Executor Library for Mosel

9.7

REFERENCE MANUAL

FICO® Xpress Optimization

FICO

FICO® Xpress Optimization 9.7

Deliverable Version: A

Last Revised: 12 December, 2024

Contents

CHAPTER 1

Introduction

1.1 Xpress Executor 3 in FICO Decision Management Platform

FICO Xpress Executor version 3 is a component that allows users to deploy optimization as a service within the FICO Decision Management Platform (DMP). The component owner can configure the Xpress Executor 3 component either with a fixed Mosel model, or to accept optimization problems in standard LP or XMPS formats, and the optimizations will be executed asynchronously in response to SOAP or REST webservice requests. For more information about Xpress Executor 3 and the FICO Decision Management Platform, see

www.ficoanalyticcloud.com

This document assumes that the reader is a registered user of the FICO Analytic Cloud and has access to at least one Xpress Executor component. For general information about DMP solutions, components, lifecycle stages and more, please consult the Decision Management Platform help by clicking the "Help" link on the DMP website. Below are summary definitions of some DMP terminology used in this document, but please consult the DMP help for full information.

- **Solution** - a named collection of Components and Services.
- **Component** - a component provides capabilities that can be added to solutions. Components typically expose a UI that can be interacted with directly, or webservices that can be called from an external application.
- **Component Lifecycle** - a component has three lifecycle stages - design, staging and production. You will edit your configuration in the design stage, submit the configuration to staging for testing, and deploy to production once you are satisfied. There is a different webservice URL for interacting with each lifecycle stage of a component.
- **DMP Manager** - the web site that allows you to manage your solutions and components.

1.2 Executor package for Mosel

The package *executor* allows the user to send commands to an Xpress Executor version 3 component running in DMP, from either a local Mosel process or from an Xpress Insight component in the same DMP solution. The package can be used to run model executions in the Xpress Executor component, as well as modify the Xpress Executor 3 design instance configuration to a limited extent.

The `executor` package cannot be used with Xpress Executor version 4 or any later versions.

The package does not support direct promotion of configuration to staging or production instances; this must be done by submitting, approving and then deploying the configuration throught the DMP UI.

CHAPTER 2

Basic Principles

The *executor* library should be loaded into your model or package as follows:

```
uses "executor"
```

When you want to interact with an Xpress Executor 3 component instance (design, staging or production), you should declare a variable of the 'Executor' type to represent the Executor component instance.

```
declarations
  myexecutor: Executor
end-declarations
```

Once the 'Executor' variable has been initialized with your component's location and credentials (see chapter 3), it can be used with Executor functions such as `executorexecute`, `executorwaitfor` and `executorfetchresult`. If you need to access more than one Executor component instance, you can declare multiple Executor variables.

After calling each function on the *executor* library, you should check the value of `Executor.status` for any errors - if this is any value other than `EXECUTOR_OK` then an error has occurred. In an error case, `Executor.status` will contain an error code and `Executor.lasterror` will return a description of the error.

CHAPTER 3

Authenticating

3.1 Using Executor Properties

This section assumes you have an Executor component available in your active FICO DMP Solution. Open the Library and select your Solution containing your Executor component.

The simplest way to specify the Executor component URL and authorization credentials is to define the corresponding properties of your Executor entity, for example:

```
model DirectInitExample
  uses "executor"
  declarations
    myexecutor: Executor
  end-declarations

  myexecutor.componenturl := "https://vm65j75lqh-vm65j75lqh.dms.usw2.ficoanalyticcloud.com/"
  myexecutor.bearertokenurl := "https://iam-svc.dms.usw2.ficoanalyticcloud.com/registration/rest/client/to
  myexecutor.clientid := "vm5qh0y37c"
  myexecutor.secret := "wdS97u648BVoI#d3e4g2Z4mP780Y1DLdSDKm"
  ! myexecutor now initialized and can be used
end-model
```

To find the component URL, navigate to your DMP Solution and open the drop-down menu by clicking the chevron beside the component name and select "View Links". Use the value from the 'REST' field but remove everything after the domain name. For example, if the link displayed is:

`https://vm65j75lqh-vm65j75lqh.dms.usw2.ficoanalyticcloud.com/rest/runtime/execution?solutionID=vm5qh0y37`

then use

`https://vm65j75lqh-vm65j75lqh.dms.usw2.ficoanalyticcloud.com/`

The clientid and secret values for your solution can be found on the "Client Apps" page of the solution screen in DMP. Finally, the `bearertokenurl` will be the URL from where we will obtain DMP authentication tokens - consult the "Requesting a Bearer Token" page of the DMP documentation for more details.

If you prefer to manage the authentication with DMP yourself, you can alternately initialize the Executor with the component URL and a DMP bearer token, e.g.:

```
model DirectInitExample
  uses "executor"
  declarations
    myexecutor: Executor
  end-declarations

  myexecutor.componenturl := "https://vm65j75lqh-vm65j75lqh.dms.usw2.ficoanalyticcloud.com/"
  myexecutor.bearertoken := "xyfuehdsoguihUGHifsihf7hi"
  ! myexecutor now initialized and can be used
```

```
end-model
```

However, note you will need to refresh the 'bearertoken' value periodically as the tokens will expire. If you initialize with clientid and secret, the 'executor' module will request new bearer tokens when it needs to.

3.2 Using a JSON Configuration File

As an alternative to setting credentials in the model, you can specify them in a JSON document, the contents of which you assign to the `executor_components` parameter. The document should be in the following format:

```
{
  "<component-id>": {
    "componentUrl": "<URL of component>",
    "bearerTokenUrl": "<URL of DMP authentication service, optional>",
    "clientId": "<clientId of DMP solution, optional>",
    "secret": "<secret of DMP solution, optional>",
    "bearerToken": "<DMP bearer token, optional>"
  }
}
```

The `"<component-id>"` string is a key that you use to refer to the component in the JSON and has no other meaning. You can specify multiple components in the same JSON file so long as they have different `"<component-id>"` strings, e.g.:

```
{
  "main": {
    "componentUrl": "https://vm65j75lqh-vm65j75lqh.dms.usw2.ficoanalyticcloud.com/",
    "bearerTokenUrl": "https://iam-svc.dms.usw2.ficoanalyticcloud.com/registration/rest/client/token",
    "clientId": "vm5qh0y37c",
    "secret": "wdS97u648BVoI#d3e4g2Z4mP780Y1DLdSDKm"
  },
  "dev": {
    "componentUrl": "https://6tk0a8qheb-6tk0a8qheb.dms.usw2.ficoanalyticcloud.com/",
    "bearerTokenUrl": "https://iam-svc.dms.usw2.ficoanalyticcloud.com/registration/rest/client/token",
    "clientId": "vm5qh0y37c",
    "secret": "wdS97u648BVoI#d3e4g2Z4mP780Y1DLdSDKm"
  }
}
```

The JSON document must always specify the `componentUrl`, and either the `bearerToken` or all of the `bearerTokenUrl`, `clientId` and `secret` values. If you do specify your own bearer token, be aware that it will eventually expire.

Then you can initialize an `Executor` in the model with the credentials from the JSON by calling the `executorinit` procedure. For example, if you've saved the above sample JSON in a file called "executors.json", then you can do:

```
model InitExample
  uses "executor","mmsystem"
  declarations
    public jsoncfg: text
    myexecutor: Executor
  end-declarations

  ! Load executors.json into a variable so it can be passed to a parameter
  fcopy("executors.json","text:jsoncfg")
  setparam("executor_components",string(jsoncfg))

  ! Initialize myexecutor using the 'main' set of credentials
  executorinit(myexecutor, "main")
```

```
  if myexecutor.status<>EXECUTOR_OK then
    writeln("Executor initialization error: ", myexecutor.lasterror)
    exit(1)
  end-if
  ! myexecutor now initialized and can be used
end-model
```

The `executor_components` parameter is special in that it has a single value shared by all models within the Mosel instance - this means that if, for example, you set it for your master model, then the same value will be used for all submodels that you start in the same Mosel process.

3.3 Using from Xpress Insight in DMP

When using Xpress Insight within DMP, the Mosel instance will automatically be configured to access any Xpress Executor components in the same solution. To use this, call `executorinit`, passing the name of your Xpress Executor component:

```
model DmpInitExample
  uses "executor"
  declarations
    myexecutor: Executor
  end-declarations

  ! Initialize myexecutor for the component called "My Executor"
  executorinit(myexecutor, "My Executor")
  if myexecutor.status<>EXECUTOR_OK then
    writeln("Executor initialization error: ", myexecutor.lasterror)
    exit(1)
  end-if
  ! myexecutor now initialized and can be used
end-model
```

Alternatively, if you call `executorinit` without a name, it will access the Xpress Executor component it finds in your solution:

```
model DmpInitExample
  uses "executor"
  declarations
    myexecutor: Executor
  end-declarations

  ! Initialize myexecutor for the an Xpress Executor 3 component we find
  executorinit(myexecutor)
  if myexecutor.status<>EXECUTOR_OK then
    writeln("Executor initialization error: ", myexecutor.lasterror)
    exit(1)
  end-if
  ! myexecutor now initialized and can be used
end-model
```

The Insight component instance will access Executor instances in the same lifecycle stage (for example, an Insight in the staging environment will access the staging environment of Executor, not production or design). You can override this by specifying the environment on the 'executorinit' line, for example:

```
model DmpInitExample
  uses "executor"
  declarations
    myexecutor: Executor
  end-declarations

  ! Initialize myexecutor for the an Xpress Executor 3 component we find
  executorinit(myexecutor,"My Executor","STAGING")
  if myexecutor.status<>EXECUTOR_OK then
```

```
    writeln("Executor initialization error: ", myexecutor.lasterror)
    exit(1)
  end-if
  ! myexecutor now initialized and can be used
end-model
```

This method cannot be used to access the Executor component from Xpress Insight 5. Instead, an Insight 5 app should call DMP Manager webservices to find the Executor component URL, and set the `componenturl` and `bearertoken` properties as described in the section 3.1.

3.4 Using local development mode

An `Executor` can also be initialized in a 'local' mode that does not require a DMP component. In this configuration, Executor will run any submitted executions locally using Mosel submodels. This can be configured as follows:

```
model LocalInitExample
  uses "executor"
  declarations
    myexecutor: Executor
  end-declarations

  ! Initialize myexecutor to simulate a Xpress Executor 3 component using local submodels
  executorinitlocal(myexecutor)
  if myexecutor.status<>EXECUTOR_OK then
    writeln("Executor initialization error: ", myexecutor.lasterror)
    exit(1)
  end-if
  ! myexecutor now initialized and can be used
end-model
```

After initialization, the 'local' executor will always have an empty configuration, so you should call `executorsetmodelfile` and `executorsetexecmode` to configure it.

Once configured, you can interact with the local execution using the same functions as you would use for a true Xpress Executor 3 component (e.g. `executorexecute`, `executorwaitfor`, `executorfetchresult`, ...). Local execution is supplied as an aid in developing your model and there may be small differences between how your model reacts to errors locally and how it behaves in a real Xpress Executor 3.

Note that the local Xpress Executor 3 does not support the MPS or LP file execution mode, and the progress field of the ModelExecution record will always contain 0 regardless of the progress events emitted by your models.

CHAPTER 4

Usage example

This example demonstrates running an execution in an already-configured Xpress Executor 3 component.

You will need to fill in the model parameters with your Xpress Executor 3 component details.

```
model ExecutorExecuteExample
uses "executor"

parameters
  COMPONENT_URL = ""
  BEARER_TOKEN_URL = ""
  CLIENT_ID = ""
  SECRET = ""
end-parameters

declarations
  INPUT_FILE="MyExecutionInputFile.dat"
  RESULT_FILE="MyExecutionResultFile.dat"
  myexecutor: Executor
  myexecution: ModelExecution
  EXECUTION_TIMEOUT = 10*60  ! Timeout in seconds
end-declarations

! Configure 'myexecutor' with our Xpress Executor 3 component details
myexecutor.componenturl := COMPONENT_URL
myexecutor.bearertokenurl := BEARER_TOKEN_URL
myexecutor.clientid := CLIENT_ID
myexecutor.secret := SECRET

! Start execution passing local input data
myexecution := executorexecute(myexecutor,INPUT_FILE)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error from Executor when starting execution: ",myexecutor.lasterror)
  exit(1)
end-if

! Wait for execution to complete
executorwaitfor(myexecutor,myexecution,EXECUTION_TIMEOUT)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error from Executor when waiting for completion: ",myexecutor.lasterror)

! If execution did not complete within time limit, display error
elif not myexecution.iscompleted then
  writeln("Execution failed to complete within ",EXECUTION_TIMEOUT,"s")

! If execution completed with error, display error
elif myexecution.status<>EXECUTION_STATUS_OK or myexecution.exitcode<>0 then
  writeln("Execution failed with status ",myexecution.status," and exit code ",myexecution.exitcode)
  ! In error cases, execution run log will contain valuable information
  executorfetchrunlog(myexecutor,myexecution,"runlog.txt")
  if myexecutor.status<>EXECUTOR_OK then
    writeln("Error from Executor when fetching run log: ",myexecutor.lasterror)
  else
```

```
    writeln("Run log saved in file runlog.txt")
  end-if

! If execution completed ok, download result
else
  ! Completed okay, download results
  executorfetchresult(myexecutor,myexecution,RESULT_FILE)
  if myexecutor.status<>EXECUTOR_OK then
    writeln("Error from Executor when fetching results: ",myexecutor.lasterror)
  else
    writeln("Result data saved in file ",RESULT_FILE)
  end-if
end-if


! Delete execution from server
executordelete(myexecutor,myexecution)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error from Executor when deleting execution: ",myexecutor.lasterror)
  exit(1)
end-if

end-model
```

Chapter 5

Types

ModelExecution : `record`

Record type representing information about the status of an execution in an Xpress Executor 3 component

id : `text`

The unique identifer of the execution

iscompleted : `boolean`

Whether the execution has completed (successfully or with errors), or not (not started or currently executing)

hasinput : `boolean`

Whether the execution has input data.

hasresult : `boolean`

Whether the execution has result data.

isexecuting : `boolean`

Whether the execution is currently running (not completed and not waiting to start)

executionnode : `text`

The name of the gear on which the execution was executed

progress : `real`

Progress value as set by the model being executed. Always 0 in local mode.

status : `string`

Status of the execution. One of: `EXECUTION_STATUS_OK`, `EXECUTION_STATUS_MATHERR`, `EXECUTION_STATUS_ERROR`, `EXECUTION_STATUS_IOERR`, `EXECUTION_STATUS_NIFCT`, `EXECUTION_STATUS_NULL`, `EXECUTION_STATUS_LICERR`, `EXECUTION_STATUS_STOP`, `EXECUTION_STATUS_LOAD_ERROR`, `EXECUTION_STATUS_INTERNAL_ERROR`, `EXECUTION_STATUS_NOT_COMPLETED`

exitcode : `integer`

The exit code of the model - the value passed to the Mosel exit() procedure, or 0. This will always be 0 when execution status is not EXECUTION_STATUS_OK.

creationtimestamp : `real`

Timestamp when the execution was created, expressed as milliseconds since 1/1/1970 00:00 UTC

starttimestamp : `real`

Timestamp when the execution started to execute, expressed as milliseconds since 1/1/1970 00:00 UTC, or 0 if unset

finishtimestamp : `real`

Timestamp when the execution finished executing, expressed as milliseconds since 1/1/1970 00:00 UTC, or 0 if unset

modelparams : `dynamic array(ExecutorParamNames) of text`

Additional model parameters used with this execution

ModelInterruptionConfig : `record`

Record type representing the Model interruption configuration of the Xpress Executor component. This controls how a model is interrupted should Executor need to interrupt it before it completes.

minruntimeseconds : `integer`

The minimum number of seconds to allow a model to run before it is interrupted. If Executor needs to interrupt a model, it will wait until it has run for `minruntimeseconds` before trying to interrupt. Useful where the model is short but interruption would terminate the Mosel instance and the Mosel instance setup is expensive.

MoselInstanceConfig : `record`

This type definition is deprecated and will be removed in a future release. New solution should use `MoselInstanceConfigExt` instead

Record type representing the Mosel instance (process) re-use policy configured in the Xpress Executor component, containing only the attributes supported in Xpress 8.11. This is a legacy type included for backwards-compatibility reasons only.

reuse : `boolean`

If false, every execution will run in a separate Mosel process. If true, Xpress Executor will re-use a Mosel process after the execution completes, in some cases. In neither case will Xpress Executor run two executions in the same Mosel process at the same time.

maxexecutions : `integer`

When reuse is true, this value limits maximum number of executions that can be run using each Mosel process before it is automatically terminated.

reuseaftererror : `boolean`

If reuse is true, and this flag is true, a Mosel process can be re-used even if a previous execution in the same process terminated with an error code.

reuseafternonzeroexit : `boolean`

If reuse is true, and this flag is true, a Mosel process can be re-used even if a previous execution in the same process terminated with a non-zero exit code.

MoselInstanceConfigExt : `record`

Record type representing the Mosel instance (process) re-use policy configured in the Xpress Executor component

reuse : `boolean`

If false, every execution will run in a separate Mosel process. If true, Xpress Executor will re-use a Mosel process after the execution completes, in some cases. In neither case will Xpress Executor run two executions in the same Mosel process at the same time.

maxexecutions : `integer`

When reuse is true, this value limits maximum number of executions that can be run

using each Mosel process before it is automatically terminated.

reuseaftererror : `boolean`

If reuse is true, and this flag is true, a Mosel process can be re-used even if a previous execution in the same process terminated with an error code.

reuseafternonzeroexit : `boolean`

If reuse is true, and this flag is true, a Mosel process can be re-used even if a previous execution in the same process terminated with a non-zero exit code.

reuseaftercancel : `boolean`

If reuse is true, and this flag is true, a Mosel process can be re-used even if a previous execution in the same process interrupted before completion.

javaversion : `string`

The version of Java to be used by the 'mosjvm' module when used by Mosel within Executor. Supported values are 'JAVA8' and 'JAVA11'.

RepositoryStatus : `record`

Record type representing a summary of the Xpress Executor 3 component's repository status

componentid : `string`

The ID string of the Xpress Executor 3 DMP component

componentenv : `string`

The component environment / lifecycle stage, either "design", "staging" or "production", or "LOCAL" in local mode.

numgears : `integer`

The number of gears assigned to the DMP component in this environment

numexecutions : `integer`

The total number of executions currently in the repository

numqueued : `integer`

The number of executions in the repository that have not yet started to execute

numinprogress : `integer`

The number of executions in the repository that are currently executing

numcompleted : `integer`

The number of executions in the repository that have completed (either successfully or with errors)

binarydatasize : `integer`

The approximate number of megabytes of binary data stored in the database

binarydataquota : `integer`

The maximum number of megabytes of binary data that may be stored in the database

dbsize : `integer`

The current size of the database, in megabytes

RunLogFragment : `record`

Record type representing part of the run log of a model execution

id : `text`

The ID of the execution in the Xpress Executor component

logversion : `text`

A version string for the run log. When Executor restarts an execution from the beginning, the run log version will change.

firstlinenum : `integer`

The index of the first line in this fragment of the run log, where the first line of the run log has index 1.

numavailablelines : `integer`

The total number of lines in the run log.

lines : `list of text`

Individual lines in this fragment of the run log.

CHAPTER 6

Variables

ExecutorParamNames : `set of string`

Set of all parameter names used in model executions

Chapter 7

Subroutines

executordelete

Purpose

Deletes an execution from the Xpress Executor 3 component

Synopsis

```
procedure executordelete(exec:Executor, execution:ModelExecution)
```

Arguments

`exec`	The Executor on which the execution resides
`execution`	The execution to delete

Example

```
executordelete( myexecutor, myexecution )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The procedure will not return until the execution has been marked for deletion or an error was detected.
3. Only the `id` field of the ModelExecution record is read by this procedure.
4. If the execution is still running, Xpress Executor will cancel it.
5. Once deleted, the execution result, status and run log will not be available.

Related topics

`getstatus`

executordeleteall

Purpose

Deletes all executions from the Xpress Executor 3 component

Synopsis

```
procedure executordeleteall(exec:Executor)
```

Argument

`exec` The Executor with which to interact

Example

```
executordeleteall( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The procedure will not return until all executions have been marked for deletion or an error was detected.
3. If any executions are still running, Xpress Executor will cancel them.
4. Once deleted, the execution result, status and run log will not be available.

Related topics

`getstatus`

executorexecute

Purpose

Submits a new execution to the Xpress Executor 3 component.

Synopsis

```
function executorexecute(exec:Executor, inputfname:text,
      modelparams:array(set of string) of text):ModelExecution
function executorexecute(exec:Executor, inputfname:text):ModelExecution
```

Arguments

`exec`	The Executor with which to interact
`inputfname`	The filename containing the input data to submit to the execution. If an empty string, execution will be submitted with an empty input data file. May reference a Mosel I/O driver, e.g. `"mmsystem.text:myinputdata"`
`modelparams`	Values of any additional model parameter to set for this execution. Module parameters such as `mmxprs.xprs_verbose` may not be set using this array.

Return value

A record containing the execution id and the initial status.

Example

```
myexecution := executorexecute( myexecutor, "inputdata.dat" )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The procedure will not return until the execution has been submitted to the component or an error was detected.
3. Once submitted, the execution will be executed asynchronously. Use `executorpoll` or `executorwaitfor` to check the execution status.
4. Once deleted, the execution result, status and run log will not be available.

Related topics

`getstatus`, `executordelete`, `executorwaitfor`, `executorpoll`, `executorfetchinput`, `executorfetchresult`, `executorfetchrunlog`

executorfetchinput

Purpose

Downloads the input data file of the given execution

Synopsis

```
procedure executorfetchinput(exec:Executor, execution:ModelExecution,
       dstfile:text)
```

Arguments

`exec`	The Executor on which the execution resides
`execution`	The execution whose input we should request
`dstfile`	The filename to which to save the input.

Example

```
executorfetchinput( myexecutor, myexecution, "text:inputdata" )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The procedure will not return until the execution input data has been downloaded or an error was detected.
3. Only the `id` field of the ModelExecution record is read by this procedure.

Related topics

`getstatus`, `executorexecute`

executorfetchresult

Purpose

Downloads the result data file of the given execution

Synopsis

```
procedure executorfetchresult(exec:Executor, execution:ModelExecution,
      dstfile:text)
```

Arguments

`exec`	The Executor on which the execution resides
`execution`	The execution whose result we should request
`dstfile`	The filename to which to save the result.

Example

```
executorfetchresult( myexecutor, myexecution, "text:resultdata" )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The procedure will not return until the execution result data has been downloaded or an error was detected.
3. Only the `id` field of the ModelExecution record is read by this procedure.
4. The execution results will not be available until the execution completes (`ModelExecution.iscompleted` is true)

Related topics

`getstatus`, `executorwaitfor`

executorfetchrunlog

Purpose

Downloads the run log of the given execution

Synopsis

```
procedure executorfetchrunlog(exec:Executor, execution:ModelExecution,
      dstfile:text)
```

Arguments

`exec`	The Executor on which the execution resides
`execution`	The execution whose run log we should request
`dstfile`	The filename to which to save the run log

Example

```
executorfetchrunlog( myexecutor, myexecution, "text:runlog" )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The procedure will not return until the execution run log has been downloaded or an error was detected.
3. Only the `id` field of the ModelExecution record is read by this procedure.
4. If the execution has not completed, this procedure will download whatever run log it has output so far.

Related topics

`getstatus` `executorfetchrunlogfragment`

executorfetchrunlogfragment

Purpose

Downloads a fragment of the run log of the given execution.

Synopsis

```
function executorfetchrunlogfragment(exec:Executor,
      execution:ModelExecution, firstline:integer,
      maxlines:integer):RunLogFragment
```

Arguments

`exec`	The Executor on which the execution resides
`execution`	The execution whose run log we should request
`firstline`	The index of the first line of the run log to fetch, where the first line of the run log has index 1
`maxlines`	The maximum number of lines to return in the fragment

Example

```
frag := executorfetchrunlogfragment( myexecutor, myexecution, 10, MAX_INT )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
forall (line in frag.lines) do
  writeln(line)
end-do
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The procedure will not return until the execution run log fragment has been downloaded or an error was detected.
3. Only the `id` field of the ModelExecution record is read by this function.
4. If the execution has not completed, this procedure will download a fragment based on however much of the run log is available.
5. When passing a value of `firstline` greater than 1, always check the `logversion` of the returned fragment; if this value is different from the `logversion` of previously fetched fragments then the execution has been restarted and you should re-fetch the run log from line 1.

Related topics

`getstatus` `executorfetchrunlog`

executorgetconcurrencylevel

Purpose

Fetch the currently configured concurrency level of the Xpress Executor 3 component.

Synopsis

```
function executorgetconcurrencylevel(exec:Executor):integer
```

Argument

`exec` The Executor to query

Return value

The number of executions that can be performed concurrently on a single Xpress Executor gear

Example

```
declarations
  concurrencylevel: integer
end-declarations
concurrencylevel := executorgetconcurrencylevel( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. ”Concurrency Level” is the number of executions that can be performed concurrently on a single gear; if the Xpress Executor component is scaled over multiple gears then the total number of concurrent executions will be concurrencylevel*numgears
3. If you start an execution when the maximum number of executions are concurrently executing, new executions are queued and will start when other executions complete.
4. This will always be 1 if the Executor is configured in local mode.

Related topics

`getstatus` `executorsetconcurrencylevel`

executorgetexectimeout

Purpose

Fetch the currently configured execution timeout of the Xpress Executor 3 component.

Synopsis

```
function executorgetexectimeout(exec:Executor):integer
```

Argument

`exec` The Executor to query

Return value

The number of seconds of runtime after which an execution may be stopped.

Example

```
declarations
  timeout: integer
end-declarations
timeout := executorgetexectimeout( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. If a model runs for longer than the configured execution timeout, Xpress Executor may interrupt it and return the execution as status STOPPED.
2. This will always be MAX_INT if the Executor is configured in local mode.

Related topics

`getstatus` `executorsetexectimeout`

executorgetexpiryhours

Purpose

Fetch the configured execution expiry time of the Xpress Executor 3 component.

Synopsis

```
function executorgetexpiryhours(exec:Executor):integer
```

Argument

`exec` The Executor to query

Return value

The number of hours after which a completed execution may be automatically deleted.

Example

```
declarations
  expiryhours: integer
end-declarations
expiryhours := executorgetexpiryhours( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. This will always be MAX_INT if the Executor is configured in local mode.

Related topics

`executordelete` `executorsetexpiryhours`

executorgetinstanceconfig

Purpose

This subroutine is deprecated and will be removed in a future release. Use `executorgetinstanceconfigext` instead.
Fetch the Mosel instance re-use configuration of the Xpress Executor component. Legacy version included for compatibility with Xpress 8.11; use `executorgetinstanceconfigext` instead.

Synopsis

```
function executorgetinstanceconfig(exec:Executor):MoselInstanceConfig
```

Argument

`exec` The Executor to query

Return value

A record describing in what circumstances (if any) Xpress Executor will re-use Mosel processes for multiple executions

Example

```
declarations
  cfg: MoselInstanceConfig
end-declarations
cfg := executorgetinstanceconfig( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

After calling, check the value of `exec.status` for any errors.

Related topics

`getstatus` `executorsetinstanceconfig`

executorgetinstanceconfigext

Purpose

Fetch the Mosel instance re-use configuration of the Xpress Executor 3 component.

Synopsis

```
function executorgetinstanceconfigext(exec:Executor):MoselInstanceConfigExt
```

Argument

`exec` The Executor to query

Return value

A record describing in what circumstances (if any) Xpress Executor will re-use Mosel processes for multiple executions

Example

```
declarations
  cfg: MoselInstanceConfigExt
end-declarations
cfg := executorgetinstanceconfigext( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

After calling, check the value of `exec.status` for any errors.

Related topics

`getstatus` `executorsetinstanceconfig`

executorgetinterruptconfig

Purpose

Fetch the model interruption configuration of the Xpress Executor 3 component.

Synopsis

```
function executorgetinterruptconfig(exec:Executor):ModelInterruptionConfig
```

Argument

`exec` The Executor to query

Return value

Configuration of how Xpress Executor will interrupt executing Mosel models, when necessary

Example

```
declarations
  cfg: ModelInterruptionConfig
end-declarations
cfg := executorgetinterruptconfig( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

After calling, check the value of `exec.status` for any errors.

Related topics

`getstatus` `executorsetinterruptconfig`

executorgetmodelparams

Purpose

Fetch the model parameters of the Xpress Executor 3 component.

Synopsis

```
procedure executorgetmodelparams(exec:Executor, modelparameters:array(set
      of string) of text)
```

Arguments

`exec`	The Executor to query
`modelparameters`	Array into which the model parameters should be written

Example

```
declarations
  myparams: array(set of string) of text
end-declarations
executorgetmodelparams( myexecutor, myparams)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The parameters returned by this procedure will be passed to the model for all subsequent executions with this Executor.
3. Where named model parameters are set both here and in `executorexecute`, the values passed to `executorexecute` shall have priority.

Related topics

`getstatus` `executorsetmodelparams`

executorinit

Purpose

Initialize the given Executor with the details of the indicated component configured in `executor_components`, or in the same DMP solution.

Synopsis

```
procedure executorinit(exec:Executor)
procedure executorinit(exec:Executor, name:text)
procedure executorinit(exec:Executor, name:text, env:text)
```

Arguments

`exec`	The Executor to initialize
`name`	The component ID in the configuration JSON, or the component name when running within DMP. Within DMP, empty string to find any Xpress Executor component in the solution.
`env`	The component lifecycle stage (”DESIGN”,”STAGING” or ”PRODUCTION”) when the component name when running within DMP, empty string to use the lifecycle stage of the current component.

Example

```
executorinit( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. When no component name is given, and the solution a single Xpress Executor component, that component will be used.
3. When no component name is given, and the solution contains multiple Executor components, an arbitrary one will be chosen.
4. When used from Xpress Insight running in DMP, this will initialize the Executor with the indicated Xpress Executor found within the solution containing the Insight component.

Related topics

`getstatus`, `executor_components`

executorinitlocal

Purpose

Initialize the given Executor in local mode.

Synopsis

```
procedure executorinitlocal(exec:Executor)
```

Argument

`exec` The Executor to initialize

Example

```
executorinitlocal( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. In local mode, submodels are used to simulate an Xpress Executor 3 component.
3. After initializing, the local Executor will have no configuration, so call `executorsetexecmode` and `executorsetmodelfile` to supply some.

Related topics

`getstatus`, `executorinit`, `executor_local_workdir`

executorlist

Purpose

Request a list of all the model executions currently present in the Xpress Executor 3 component

Synopsis

```
function executorlist(exec:Executor):list of ModelExecution
```

Argument

`exec` The Executor to initialize

Return value

a list of the id and status of every execution in the repository.

Example

```
executions := executorlist( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

After calling, check the value of `exec.status` for any errors.

Related topics

`getstatus`, `executorpoll`

executorpoll

Purpose

Update the status of the given ModelExecution

Synopsis

```
procedure executorpoll(exec:Executor, execution:ModelExecution)
```

Arguments

`exec`	The Executor to on which the execution resides
`execution`	The ModelExecution record to update

Example

```
executorpoll( myexecutor, myexecution )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. Only the `id` field of the ModelExecution record is read by this procedure.
3. When the procedure returns without an error, all fields of the ModelExecution record will have been refreshed with the latest status from the server.

Related topics

`getstatus`, `executorexecute`, `executorwaitfor`

executorrepositorystatus

Purpose

Query the current repository status for the Xpress Executor 3 component

Synopsis

```
function executorrepositorystatus(exec:Executor):RepositoryStatus
```

Argument

`exec` The Executor to query

Example

```
repostatus := executorrepositorystatus( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
if repostatus.binarydataquota - repostatus.binarydatasze < 10 then
  writeln("Warning: Executor is less than 10Mb under binary data quota")
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The repository status will summarise how many executions are stored on the component and how close it is to the data storage quota.

Related topics

`getstatus`

executorsetconcurrencylevel

Purpose

Configure the concurrency level of the Xpress Executor 3 component.

Synopsis

```
procedure executorsetconcurrencylevel(exec:Executor, newlevel:integer)
```

Arguments

`exec` The Executor to configure

`level` The new concurrency level, must be an integer greater than 1

Example

```
executorsetconcurrencylevel( myexecutor, 4)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. ”Concurrency Level” is the number of executions that can be performed concurrently on a single gear; if the Xpress Executor component is scaled over multiple gears then the total number of concurrent executions will be concurrencylevel*numgears
3. If you start an execution when the maximum number of executions are concurrently executing, new executions are queued and will start when other executions complete.
4. In local mode, the concurrency level cannot be modified and calling this procedure will do nothing.
5. This can only be used with the design instance of the Xpress Executor component, or with an Executor initialized using `executorinitlocal`.

Related topics

`getstatus` `executorgetconcurrencylevel`

executorsetexecmode

Purpose

Configure the execution mode of the Xpress Executor 3 component.

Synopsis

```
procedure executorsetexecmode(exec:Executor, execmode:string)
```

Arguments

`exec`	The Executor to configure
`execmode`	The execution mode; one of `EXECUTOR_MODE_MODEL_BIM` (if modelfile is a single .bim), `EXECUTOR_MODE_MODEL_ZIP` (if modelfile is a .zip file containing a model.bim), `EXECUTOR_MODE_LP_XMPS` (if the execution input data is a model in LP or XMPS format).

Example

```
executorsetexecmode( myexecutor, EXECUTOR_MODE_MODEL_ZIP )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. This can only be used with the design instance of the Xpress Executor component, or with an Executor initialized using `executorinitlocal`.
3. If the executor has been initialized with `executorinitlocal`, the execution mode may not be `EXECUTOR_MODE_LP_XMPS`.

Related topics

`getstatus`

executorsetexectimeout

Purpose

Configure the execution timeout of the Xpress Executor 3 component.

Synopsis

```
procedure executorsetexectimeout(exec:Executor, newlevel:integer)
```

Arguments

`exec`	The Executor to configure
`timeout`	The new execution timeout in seconds, must be an integer greater than 1

Example

```
executorsetexectimeout( myexecutor, 4)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. If a model runs for longer than the configured execution timeout, Xpress Executor may interrupt it and return the execution as status STOPPED.
3. In local mode, the execution timeout cannot be set and calling this procedure will do nothing.
4. This can only be used with the design instance of the Xpress Executor component, or with an Executor initialized using `executorinitlocal`.

Related topics

`getstatus` `executorgetexectimeout`

executorsetexpiryhours

Purpose

Configure the number of hours after which a completed execution may be automatically deleted.

Synopsis

```
procedure executorsetexpiryhours(exec:Executor, hours:integer)
```

Arguments

`exec` The Executor to configure

`hours` The new number of hours after which a completed execution may be automatically deleted,

Example

```
executorsetexpiryhours( myexecutor, 4)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. In local mode, executions are never automatically expired so calling this procedure does nothing.
3. This can only be used with the design instance of the Xpress Executor component, or with an Executor initialized using `executorinitlocal`.

Related topics

`getstatus` `executorgetexpiryhours`

executorsetinputxsd

Purpose

Configure the input XSD describing the format of the input data.

Synopsis

```
procedure executorsetinputxsd(exec:Executor, srcfile:text)
```

Arguments

`exec`	The Executor to configure
`srcfile`	Filename of the local input XSD file

Example

```
executorsetinputxsd( myexecutor, "input.xsd" )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. This cannot only be used with the design instance of the Xpress Executor component.
3. The input XSD will be incorporated into the SOAP WSDL for the Xpress Executor component.
4. If you aren't calling the Xpress Executor SOAP interface from FICO Application Studio, you don't need to set the Input XSD.

Related topics

`getstatus`

executorsetinstanceconfig

Purpose

This subroutine is deprecated and will be removed in a future release. Use `executorsetinstanceconfigext` instead.
Configure when Xpress Executor 3 is allowed to re-use a Mosel instance (process) for multiple executions. Legacy version included for compatibility with Xpress 8.11; use `executorsetinstanceconfigext` instead.

Synopsis

```
procedure executorsetinstanceconfig(exec:Executor,
      newcfg:MoselInstanceConfig)
```

Arguments

`exec`	The Executor to configure
`newcfg`	A record describing in what circumstances (if any) Xpress Executor will re-use Mosel processes for multiple executions

Example

```
declarations
  cfg: MoselInstanceConfig
end-declarations
cfg.reuse:=false
executorsetinstanceconfig( myexecutor, cfg)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. In local mode, executions are always run in the same Mosel instance so this procedure has no effect.
3. This can only be used with the design instance of the Xpress Executor component, or with an Executor initialized using `executorinitlocal`.

Related topics

`getstatus` `executorgetinstanceconfig`

executorsetinstanceconfigext

Purpose

Configure when Xpress Executor 3 is allowed to re-use a Mosel instance (process) for multiple executions.

Synopsis

```
procedure executorsetinstanceconfigext(exec:Executor,
      newcfg:MoselInstanceConfigExt)
```

Arguments

`exec` The Executor to configure

`newcfg` A record describing in what circumstances (if any) Xpress Executor will re-use Mosel processes for multiple executions

Example

```
declarations
  cfg: MoselInstanceConfigExt
end-declarations
cfg.reuse:=false
executorsetinstanceconfigext( myexecutor, cfg)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. In local mode, executions are always run in the same Mosel instance so this procedure has no effect.
3. This can only be used with the design instance of the Xpress Executor component, or with an Executor initialized using `executorinitlocal`.

Related topics

`getstatus` `executorgetinstanceconfig`

executorsetinterruptconfig

Purpose

Configure how Xpress Executor 3 will interrupt an executing Mosel model, when it is necessary to do so

Synopsis

```
procedure executorsetinterruptconfig(exec:Executor,
       newcfg:ModelInterruptionConfig)
```

Arguments

`exec` The Executor to configure

`newcfg` Configuration of how Xpress Executor will interrupt executing Mosel models

Example

```
declarations
  cfg: ModelInterruptionConfig
end-declarations
cfg.minruntimeseconds:=17
executorsetinterruptconfig( myexecutor, cfg)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. In local mode, this procedure has no effect.
3. This can only be used with the design instance of the Xpress Executor component, or with an Executor initialized using `executorinitlocal`.

Related topics

`getstatus` `executorgetinterruptconfig`

executorsetmodelfile

Purpose

Configure the model file of the Xpress Executor 3 component.

Synopsis

```
procedure executorsetmodelfile(exec:Executor, srcfile:text)
```

Arguments

`exec` The Executor to configure

`srcfile` Filename of the local model file

Example

```
executorsetmodelfile( myexecutor, "mymodel.bim" )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The model file is the Mosel model executed by Xpress Executor executions. It should be a .bim file if the execution mode is "FIXED_MODEL", or a .zip containing a file called model.bim if "FIXED_MODEL_IN_ZIPFILE".
3. This can only be used with the design instance of the Xpress Executor component, or with an Executor initialized using `executorinitlocal`.

Related topics

`getstatus`

executorsetmodelparams

Purpose

Configure the model parameters of the Xpress Executor 3 component.

Synopsis

```
procedure executorsetmodelparams(exec:Executor, modelparameters:array(set
      of string) of text)
```

Arguments

`exec`	The Executor to configure
`modelparameters`	The model parameters

Example

```
declarations
  myparams: array(set of string) of text
end-declarations
myparams("VERBOSE") := "true"
myparams("MAX_COST") := "1.5"
executorsetmodelparams( myexecutor, myparams)
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. The parameters set by this procedure will be passed to the model for all subsequent executions with this Executor.
3. Where named model parameters are set both here and in `executorexecute`, the values passed to `executorexecute` shall have priority.
4. This can only be used with the design instance of the Xpress Executor component, or with an Executor initialized using `executorinitlocal`.

Related topics

`getstatus` `executorgetmodelparams`

executorsetresultxsd

Purpose

Configure the result XSD describing the format of the result data.

Synopsis

```
procedure executorsetresultxsd(exec:Executor, srcfile:text)
```

Arguments

`exec`	The Executor to configure
`srcfile`	Filename of the local result XSD file

Example

```
executorsetresultxsd( myexecutor, "result.xsd" )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. This cannot only be used with the design instance of the Xpress Executor component.
3. The result XSD will be incorporated into the SOAP WSDL for the Xpress Executor component.
4. If you aren't calling the Xpress Executor SOAP interface from FICO Application Studio, you don't need to set the Result XSD.

Related topics

`getstatus`

executorwaitfor

Purpose

Wait until the given execution has completed

Synopsis

```
procedure executorwaitfor(exec:Executor, execution:ModelExecution,
       timeoutseconds:integer)
```

Arguments

`exec`	The Executor to on which the execution resides
`execution`	The ModelExecution to wait for
`timeoutseconds`	The maximum number of seconds to wait before returning

Example

```
executorwaitfor( myexecutor, myexecution )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
```

Further information

1. After calling, check the value of `exec.status` for any errors.
2. Only the `id` field of the ModelExecution record is read by this procedure.
3. When the procedure returns without an error, all fields of the ModelExecution record will have been refreshed with the latest status from the server.
4. Check the `ModelExecution.iscompleted` field to see whether the execution completed within the given timeout period.
5. This procedure will make repeated calls to `executorpoll` will delays in between. You can configure how frequently it should call `executorpoll` by modifying the parameter `executor_poll_interval`.

Related topics

`getstatus`, `executorexecute`, `executorpoll`, `executor_poll_interval`

getlasterror

Purpose

Returns a string describing the error that occurred during the most recent operation on the Executor instance.

Synopsis

```
function getlasterror(exec:Executor):text
```

Return value

A text value containing the error message, which will be empty if the most recent operation on the Executor instance succeeded.

Further information

1. After every call to an Executor-related function or procedure, you should check the value of `getstatus` to see if your request succeeded. If it's unclear why an error is being returned, more troubleshooting output can be generated by setting the parameter `executor_verbose` to true, or inspecting the return value of `getlasterror`.
2. This function returns human-readable English description of the error that may be useful for troubleshooting purposes, but you should not assume that it is in any particular format. To distinguish between different types of error, use `getstatus` instead.

Related topics

`getstatus` `executor_verbose`

getstatus

Purpose

Indicates the status of the most recent request this model has made using the Executor instance

Synopsis

```
function getstatus(exec:Executor):integer
```

Return value

One of the following constants:

`EXECUTOR_OK` The operation completed successfully.

`EXECUTOR_NOT_FOUND` The requested execution was not found in the Xpress Executor service.

`EXECUTOR_ACCESS_DENIED` User is not authorized to access the Xpress Executor service

`EXECUTOR_CONNECTION_ERROR` Unable to connect to the Xpress Executor service.

`EXECUTOR_SERVICE_ERROR` The Xpress Executor service returned an unexpected error code.

`EXECUTOR_IO_ERROR` The executor module encountered an error reading or writing local files.

`EXECUTOR_PARSE_ERROR` The executor module did not understand the response from the Executor component.

`EXECUTOR_USAGE_ERROR` The executor module did not understand a parameter you passed to it.

Further information

After every call to an Executor-related function or procedure, you should check the value of `getstatus` to see if your request succeeded. If it's unclear why an error is being returned, more troubleshooting output can be generated by setting the parameter 'executor_verbose' to true, or inspecting the return value of `getlasterror`.

Related topics

`getlasterror`

getversion

Purpose

Query the version of the Xpress Executor component

Synopsis

```
function getversion(exec:Executor):string
```

Argument

`exec` The Executor to query

Return value

The Executor component verison, e.g. 3.0.6

Example

```
version := getversion( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
writeln('Executor component version: ',version)
```

Further information

After calling, check the value of `exec.status` for any errors.

Related topics

`getstatus`

getxpressversion

Purpose

Query the version of Xpress running inside the Xpress Executor component

Synopsis

```
function getxpressversion(exec:Executor):string
```

Argument

`exec` The Executor to query

Return value

The Xpress verison, e.g. 8.11.1

Example

```
version := getxpressversion( myexecutor )
if myexecutor.status<>EXECUTOR_OK then
  writeln("Error returned by Executor: ",myexecutor.lasterror)
  exit(1)
end-if
writeln('Executor component Xpress version: ',version)
```

Further information

After calling, check the value of `exec.status` for any errors.

Related topics

`getstatus`

CHAPTER 8

Parameters

Via the `getparam` function and the `setparam` procedure it is possible to access the following control parameters of module *executor* :

executor_verbose

Description Set to 'true' to activate additional logging of Executor-related actions and errors, to the model's error stream.

Type Boolean, read/write

Default value `false`

executor_bearer_refresh_interval

Description The interval, in seconds, after which to refresh the bearer token, when a clientid and secret has been supplied to the Executor.

Type Integer, read/write

Default value `1800`

Note This parameter has no effect when the Executor was initialized with a bearerToken value, or when used from Insight within DMP.

executor_poll_interval

Description Sets the interval, in milliseconds, between checks of the execution status in the `executorwaitfor` procedure.

Type Integer, read/write

Default value `500`

Note Values less than 100 will raise an error from `executorwaitfor`.

executor_local_workdir

Description Sets the path under which to store configuration files and model working directories for an executor configured using `executorinitlocal`.

Type String, read/write

Default value `tmp:executorlocal`

executor_components

Description This allows Executor component details to be configured using a JSON file. See the chapter on authentication, 3.2, for more details.

Type String, write only

Default value `{}`

Note For security reasons, the value of this parameter cannot be read using `getparam`.

Note The value of this parameter will be shared by all Mosel models running in the current process.

executor_max_retries

Description Set the maximum number of times a 'failed' Executor request will be retried.

Type Integer, read/write

Default value `9`

Note Each retry will be after an exponentially larger delay (0ms, 200ms, 400ms, 800ms, etc) so be aware that large values can take a long time to report errors.

Note The default setting will retry for approximately 52 seconds.

Note The types of operation that are retried can be controlled using the `executor_retry_error_codes` parameter.

executor_retry_error_codes

Description Comma-separated list of the HTTP status codes that should be retried.

Type String, read/write

Default value `999,500,503,504,403,401`

Note Controls which errors to retry when `executor_max_retries` is non-zero.

Note Supported error codes are HTTP status codes returned by the Executor HTTP component, with code 999 meaning a communication error.

CHAPTER 9

Deprecated Functionality

9.1 dmpmanagerurl

As an alternative to specifying the `bearertokenurl` attribute of the Executor variable, you can specify the `dmpmanagerurl` attribute to request the bearer token from DMP Manager, e.g.:

```
model DirectInitExample
  uses "executor"
  declarations
    myexecutor: Executor
  end-declarations

  myexecutor.componenturl := "https://vm65j75lqh-vm65j75lqh.dms.usw2.ficoanalyticcloud.com/"
  myexecutor.dmpmanagerurl := "https://console.dms.usw2.ficoanalyticcloud.com/"
  myexecutor.clientid := "vm5qh0y37c"
  myexecutor.secret := "wdS97u648BVoI#d3e4g2Z4mP780Y1DLdSDKm"
  ! myexecutor now initialized and can be used
end-model
```

Similarly, you can set the `dmpManagerUrl` field of a JSON configuration document:

```
{
  "main": {
    "componentUrl": "https://vm65j75lqh-vm65j75lqh.dms.usw2.ficoanalyticcloud.com/",
    "dmpManagerUrl": "https://console.dms.usw2.ficoanalyticcloud.com/",
    "clientId": "vm5qh0y37c",
    "secret": "wdS97u648BVoI#d3e4g2Z4mP780Y1DLdSDKm"
  }
}
```

Requesting a bearer token from DMP Manager is deprecated functionality in recent versions of DMP and this approach should not be used in new solutions.

Appendix A

Contacting FICO

FICO provides clients with support and services for all our products.

FICO Customer Support

FICO Customer Support offers technical support and services ranging from self-help tools to direct assistance with a FICO technical support engineer. Support is available to all clients who have an active maintenance contract.

The FICO Customer Self-Service Portal (support.fico.com) is a secure web portal that allows users to open, review, and update their support cases; manage their organization's portal users; find solutions to common problems in the FICO Knowledge Base; and view the availability of their cloud applications 24 hours a day, 7 days a week.

You can find support contact information and a link to the FICO Customer Self-Service Portal (online support) on the Product Support home page (www.fico.com/en/product-support).

Please include *'Xpress'* in the subject line of your support queries.

Documentation

FICO continually looks for new ways to improve and enhance the value of the products and services we provide.

If you have comments or suggestions regarding how we can improve this documentation, let us know by sending your suggestions to techpubs@fico.com. Please include your contact information (name, company, email address, and optionally, your phone number) so we may reach you if we have questions.

FICO Learning

FICO Learning is the principal provider of product training for our clients and partners. FICO Learning offers instructor-led classroom courses, web-based training, seminars, and training tools for both new user enablement and ongoing performance support.

For additional information, visit the FICO Learning home page at www.fico.com/en/product-training or email producteducation@fico.com.

Sales and maintenance

If you need information on other Xpress Optimization products, or you need to discuss maintenance contracts or other sales-related items, contact FICO by:

- Phone: +1 (408) 535-1500 or +44 207 940 8718
- Web: www.fico.com/optimization and use the available contact forms

About FICO

FICO (NYSE:FICO) is a leading analytics software company, helping businesses in 90+ countries make better decisions that drive higher levels of growth, profitability, and customer satisfaction. Learn more at www.fico.com or contact us at www.fico.com/en/contact-us.

Index