The ECL Scheduler

Boca Raton Documentation Team

ECL Scheduler

Boca Raton Documentation Team

2026 Version 10.2.0-1

The Ecl Scheduler

Introduction

The ECL Scheduler is a component process installed with the HPCC Systems platform. It typically starts with the platform.

An interface to the scheduler is available through ECL Watch. The ECL Scheduler interface allows you to see a list of scheduled workunits. It can also trigger an event. An Event is a case-insensitive string constant naming the event to trap.

A command line tool, *scheduleadmin* is available on the server installed in /opt/HPCCSystems/bin.

ECL Scheduling

ECL Scheduling provides a means of automating processes within ECL code or to chain processes together to work in sequence. For example, you can write ECL code that watches a landing zone for the arrival of a file, and when it arrives, sprays it to Thor, processes it, builds an index, and then adds it to a superfile.

How it Works

ECL Scheduling is event-based. The ECL Scheduler monitors a Schedule list containing registered Workunits and Events and executes any Workunits associated with an Event when that Event is triggered.

Your ECL Code can execute when an Event is triggered, or can trigger an Event. If you submit code containing a **WHEN** clause, the Event and Workunit registers in the Schedule list. When that Event triggers, the Workunit compiles and executes. When the Workunit completes, ECL Scheduler removes it from the Schedule list.

For example, if you submit a Workunit using **WHEN('Event1','MyEvent', COUNT(2))** in the appropriate place, it will execute twice (the value of **COUNT**) before the ECL Scheduler removes it from the Schedule list and the Workunit is marked as completed.

ECL Scheduler Component

Installation and configuration

The ECL Scheduler installs when you install the HPCC Systems platform. It starts and stops using hpcc-init, just as all other HPCC Systems platform components.

Using the ECL Scheduler

ECL Language Statements Used

The Following ECL Language Statements are used:

WHEN

The **WHEN** service executes the action whenever the event is triggered. The optional **COUNT** option specifies the number of events to trigger instances of the action.

NOTIFY

The **NOTIFY** action triggers the event so that the **WHEN** workflow service can proceed with operations they are assigned to execute.

EVENT

The **EVENT** function returns a trigger event, which may be used within the **WHEN** workflow service or the **NOTIFY** action. EVENT is not really a statement, rather a parameter to WHEN/NOTIFY to describe what kind of event it is used for.

CRON

The **CRON** function defines a timer event for use within the **WHEN** workflow service. This is synonymous with **EVENT('CRON', time)**. CRON itself is not a statement, rather a parameter to WHEN/NOTIFY to describe what kind of event it is used for.

WAIT

The **WAIT** function is a string constant containing the name of the event to wait for. It is used much like the **WHEN** workflow service, but may be used within conditional code.

Monitoring Functions in the Standard Library (STD.File)

MonitorFile

The **MonitorFile** function creates a file monitor job in the DFU Server for a physical file.

MonitorLogicalFileName

The **MonitorLogicalFileName** function creates a file monitor job in the DFU Server for a logical file.

DFUPlus: Monitor Option

```
    dfuplus action=monitor event=MyEvent
```

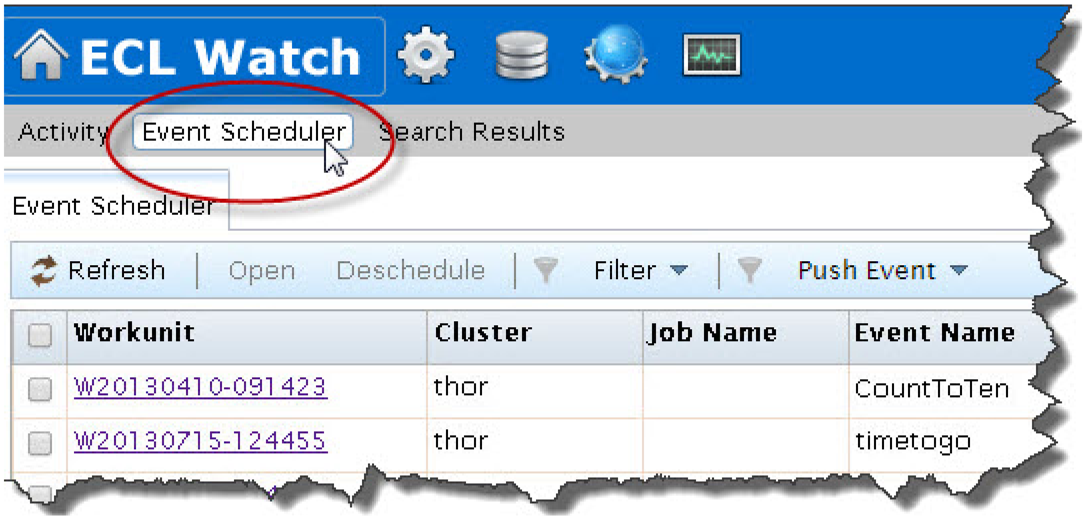
Note: DFUServer file monitoring (either using the Standard Library or DFUPlus) creates a DFU Workunit. While monitoring, the Workunit's state is *monitoring* and once it triggers the event, it is set to *finished*. You can Abort a "monitoring" DFU Workunit to stop monitoring from ECL Watch.

Interface in ECL Watch

To access the ECL Scheduler interface in ECL Watch, click on the **Event Scheduler** link in the navigation sub-menu. The Scheduler interface displays and you can see the scheduled workunits, if any.

The list of scheduled workunits has two significant columns, the **EventName** and the **EventText**.

Figure 1. ECL Scheduler Interface



	Workunit	Cluster	Job Name	Event Name
	W20130410-091423	thor		CountToTen
	W20130715-124455	thor		timetogo

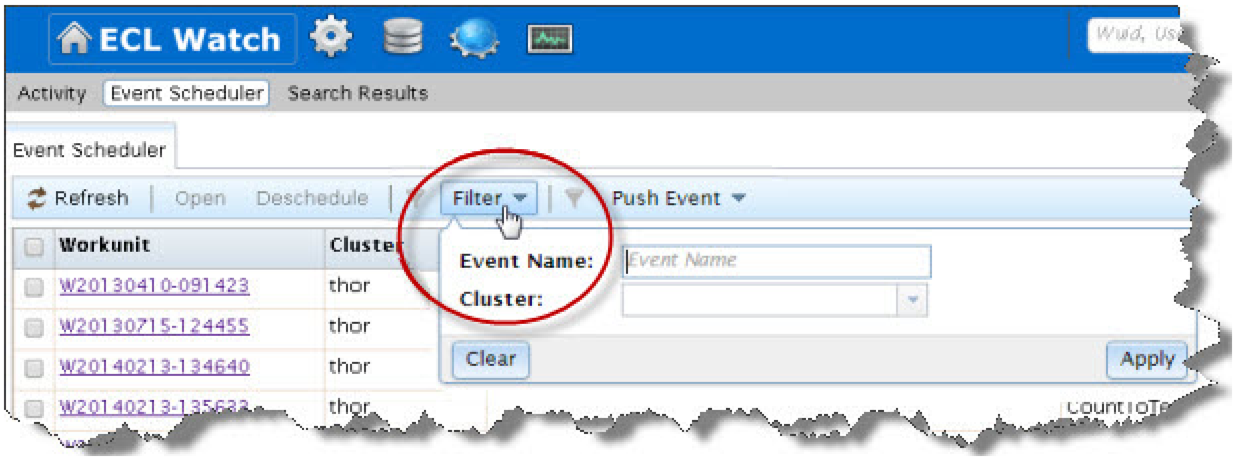
The EventName is a created when scheduling a workunit. The EventText is an accompanying sub event.

You can trigger an event by entering the EventName and Event Text in the entry boxes and then pressing the **PushEvent** button. This is the same as triggering an event using NOTIFY.

Scheduler Workunit List

You can search scheduled workunits by cluster or event name. To filter by cluster or event name, click on the **Filter** Action button. The Filter sub-menu displays. Fill in values for the filter criteria, Eventname or Cluster, then press the **Apply** button. When you specify any Filter options, the Filter Action button displays *Filter Set*.

Figure 2. Workunits in the Scheduler Interface



	Workunit	Cluster
	W20130410-091423	thor
	W20130715-124455	thor
	W20140213-134640	thor
	W20140213-135633	thor

You can sort the workunits by clicking on the column header.

To view the workunit details, click on the workunit ID (WUID) link for the workunit.

You can modify scheduled workunits from the workunit details page in ECL Watch. Select the workunit details page, then press the **Reschedule** button to reschedule a descheduled workunit. Press the **Deschedule** button to stop a selected scheduled workunit from running. You can also access the Reschedule and Deschedule options from the context menu when you right click on a workunit.

If you are using a WHEN clause and it contains a COUNT number, when rescheduled the workunit will continue the COUNT from the point where it stopped and resumes the remaining COUNT. Once a workunit completes the COUNT, there is no reschedule option.

Pushing Events

The Event Scheduler allows you to trigger or "push" an event to help manage and test your scheduled jobs.

1. Press the **PushEvent** action button.

   The Push Event dialog opens.

2. Enter the EventName:

   The EventName is a case-insensitive string constant naming the event to trap.

   See Also: EVENT

3. Enter the EventText:

   The EventText is case-insensitive string constant naming the specific type of event to trap. It may contain * and ? to wildcard-match.

   See Also: EVENT

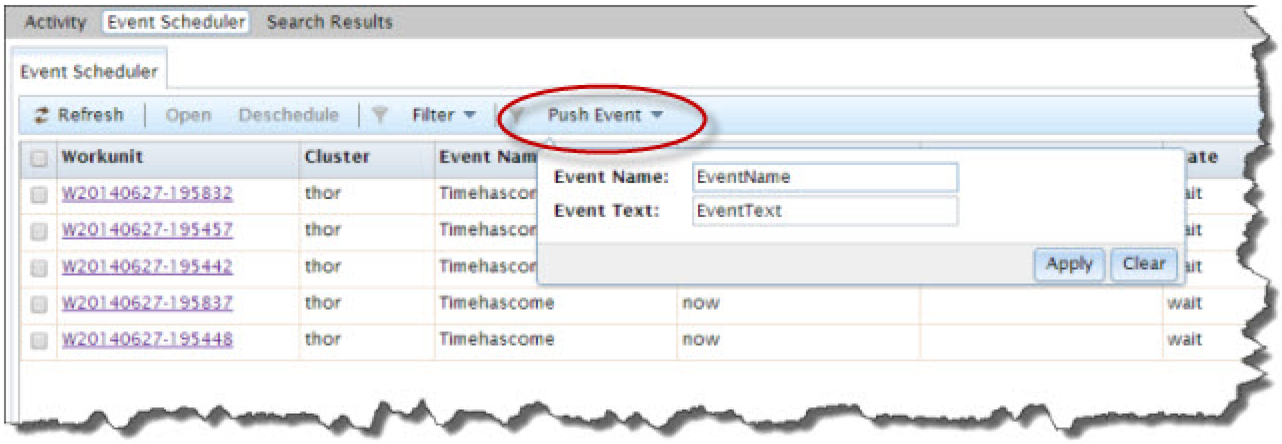
4. Press the **Apply** button

   This is the equivalent of

   ```
   NOTIFY(EVENT(EventName,EventText));
   ```

   See Also: NOTIFY, EVENT

Figure 3. PushEvent



Event Name: EventName
Event Text: EventText

Workunit	Cluster	Event Nam			ate
W20140627-195832	thor	Timehascor			ait
W20140627-195457	thor	Timehascor			ait
W20140627-195442	thor	Timehascor			ait
W20140627-195837	thor	Timehascome	now		wait
W20140627-195448	thor	Timehascome	now		wait

ECL Scheduler Admin

Command Line Interface: scheduleadmin

The **scheduleadmin** is the command line interface to the ECL Scheduler. The scheduleadmin is located by default in **/opt/HPCCSystems/bin/** on your HPCC Systems platform.

scheduleadmin *daliserver operation* **[***options* **]**

daliserver	The URL (http:// or https://) and/or IP address of the Dali server. The port may also be included.
operation	One of the following actions: servers add remove removeall list monitor cleanup push
options	Optional. A space-delimited list of optional items (listed below) appropriate to the *operation* being executed.

The **scheduleadmin** application accepts command line parameters to maintain the list of workunits the ECL Scheduler monitors.

Support Operations

The following operations are supported.

Servers

The server operation returns a list of the ECL Server queues attached to the specifed daliserver that have events being monitored.

Example:

```
      scheduleadmin 10.150.50.11:7070 servers

      //returns data that looks like this:
      eclserver_training
```

Add wuid

The add operation allows you to re-add the specified wuid after having removed it from the monitor list.

These options are used by the add operation:

wuid	A workunit identifier that contains an action with a WHEN workflow service.

Example:

```
      scheduleadmin 10.150.50.11 add W20120303-100635
```

Remove wuid

The remove operation allows you to remove the specified wuid from the monitor list.

These options are used by the remove operation:

wuid	A workunit identifier that contains an action with a WHEN workflow service.

Example:

```
     scheduleadmin 10.150.50.11 remove W20120303-100635
```

Removeall

The removeall operation allows you to remove all workunits that contain actions with WHEN workflow services from the monitor list.

Example:

```
      scheduleadmin 10.150.50.11 removeall
```

List [*eclserver* | *event*]

The list operation displays the list of monitored workunits and the events that they are waiting to occur.

These options are used by the List operation.

eclserver	The name of an ECL Server queue attached to the daliserver.
event	Optional. The name of an event. If omitted, all events are displayed.

Example:

```
      scheduleadmin 10.150.50.11 list eclserver_training

       //returns data that looks like this:
       2012-03-16T19:18:40

       CRON
          10 19 * * *
             W20120316-130812

       MyEvent
          *
            W20120316-133145
```

Monitor[*eclserver*| *event*]

The monitor operation blocks and updates the display of the list of monitored workunits as changes occur. Press the ENTER key to return to the command prompt.

These options are used by the monitor operation.

eclserver	The name of an ECL Server queue attached to the daliserver.
event	Optional. The name of an event. If omitted, all events are displayed.

Example:

```
       scheduleadmin 10.150.50.11 monitor eclserver_training

        //returns data that looks like this:
        2012-03-16T19:07:22

        CRON
           40 18 * * *
              W20120316-124216
           10 19 * * *
           W20120316-130812
        monitoring...
```

Cleanup

The cleanup operation trims unused branches from the tree list of monitored workunits.

Example:

```
       scheduleadmin 10.150.50.11 cleanup
```

Push [*eclserver*| *event*]

The push operation posts the specified event as having occurred. This allows you to "fake" an event occurrence for testing purposes.

These options are used by the push operation.

event	The name of a user-defined event (this must NOT be "CRON").
subtype	The string value to match the second parameter to the EVENT function.

Example:

```
      scheduleadmin 10.150.50.11 push MyFileEvent MyFile.d00
```

ECL Usage

The ECL Scheduler is a tool that can perform a specific action based on a specific event. The following functions can be viewed or manipulated in the scheduler.

WHEN

WHEN(*trigger, action* **[, BEFORE | SUCCESS | FAILURE])**

trigger	A dataset or action that launches the *action*.
action	The action to execute.
BEFORE	Optional. Specifies an *action* that should be executed before the input is read.
SUCCESS	Optional. Specifies an *action* that should only be executed on SUCCESS of the *trigger* (e.g., no LIMITs exceeded).
FAILURE	Optional. Specifies an *action* that should only be executed on FAILURE of the *trigger* (e.g., a LIMIT was exceeded).

The **WHEN** function associates an *action* with a *trigger* (dataset or action) so that when the *trigger* is executed the *action* is also executed. This allows job scheduling based upon triggers.

Example:

```
//a FUNCTION with side-effect Action
namesTable := FUNCTION
   namesRecord := RECORD
     STRING20 surname;
     STRING10 forename;
     INTEGER2 age := 25;
   END;
   o := OUTPUT('namesTable used by user <x>');
   ds := DATASET([{'x','y',22}],namesRecord);
   RETURN WHEN(ds,O);
END;

z := namesTable : PERSIST('z');
  //the PERSIST causes the side-effect action to execute only when the PERSIST is re-built
OUTPUT(z);
```

NOTIFY

[*attributename* := **] NOTIFY(** *event* **[**, *parm* **] [**, *expression* **])**

attributename	Optional. The identifier for this action.
event	The EVENT function, or a case-insensitive string constant naming the event to generate.
parm	A case-insensitive string constant containing the event's parameter as either a single asterisk ('*') or an XML string beginning and ending with "Event" tags and user-defined tags within those to contain the specific extra information to pass along with the event.
expression	Optional. A case-insensitive string constant allowing simple message passing, to restrict the event to a specific workunit.

The **NOTIFY** action fires the *event* so that the WAIT function or WHEN workflow service can proceed with operations they are defined to perform.

The *expression* parameter allows you to define a service in ECL that is initiated by an *event*, and only responds to the workunit that initiated it.

Example:

```
//run this first
doMyService := FUNCTION
  O := OUTPUT('Did a Service for: ' + 'EVENTNAME=' + EVENTNAME);
  N := NOTIFY(EVENT('MyServiceComplete',
                    '<Event><returnTo>FRED</returnTo></Event>'),
                    EVENTEXTRA('returnTo'));
  RETURN WHEN(EVENTEXTRA('returnTo'),ORDERED(O,N));
END;
OUTPUT(doMyService) : WHEN('MyService');
```

Then:

```
// run this in a separate workunit after the first part above completes:
NOTIFY('MyService',
       '<Event><returnTo>'+ WORKUNIT + '</returnTo></Event>');
WAIT('MyServiceComplete');
OUTPUT('WORKUNIT DONE')
```

EVENT

EVENT(*event* , *subtype* **)**

event	A case-insensitive string constant naming the event to trap.
subtype	A case-insensitive string constant naming the specific type of event to trap. This may contain * and ? to wildcard-match the event's sub-type.
Return:	EVENT returns a single event.

The **EVENT** function returns a trigger event, which may be used within the WHEN workflow service or the WAIT and NOTIFY actions.

Example:

```
IMPORT STD;
MyEventName := 'MyFileEvent';
MyFileName  := 'test::myfile';

IF (STD.File.FileExists(MyFileName),
 STD.File.DeleteLogicalFile(MyFileName));
 //deletes the file if it already exists

STD.File.MonitorLogicalFileName(MyEventName,MyFileName);
 //sets up monitoring and the event name
 //to fire when the file is found

OUTPUT('File Created') : WHEN(EVENT(MyEventName,'*'),COUNT(1));
 //this OUTPUT occurs only after the event has fired

afile := DATASET([{ 'A', '0'}], {STRING10 key,STRING10 val});
OUTPUT(afile,,MyFileName);
 //this creates a file that the DFU file monitor will find
 //when it periodically polls


//**********************************
EXPORT events := MODULE
  EXPORT dailyAtMidnight := CRON('0 0 * * *');
  EXPORT dailyAt( INTEGER hour,
   INTEGER minute=0) :=
  EVENT('CRON',
   (STRING)minute + ' ' + (STRING)hour + ' * * *');
  EXPORT dailyAtMidday := dailyAt(12, 0);
END;
BUILD(teenagers): WHEN(events.dailyAtMidnight);
BUILD(oldies)  : WHEN(events.dailyAt(6));
```

CRON

CRON(*time* **)**

time	A string expression containing a unix-standard cron time.
Return:	CRON defines a single timer event.

The **CRON** function defines a timer event for use within the WHEN workflow service or WAIT function. This is synonymous with EVENT('CRON', *time*).

The *time* parameter is unix-standard cron time, expressed in UTC (aka Greenwich Mean Time) as a string containing the following, space-delimited components:

minute hour dom month dow

minute	An integer value for the minute of the hour. Valid values are from 0 to 59.
hour	An integer value for the hour. Valid values are from 0 to 23 (using the 24 hour clock).
dom	An integer value for the day of the month. Valid values are from 1 to 31.
month	An integer value for the month. Valid values are from 1 to 12.
dow	An integer value for the day of the week. Valid values are from 0 to 6 (where 0 represents Sunday).

Any *time* component that you do not want to pass is replaced by an asterisk (*). You may define ranges of times using a dash (-), lists using a comma (,), and 'once every n' using a slash (/). For example, 6-18/3 in the hour field will fire the timer every three hours between 6am and 6pm, and 18-21/3,0-6/3 will fire the timer every three hours between 6pm and 6am.

Example:

```
EXPORT events := MODULE
  EXPORT dailyAtMidnight := CRON('0 0 * * *');
  EXPORT dailyAt( INTEGER hour,
   INTEGER minute=0) :=
  EVENT('CRON',
   (STRING)minute + ' ' + (STRING)hour + ' * * *');
  EXPORT dailyAtMidday := dailyAt(12, 0);
  EXPORT EveryThreeHours :=  CRON('0 0-23/3 * * *');
END;

BUILD(teenagers) : WHEN(events.dailyAtMidnight);
BUILD(oldies)    : WHEN(events.dailyAt(6));
BUILD(NewStuff)    : WHEN(events.EveryThreeHours);
```

WAIT

WAIT(*event***)**

event	A string constant containing the name of the event to wait for.

The **WAIT** action is similar to the WHEN workflow service, but may be used within conditional code.

Example:

```
//You can either do this:
action1;
action2 : WHEN('expectedEvent');
//can also be written as:
SEQUENTIAL(action1,WAIT('expectedEvent'),action2);
```

DFU Monitoring and Events

The following are supported methods for the ECL Scheduler included in the ECL Standard Library Reference.

MonitorFile

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

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

event	A null-terminated string containing the user-defined name of the event to fire when the *filename*appears. This value is used as the first parameter to the EVENT function.
ip	Optional. A null-terminated string containing the ip address for the file to monitor. This is typically a landing zone. This may be omitted only if the *filename*parameter contains a complete URL.
filename	A null-terminated string containing the full path to the file to monitor. This may contain wildcard characters (* and ?).
subdirs	Optional. A boolean value indicating whether to include files in sub-directories that match the wildcard mask when the *filename* contains wildcards. If omitted, the default is false.
shotcount	Optional. An integer value indicating the number of times to generate the event before the monitoring job completes. A negative one (-1) value indicates the monitoring job continues until manually aborted. If omitted, the default is 1.
espserverIPport	Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with "/FileSpray" appended.
dfuwuid	The attribute name to recieve the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the monitoring job.
Return:	fMonitorFile returns a null-terminated string containing the DFU workunit ID (DFUWUID).

The **MonitorFile** function creates a file monitor job in the DFU Server. Once the job is received it goes into a 'monitoring' mode (which can be seen in the ECL Watch DFU Workunit display), which polls at a fixed interval. This interval is specified in the DFU Server's **monitorinterval** configuration setting. The default interval is 900 seconds (15 minutes). If an appropriately named file arrives in this interval it will fire the *event* with the name of the triggering object as the event subtype (see the EVENT function).

This process continues until either:

1) The *shotcount* number of events have been generated.

2) The user aborts the DFU workunit.

The STD.File.AbortDfuWorkunit and STD.File.WaitDfuWorkunit functions can be used to abort or wait for the DFU job by passing them the returned *dfuwuid*.

Note the following caveats and restrictions:

1) Events are only generated when the monitor job starts or subsequently on the polling interval.

2) Note that the *event* is generated if the file has been created since the last polling interval. Therefore, the *event* may occur before the file is closed and the data all written. To ensure the file is not subsequently read before it is complete you should use a technique that will preclude this possibility, such as using a separate 'flag' file instead of the file, itself or renaming the file once it has been created and completely written.

3) The EVENT function's subtype parameter (its 2nd parameter) when monitoring physical files is the full URL of the file, with an absolute IP rather than DNS/netbios name of the file. This parameter cannot be retrieved but can only be used for matching a particular value.

Example:

```
EventName := 'MyFileEvent';
FileName  := 'c:\\test\\myfile';
LZ := '10.150.50.14';
STD.File.MonitorFile(EventName,LZ,FileName);
OUTPUT('File Found') : WHEN(EVENT(EventName,'*'),COUNT(1));
```

MonitorLogicalFileName

STD.File.MonitorLogicalFileName(*event,* *filename,* **[** *, shotcount* **] [** *, espserverIPport* **])**

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

event	A null-terminated string containing the user-defined name of the event to fire when the *filename* appears. This value is used as the first parameter to the EVENT function.
filename	A null-terminated string containing the name of the logical file in the DFU to monitor.
shotcount	Optional. An integer value indicating the number of times to generate the event before the monitoring job completes. A negative one (-1) value indicates the monitoring job continues until manually aborted. If omitted, the default is 1.
espserverIPport	Optional. This should almost always be omitted, which then defaults to the value contained in the lib_system.ws_fs_server attribute. When not omitted, it should be a null-terminated string containing the protocol, IP, port, and directory, or the DNS equivalent, of the ESP server program. This is usually the same IP and port as ECL Watch, with "/FileSpray" appended.
dfuwuid	The attribute name to recieve the null-terminated string containing the DFU workunit ID (DFUWUID) generated for the monitoring job.
Return:	fMonitorLogicalFileName returns a null-terminated string containing the DFU workunit ID (DFUWUID).

The **MonitorLogicalFileName** function creates a file monitor job in the DFU Server. Once the job is received it goes into a 'monitoring' mode (which can be seen in the eclwatch DFU Workunit display), which polls at a fixed interval (default 15 mins). If an appropriately named file arrives in this interval it will fire the *event* with the name of the triggering object as the event subtype (see the EVENT function).

This function does not support wildcard characters. To monitor physical files or directories using wildcards, use the MonitorFile function.

This process continues until either:

1) The *shotcount* number of events have been generated.

2) The user aborts the DFU workunit.

The STD.File.AbortDfuWorkunit and STD.File.WaitDfuWorkunit functions can be used to abort or wait for the DFU job by passing them the returned *dfuwuid*.

Note the following caveats and restrictions:

1) If a matching file already exists when the DFU Monitoring job is started, that file will not generate an event. It will only generate an event once the file has been deleted and recreated.

2) If a file is created and then deleted (or deleted then re-created) between polling intervals, it will not be seen by the monitor and will not trigger an event.

3) Events are only generated on the polling interval.

Example:

```
EventName := 'MyFileEvent';
FileName  := 'test::myfile';
```

```
IF (STD.File.FileExists(FileName),
         STD.File.DeleteLogicalFile(FileName));
STD.File.MonitorLogicalFileName(EventName,FileName);
OUTPUT('File Created') : WHEN(EVENT(EventName,'*'),COUNT(1));

rec := RECORD
  STRING10 key;
  STRING10 val;
END;
afile := DATASET([{ 'A', '0'}], rec);
OUTPUT(afile,,FileName);
```