TRUE);
IF(result2.was_found,
   OUTPUT(result2.value),
   OUTPUT('Key not found'));

// Check for non-existent key
result3 := myNamespace.GetKeyValue('NonExistentKey');
OUTPUT(result3.was_found);  // FALSE
```

See Also: GetKeyValueResponseRec

DeleteKeyValue

myNamespaceModule.**DeleteKeyValue(***keyName***[,** *isUserSpecific***][,** *timeoutInSeconds***]);**

myNamespaceModule	The name of the namespace-specific module instance
keyName	A STRING naming the key; may not be an empty string; REQUIRED
isUserSpecific	If TRUE, the system will look only for private keys; if FALSE then the system will look for global keys; OPTIONAL, defaults to FALSE
timeoutInSeconds	The number of seconds to wait for the underlying SOAPCALL to complete; set to zero to wait forever; OPTIONAL, defaults to zero
Return:	A *DeleteKeyValueResponseRec* RECORD

The **DeleteKeyValue** function deletes a previously-set key and value within a namespace.

Example:

```
IMPORT STD;

myStore := STD.System.Store();
myNamespace := myStore.WithNamespace('Config', 'MyStore');

// Delete a global key
result1 := myNamespace.DeleteKeyValue('ApiEndpoint');
OUTPUT(result1.succeeded);

// Delete a user-specific key
result2 := myNamespace.DeleteKeyValue('UserPreference', TRUE);
OUTPUT(result2.succeeded);
```

See Also: DeleteKeyValueResponseRec

GetAllKeys

myNamespaceModule.**GetAllKeys([***isUserSpecific***][,** *timeoutInSeconds***]);**

myNamespaceModule	The name of the namespace-specific module instance
isUserSpecific	If TRUE, the system will look only for private keys; if FALSE then the system will look for global keys; OPTIONAL, defaults to FALSE
timeoutInSeconds	The number of seconds to wait for the underlying SOAPCALL to complete; set to zero to wait forever; OPTIONAL, defaults to zero
Return:	A *GetAllKeysResponseRec* RECORD

The **GetAllKeys** function gets a list of all keys currently defined within a namespace.

Example:

```
IMPORT STD;

myStore := STD.System.Store();
myNamespace := myStore.WithNamespace('Config', 'MyStore');

// Get all global keys
result1 := myNamespace.GetAllKeys();
OUTPUT(result1.namespace);
OUTPUT(result1.keys);

// Get all user-specific keys
result2 := myNamespace.GetAllKeys(TRUE);
OUTPUT(result2.keys);
```

See Also: GetAllKeysResponseRec

GetAllKeyValues

myNamespaceModule.**GetAllKeyValues([***isUserSpecific***][,** *timeoutInSeconds***]);**

myNamespaceModule	The name of the namespace-specific module instance
isUserSpecific	If TRUE, the system will look only for private keys; if FALSE then the system will look for global keys; OPTIONAL, defaults to FALSE
timeoutInSeconds	The number of seconds to wait for the underlying SOAPCALL to complete; set to zero to wait forever; OPTIONAL, defaults to zero
Return:	A *GetAllKeyValuesResponseRec* RECORD

The **GetAllKeyValues** function gets a list of all keys and their associated values currently defined within a namespace.

Example:

```
IMPORT STD;

myStore := STD.System.Store();
myNamespace := myStore.WithNamespace('Config', 'MyStore');

// Get all global key/value pairs
result1 := myNamespace.GetAllKeyValues();
OUTPUT(result1.namespace);
OUTPUT(result1.key_values);

// Get all user-specific key/value pairs
result2 := myNamespace.GetAllKeyValues(TRUE);
OUTPUT(result2.key_values);
```

See Also: GetAllKeyValuesResponseRec

DeleteNamespace

myNamespaceModule.**DeleteNamespace([***isUserSpecific***][,** *timeoutInSeconds***]);**

myNamespaceModule	The name of the namespace-specific module instance
isUserSpecific	If TRUE, the system will look only for private keys; if FALSE then the system will look for global keys; OPTIONAL, defaults to FALSE
timeoutInSeconds	The number of seconds to wait for the underlying SOAPCALL to complete; set to zero to wait forever; OPTIONAL, defaults to zero
Return:	A *DeleteNamespaceResponseRec* RECORD

The **DeleteNamespace** function deletes the namespace defined for this module and all keys and values defined within it.

Example:

```
IMPORT STD;

myStore := STD.System.Store();
myNamespace := myStore.WithNamespace('TempData', 'MyStore');

// Delete the entire namespace and all its keys
result := myNamespace.DeleteNamespace();
OUTPUT(result.succeeded);

// Delete a user-specific namespace
privateNamespace := myStore.WithNamespace('PrivateTemp', 'MyPrivateStore');
result2 := privateNamespace.DeleteNamespace(TRUE);
OUTPUT(result2.succeeded);
```

See Also: DeleteNamespaceResponseRec

Record Definitions

The Store module exports several RECORD definitions that are used as return types for the various functions. These records provide structured access to results and error information.

All response records include exception handling fields to provide detailed error information when operations fail. Check the *has_exceptions* field before accessing operation results.

ExceptionLayout

Defines the structure of a single exception or error message.

```
EXPORT ExceptionLayout := RECORD
  STRING    code;
  STRING    audience;
  STRING    source;
  STRING    message;
END;
```

code	Error code identifying the type of exception
audience	Intended audience for the exception message
source	Source component that generated the exception
message	Human-readable description of the exception

ExceptionListLayout

Contains a list of exceptions that occurred during an operation.

```
EXPORT ExceptionListLayout := RECORD
  STRING                      source;
  DATASET(ExceptionLayout)    exceptions;
END;
```

source	Source component that generated the exceptions
exceptions	Dataset of ExceptionLayout records containing detailed exception information

StoreInfoRec

Contains information about a key/value store.

```
EXPORT StoreInfoRec := RECORD
  STRING        store_name;
  STRING        description;
  STRING        owner;
  STRING        create_time;
  UNSIGNED4     max_value_size;
  BOOLEAN       is_default;
END;
```

store_name	Name of the store
description	Description of the store's purpose
owner	Username of the store owner
create_time	Timestamp when the store was created

max_value_size	Maximum size in bytes for values stored in this store
is_default	TRUE if this is the default store for the cluster

ListStoresResponseRec

Response record returned by the ListStores function.

```
EXPORT ListStoresResponseRec := RECORD
  DATASET(StoreInfoRec)       stores;
  BOOLEAN                     has_exceptions;
  ExceptionListLayout         exceptions;
END;
```

stores	Dataset of StoreInfoRec records containing information about each matching store
has_exceptions	TRUE if any exceptions occurred during the operation
exceptions	ExceptionListLayout record containing any error details

CreateStoreResponseRec

Response record returned by the CreateStore function.

```
EXPORT CreateStoreResponseRec := RECORD
  BOOLEAN                     succeeded;
  BOOLEAN                     already_present;
  STRING                      store_name;
  STRING                      description;
  BOOLEAN                     has_exceptions;
  ExceptionListLayout         exceptions;
END;
```

succeeded	TRUE if a new store was created, FALSE if store already existed or an error occurred
already_present	TRUE if the store already existed
store_name	Name of the store
description	Description of the store
has_exceptions	TRUE if any exceptions occurred during the operation
exceptions	ExceptionListLayout record containing any error details

SetKeyValueResponseRec

Response record returned by the SetKeyValue function.

```
EXPORT SetKeyValueResponseRec := RECORD
  BOOLEAN                     succeeded;
  BOOLEAN                     has_exceptions;
  ExceptionListLayout         exceptions;
END;
```

succeeded	TRUE if the key/value was successfully set
has_exceptions	TRUE if any exceptions occurred during the operation
exceptions	ExceptionListLayout record containing any error details

GetKeyValueResponseRec

Response record returned by the GetKeyValue function.

```
EXPORT GetKeyValueResponseRec := RECORD
  BOOLEAN                   was_found;
  STRING                    value;
  BOOLEAN                   has_exceptions;
  ExceptionListLayout       exceptions;
END;
```

was_found	TRUE if the key was found, FALSE if the key does not exist
value	The value associated with the key, if found
has_exceptions	TRUE if any exceptions occurred during the operation
exceptions	ExceptionListLayout record containing any error details

DeleteKeyValueResponseRec

Response record returned by the DeleteKeyValue function.

```
EXPORT DeleteKeyValueResponseRec := RECORD
  BOOLEAN                   succeeded;
  BOOLEAN                   has_exceptions;
  ExceptionListLayout       exceptions;
END;
```

succeeded	TRUE if the key/value was successfully deleted
has_exceptions	TRUE if any exceptions occurred during the operation
exceptions	ExceptionListLayout record containing any error details

GetAllKeysResponseRec

Response record returned by the GetAllKeys function.

```
EXPORT GetAllKeysResponseRec := RECORD
  STRING                    namespace;
  DATASET(KeySetRec)        keys;
  BOOLEAN                   has_exceptions;
  ExceptionListLayout       exceptions;
END;
```

namespace	Name of the namespace
keys	Dataset containing all keys defined in the namespace
has_exceptions	TRUE if any exceptions occurred during the operation
exceptions	ExceptionListLayout record containing any error details

GetAllKeyValuesResponseRec

Response record returned by the GetAllKeyValues function.

```
EXPORT GetAllKeyValuesResponseRec := RECORD
  STRING                    namespace;
  DATASET(KeyValueRec)      key_values;
```

```
  BOOLEAN                   has_exceptions;
  ExceptionListLayout       exceptions;
END;
```

namespace	Name of the namespace
key_values	Dataset containing all key/value pairs defined in the namespace
has_exceptions	TRUE if any exceptions occurred during the operation
exceptions	ExceptionListLayout record containing any error details

ListNamespacesResponseRec

Response record returned by the ListNamespaces function.

```
EXPORT ListNamespacesResponseRec := RECORD
  DATASET(NamespaceLayout)  namespaces;
  BOOLEAN                   has_exceptions;
  ExceptionListLayout       exceptions;
END;
```

namespaces	Dataset containing all namespaces defined in the store
has_exceptions	TRUE if any exceptions occurred during the operation
exceptions	ExceptionListLayout record containing any error details

DeleteNamespaceResponseRec

Response record returned by the DeleteNamespace function.

```
EXPORT DeleteNamespaceResponseRec := RECORD
  BOOLEAN                   succeeded;
  BOOLEAN                   has_exceptions;
  ExceptionListLayout       exceptions;
END;
```

succeeded	TRUE if the namespace was successfully deleted
has_exceptions	TRUE if any exceptions occurred during the operation
exceptions	ExceptionListLayout record containing any error details