HPCC Systems®

Standard Library Reference

Boca Raton Documentation Team

LexisNexis® Risk Solutions

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.

2013 Version 3.10.8.2

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* **])**

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.
Return:	LogicalFileList returns returns a dataset in the following format:

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

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

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

Example:

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

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

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) and is not a SuperFile (use the STD.File.SuperFileExists function to detect their presence or absence). 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');
```

ForeignLogicalFileName

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

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

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

Example:

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

OUTPUT(STD.File.GetLogicalFileAttribute(file,'recordSize'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'recordCount'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'size'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'clusterName'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'directory'));
OUTPUT(STD.File.GetLogicalFileAttribute(file,'numparts'));
```

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
```

RemoteDirectory

STD.File.RemoteDirectory(*machineIP*, *directory* **[** , *mask* **][** , *includesubs* **]** **)**

machineIP	A null-terminated string containing the IP address of the remote machine.
directory	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).
includesubdir	Optional. A boolean flag indicating whether to include files from sub-directories under the *directory*. If omitted, the default is FALSE.
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 *includesubdir* is set to TRUE, then the name field contains the relative path to the file from the specified *directory*.

Example:

```
OUTPUT(STD.File.RemoteDirectory('edata12','\in','*.d00'));
OUTPUT(STD.File.RemoteDirectory('10.150.254.6',
      '/c$/training',,TRUE));
```

RenameLogicalFile

STD.File.RenameLogicalFile(*filename, newname* **)**

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.

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* **] [** , *allow_exist* **]** **)**

superfile	A null-terminated string containing the logical name of the superfile.
sequentialparts	Optional. A boolean value indicating whether to the sub-files must be sequentially ordered. If omitted, the default is FALSE.
allow_exist	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 sub-files must be sequential (for example, where all sub-files are already globally sorted). With *sequentialparts* FALSE (the default) the subfile parts are interleaved so the parts are found locally.

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

A._1_of_4, B._1_of_4, and C_1_of_4 are on node 1

A._2_of_4, B._2_of_4, and C_2_of_4 are on node 2

A._3_of_4, B._3_of_4, and C_3_of_4 are on node 3

A._4_of_4, B._4_of_4, and C_4_of_4 are on node 4

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

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

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

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

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

Example:

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

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 in the DFU but is not a SuperFile (i.e. 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');
```

DeleteSuperFile

STD.File.DeleteSuperFile(*superfile* **[** , *subdeleteflag* **]** **)**

superfile	A null-terminated string containing the logical name of the superfile.
subdeleteflag	A boolean value indicating whether to delete the sub-files. 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 *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.GetSuperFileSubName('~CLASS::RT::IN::SF1', 'Sue');    //get position of sub-file 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 adding, removing, clearing, swapping, and replacing sub-files.

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:

```
STD.File.StartSuperFileTransaction();
```

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. Ths 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* **[** , *delete* **] [** , *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.
delete	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()
);
```

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(*superfiles* **[** , *addhead* **]** **[** , *deltail* **]** **[** , *createjustone* **]** **[** , *reverse* **]** **)**

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

superfiles	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 procesing the *superfiles* 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:	PromoteSupeFileList returns Null; fPromoteSupeFileList returns a string.

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

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 transction is committed atomically and the transaction frame closes. Otherwise, the transaction is rolled back and the transaction frame closes.

Example:

```
STD.File.FinishSuperFileTransaction();
```

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.

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
```

MoveExternalFile

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

location	A null-terminated string containing the IP address of the remote machine.
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.

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:

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

DeleteExternalFile

STD.File.DeleteExternalFile(*location, path* **)**

location	A null-terminated string containing the IP address of the remote machine.
path	A null-terminated string containing the path/name of the file to remove.

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:

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

CreateExternalDirectory

STD.File.CreateExternalDirectory(*location, path* **)**

location	A null-terminated string containing the IP address of the remote machine.
path	A null-terminated string containing the directory path to create.

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

Example:

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

Remote Files

EncodeRfsQuery

result **:= STD.File.EncodeRfsQuery(** *server, query* **);**

server	A null-terminated string containing the ip:port address for the remote file server.
query	A null-terminated string containing the query to send to the *server*.
Return:	RfsQuery returns a null-terminated string containing the result of the *query*.

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

Example:

```
IMPORT Std;
rfsserver := '10.173.207.1:7080';
rec := RECORD,MAXLENGTH(8192)
  STRING  mydata;
END;
OUTPUT(DATASET(STD.File.EncodeRfsQuery( rfsserver,
              'SELECT data FROM xml_testnh'),rec,CSV(MAXLENGTH(8192))));
```

RfsAction

STD.File.RfsAction(*server, query* **);**

server	A null-terminated string containing the ip:port address for the remote file server.
query	A null-terminated string containing the query to send to the *server*.

The **RfsAction** function sends the *query* to the *server*. This is used when there is no expected return value

Example:

```
rfsserver := '10.173.207.1:7080';
STD.File.RfsAction(rfsserver,'INSERT INTO xml_testnh (data) VALUES (\''+TRIM(A)+'\' )');
```

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* mist 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 sub-files.

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

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

```
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.exe command line to execute. The valid arguments are documented in the Client Tools manual, in the section describing the DfuPlus.exe program.

The **DfuPlusExec** action executes the specified *commandline* just as the DfuPLus.exe program would do. This simply allows you to have all the functionality of the DfuPLus.exe program available within your ECL code.

Example:

```
IMPORT STD;

serv := 'server=http://10.150.50.12:8010 ';
user := 'username=rlor ';
pswd := 'password=password ';
over := 'overwrite=1 ';
repl := 'replicate=1 ';

action  := 'action=despray ';
dstip   := 'dstip=10.150.50.14 ';
dstfile := 'dstfile=c:\\import\\despray\\timezones.txt ';
srcname := 'srcname=RTTEMP::timezones.txt ';

cmd := serv + user + pswd + over + repl + action + dstip + dstfile + srcname;
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. 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. If omitted, the default is the value contained in the lib_system.ws_fs_server attribute.

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(*logicalname, destinationGroup* **,** *destinationLogicalname,* **[** *,scrDali* **] [** *,timeout* **] [** *,espserverIPport* **] [** *,maxConnections* **] [** *,allowoverwrite* **] [** *,replicate* **] [** *,asSuperfile* **]);**

dfuwuid **:= STD.File.fCopy(** *logicalname, destinationGroup* **,** *destinationLogicalname,* **[** *,scrDali* **] [** *,timeout* **] [** *,espserverIPport* **] [** *,maxConnections* **] [** *,allowoverwrite* **] [** *,replicate* **] [** *,asSuperfile* **]);**

logicalname	A null-terminated string containing the logical name of the file.
destinationGroup	A null-terminated string containing the destination cluster for the file.
destinationLogicalname	A null-terminated string containing the new logical name of the file.
srcDali	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. 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. If omitted, the default is the value contained in the lib_system.ws_fs_server attribute.
maxConnections	Optional. An integer specifying the maximum number of connections. If omitted, the default is one (1).
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.
dfuwuid	The attribute name to recieve the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job.
Return:	fCopy 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, or to a cluster in a completely separate Dali.

Example:

```
STD.File.Copy('OUT::MyFile','400way','OUT::MyNewFile');
```

DeSpray

STD.File.DeSpray(*logicalname, destinationIP* **,** *destinationpath,* **[** *,timeout* **] [** *,espserverIPport* **] [** *,maxConnections* **] [** *,allowoverwrite* **])**

dfuwuid **:= STD.File.fDeSpray(** *logicalname, destinationIP* **,** *destinationpath,* **[** *,timeout* **] [** *,espserverIPport* **] [** *,maxConnections* **] [** *,allowoverwrite* **]);**

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.
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. 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. If omitted, the default is the value contained in the lib_system.ws_fs_server attribute.
maxConnections	Optional. An integer specifying the maximum number of connections. If omitted, the default is one (1).
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.
dfuwuid	The attribute name to recieve 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* **])**

dfuwuid **:= STD.File.RemotePull(** *remoteURL, sourcelogicalname, destinationGroup* **,** *destinationlogicalname,* **[** *,timeout* **] [** *,maxConnections* **] [** *,allowoverwrite* **] [** *,replicate* **] [** *,asSuperfile* **]);**

remoteURL	A null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the remote ESP server program. This is usually the same IP and port as its ECL Watch, with "/FileSpray" appended.
sourcelogicalname	A null-terminated string containing the local logical name of the file.
destinationGroup	A null-terminated string containing the name of the destination cluster.
destinationlogicalname	A null-terminated string containing the logical name to give the file on the remote cluster (this must be completely specified, including the domain).
timeout	Optional. An integer value indicating the timeout setting. If omitted, the default is -1. If set to zero (0), execution control returns immediately to the ECL workunit without waiting for the DFU workunit to complete.
maxConnections	Optional. An integer specifying the maximum number of connections. If omitted, the default is one (1).
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.
dfuwuid	The attribute name to recieve 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. 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. If omitted, the default is the value contained in the lib_system.ws_fs_server attribute.
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* **])**

dfuwuid **:= STD.File.fSprayFixed(** *sourceIP* **,** *sourcepath, recordsize, destinationgroup, destinationlogicalname* **[** *,timeout* **] [** *, espserverIPport* **] [** *, maxConnections* **] [** *, allowoverwrite* **] [** *, replicate* **] [** *, compress* **]);**

sourceIP	A null-terminated string containing the IP address of the file.
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	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. If omitted, the default is the value contained in the lib_system.ws_fs_server attribute.
maxConnections	Optional. An integer specifying the maximum number of connections. If omitted, the default is one (1).
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 FALSE.
dfuwuid	The attribute name to recieve 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');
```

SprayVariable

STD.File.SprayVariable(*sourceIP* **,** *sourcepath* **,** **[** *maxrecordsize* **]** **,** **[** *srcCSVseparator* **]** **,** **[** *srcCSVterminator* **]** **,** **[** *srcCSVquote* **]** **,** *destinationgroup, destinationlogicalname* **[** *,timeout* **]** **[** *,espserverIPport* **]** **[** *,maxConnections* **]** **[** *,allowoverwrite* **]** **[** *,replicate* **]** **[** *, compress* **])** *, sourceCsvEscape* **])**

dfuwuid **:= STD.File.fSprayVariable(** *sourceIP* **,** *sourcepath* **,** **[** *maxrecordsize* **]** **,** **[** *srcCSVseparator* **]** **,** **[** *srcCSVterminator* **]** **,** **[** *srcCSVquote* **]** *, destinationgroup, destinationlogicalname* **[** *,timeout* **]** **[** *,espserverIPport* **]** **[** *,maxConnections* **]** **[** *,allowoverwrite* **]** **[** *,replicate* **]** **[** *, compress* **]);** *, sourceCsvEscape* **]);**

sourceIP	A null-terminated string containing the IP address of the file.
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 4096.
srcCSVseparator	Optional. A null-terminated string containing the CSV field separator. If omitted, the default is '\\,'
srcCSVterminator	Optional. A null-terminated string containing the CSV record separator. If omitted, the default is '\\n,\\r\\n'
srcCSVquote	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. 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. If omitted, the default is the value in the lib_system.ws_fs_server attribute.
maxConnections	Optional. An integer specifying the maximum number of connections. If omitted, the default is one (1).
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 FALSE.
sourceCsvEscape	Optional. A null-terminated string containing the CSV escape characters. If omitted, the default is none.
dfuwuid	The attribute name to recieve the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the job.
Return:	fSprayVariable returns a null-terminated string containing the DFU workunit ID (DFUWUID).

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

Example:

```
STD.File.SprayVariable('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* **])**

dfuwuid **:= STD.File.fSprayXML(** *sourceIP* **,** *sourcepath,* **[** *maxrecordsize* **] ,** *srcRowTag* **, [** *srcEncoding* **] ,***destinationgroup, destinationlogicalname* **[** *,timeout* **] [** *,espserverIPport* **] [** *,maxConnections* **] [** *,allowoverwrite* **] [** *,replicate* **] [** *, compress* **]);**

sourceIP	A null-terminated string containing the IP address of the file.
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.
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. 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. If omitted, the default is the value contained in the lib_system.ws_fs_server attribute.
maxConnections	Optional. An integer specifying the maximum number of connections. If omitted, the default is one (1).
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 FALSE.
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');
```

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. 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. If omitted, the default is the value contained in the lib_system.ws_fs_server attribute.
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');
```

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'
```

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);

B:= STD.Str.Contains(
 'the speedy ochre vixen leapt over the indolent retriever',
 'abcdefghijklmnopqrstuvwxyz', false);
```

CountWords

STD.Str.CountWords(*source*, *separator* **)**

source	A string containing the words to count.
separator	A string containing the word delimiter to use.
Return:	CountWords returns an integer value.

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

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
```

EditDistance

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

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

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.
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 i\reflects the minimum number of operations needed to transform *string1* into *string2*.

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
```

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
```

EndsWith

STD.Str.EndsWith(*source, suffix* **)**

source	The string to search.
suffix	The string to find.
Return:	EndsWith returns a BOOLEAN value.

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

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* **)**

text	A string containing words separated by whitespace.
Return:	ExcludeFirstWord returns a STRING value.

The **ExcludeFirstWord** function returns the *text* string with the first word removed. Words are separated by one or more whitespace characters. 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* **)**

text	A string containing words separated by whitespace.
Return:	ExcludeLastWord returns a STRING value.

The **ExcludeLastWord** function returns the *text* string with the last word removed. Words are separated by one or more whitespace characters. 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* **)**

text	A string containing words separated by whitespace.
n	A integer containing the ordinal position of the word to remove.
Return:	ExcludeNthWord returns a STRING value.

The **ExcludeNthWord** function returns the *text* string with the *n*th word removed. Words are separated by one or more whitespace characters. Whitespace after the *n*th word is also removed (along with whitespace before, if *n*=1).

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');
```

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*, StringFind returns zero (0).

Example:

```
A := IF(STD.Str.Find('ABCDE', 'BC',1) = 2,
   'Success',
   'Failure - 1');  //success

B := IF(STD.Str.Find('ABCDEABCDE', 'BC', 2) = 7,
   'Success',
   'Failure - 2');  //success

C := IF(STD.Str.Find('ABCDEABCDE', '') = 0,
   'Success',
   'Failure - 3');  //syntax error, missing 3rd parameter

D := IF(STD.Str.Find('', 'BD', 1) = 0,
   'Success',
   'Failure - 4');  //success
```

FindCount

STD.Str.FindCount(*source, target* **)**

source	A string containing the data to search.
target	A string containing the substring to search for.
Return:	StringFindCount returns an INTEGER value.

The **FindCount** function returns the number of non-overlapping instances of the *target* string within the *source* string.

Example:

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

B := IF(STD.Str.FindCount('ABCDEABCDE', 'BC') = 2,
   'Success',
   'Failure - 2');  //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'
```

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(*source, suffix* **)**

source	The string to search.
suffix	The ending string to remove.
Return:	RemoveSuffix returns a string value.

The **RemoveSuffix** function returns the *source* string with the ending text in the *suffix* parameter removed. If the *source* string does not end with the *suffix*, then the *source* string is returned unchanged.

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'
```

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(*source, separator* **)**

source	A string containing the words to extract.
separator	A string containing the word delimiter to use.
Return:	SplitWords returns a SET OF STRING values.

The **SplitWords** function returns the list of words in the *source* string split out by the specified *separator*.

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(*source, prefix* **)**

source	The string to search.
prefix	The string to find.
Return:	StartsWith returns a BOOLEAN value.

The **StartsWith** function returns TRUE if the *source* starts with the text in the *suffix* parameter.

Example:

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

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’
```

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.CountWords(str1);    //finds eight words based on space delimiter
```

Date 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;
```

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
```

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
```

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
```

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 a Date_t value.

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

Example:

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

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

Y := STD.Date.ToGregorianYMD(J).Year;   //Y contains 2012
M := STD.Date.ToGregorianYMD(J).Month;  //M contains 1
D := STD.Date.ToGregorianYMD(J).Day;    //D contains 1
```

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) servers for the environment running the workunit.

Example:

```
A := Thorlib.Daliservers();
```

Group

STD.System.Thorlib.Group ( )

Return: Group returns a VARSTRING value.

The **Group** function returns the name of the node group running the workunit. Not supported on Roxie clusters. This name is used in ECL code to specify the target CLUSTER for an OUTPUT action or a PERSISTed attribute.

Example:

```
A := Thorlib.Group();
```

GetExpandLogicalFileName

ThorLib.GetExpandLogicalFileName (*filename*)

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

The **GetExpandLogicalFileName** function returns a string containing the expanded logical filename (ncluding 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:

```
A := ThorLib.GetExpandLogicalFileName('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.Thorlib.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 ECLplus.exe to specify the the target cluster for a workunit.

Example:

```
A := STD.System.Thorlib.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. 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. If omitted, the default is the value contained in the lib_system.ws_fs_server attribute.
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 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 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. This may contain wildcard characters (* and ?).
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. 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. If omitted, the default is the value contained in the lib_system.ws_fs_server attribute.
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 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** function writes the *text* string to the eclagent.log file for the workunit.

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** function 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** function 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** function 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
```

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));
```

CmdProcess

result **:= STD.System.Util.CmdProcess (*program*, *input*);**

program	A null-terminated string containing the name of the program to execute. This may include command-line parameters.
input	A string containing the text to pipe into the *program* through stdin.
Return:	CmdProcess returns returns a STRING value.

The **CmdProcess** function pipes the *input* text to the specified *program*. This is similar to the PIPE built-in function, but limited to simple text input and output.

Example:

```
IMPORT STD;

OUTPUT(STD.System.Util.CmdProcess('echo','George Jetson'));
```

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 slave nodes of the specified *dali*.

Example:

```
IMPORT STD;

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

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 (*sendto, subject, body, server, port, sender*)

sendto	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).
server	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.

The **SendEmail** function sends an email message.

Example:

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

SendEmailAttachData

STD.System.Email.SendEmailAttachData (*sendto, subject, body, attachment, mimietype, filename, server, port, sender*)

sendto	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.
server	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.

The **SendEmailAttachData** function sends an email message with a binary *attachment*.

Example:

```
DATA15 attachment :=  D'test attachment';
STD.System.Email.SendEmailAttachData( 'me@mydomain.com',
                                  'testing 1,2,3',
                                  'this is a test message',
                                   attachment,
                                  'application/octet-stream',
                                  'attachment.txt');
```

SendEmailAttachText

STD.System.Email.SendEmailAttachText (*sendto, subject, body, attachment, mimietype, filename, server, port, sender*)

sendto	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.
server	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.

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*])

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.
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
```

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 WsTimeStamp := RECORD
  STRING32 application;
  STRING16 id;
  STRING20 time;
  STRING16 instance;
END;
```

Each record in the returned dataset specifies a step in the workunit's execution process (creation, compilation, etc.).

Example:

```
OUTPUT(STD.System.Workunit.WorkunitTimeStamps('W20070308-164946'));
/* produces output like this:
'workunit     ','Created ','2008-02-13T18:28:20Z','              '
'workunit     ','Modified','2008-02-13T18:32:47Z','              '
'EclServer    ','Compiled','2008-02-13T18:28:20Z','10.173.9.2:0  '
'EclAgent     ','Started ','2008-02-13T18:32:35Z','training009003'
'Thor - graph1','Finished','2008-02-13T18:32:47Z','training009004'
'Thor - graph1','Started ','2008-02-13T18:32:13Z','training009004'
'EclAgent     ','Finished','2008-02-13T18:33:09Z','training009003'
*/
```

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 := RECORD
  UNSIGNED4 severity;
  INTEGER4 code;
  STRING32 location;
  UNSIGNED4 row;
  UNSIGNED4 col;
  STRING16 source;
  STRING20 time;
  STRING message{MAXLENGTH(1024)};
END;
```

Each record in the returned dataset specifies a message in the workunit.

Example:

```
OUTPUT(STD.System.Workunit.WorkunitMessages('W20070308-164946'));
```

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'
*/
```