AEC2: Managing Amazon EC2 from Mosel

3.0

USER GUIDE

FICO® Xpress Optimization

FICO

AEC2 3.0

Deliverable Version: A

Last Revised: June 2025

Contents

CHAPTER 1

Introduction

Amazon Elastic Compute Cloud (EC2) is part of Amazon Web Services (AWS http://aws.amazon.com). Using either a web interface or a set of API calls the service allows to run virtual machines on the Amazon infrastructures. A virtual machine *instance* is started from an Amazon Machine Image (AMI): this is the image of the machine to run (it includes both the operating system and a set of pre-installed applications).

The AEC2 Mosel package is a subset of the AWS API for Mosel. It enables a model to:

- manage basic AWS functionality like creating keys, defining security credentials, building AMIs, ...
- start/stop and query the status of running virtual machines in the Cloud
- start and use Mosel instances on virtual machines just like they were running on the local network (using the standard *mmjobs* functionality)

1.1 Overview

The AEC2 package consists in a Mosel package (`aec2.bim`) and the program `aec2setup`. In addition to these components, system commands `ssh` and `scp` to connect to remote hosts using the SSH protocol are also required.

Using Amazon EC2 requires the availability of machine images (AMI) in order to start virtual machines. For the particular case of Mosel, the AMI must have Xpress installed and accept connections for running remote Mosel instances either via SSH or using the *xprmsrv* server: the provided setup program (`aec2setup`) can be used to create appropriate images. This program, that is to be run once only, also results in a set of configuration files that are required to initialise the package when it is used by a model.

With the help of the *mmjobs* module a Mosel model can already start and manage remote instances for creating distributed applications. The AEC2 package adds a set of routines to this framework for managing EC2 virtual machines and generating the appropriate connection strings for the `connect` function of *mmjobs* AEC2 is designed such that enabling an existing distributed model to use resources in the Cloud merely requires the user to add a few statements to the original Mosel model source code. For instance, consider the following model that compiles and then executes the model `"mymodel.mos"` on some remote host `"myserver"`:

```
model dist
uses 'mmjobs'

declarations
 minst: Mosel
 rmod: Model
end-declarations

if connect(minst, "myserver")<>0 then
```

```
 exit(1)
end-if

if compile(minst, "", "rmt:mymodel.mos", "tmp:m.bim")<>0 then
 exit(2)
end-if

load(minst, rmod, "tmp:m.bim")
run(rmod)
wait
dropnextevent
unload(rmod)
disconnect(minst)
end-model
```

The following example performs the same operations but using a virtual machine in EC2 instead of a local server:

```
model distec2
uses 'mmjobs','aec2'

declarations
 ecf: EC2Conf
 ainst: EC2instance
 minst: Mosel
 rmod: Model
end-declarations

if not loadAEC2Config(ecf,"aec2.acf") then  ! load AEC2 configuration
 exit(-1)
end-if

ainst:=runInstance(ecf)                     ! start a virtual machine in the Cloud
if not waitInstanceReady(ecf,ainst, 300, true) then
 exit(-2)
end-if

if connect(minst,getConnString(ecf,ainst))<>0 then
 exit(1)
end-if

if compile(minst, "", "rmt:mymodel.mos", "tmp:m.bim")<>0 then
 exit(2)
end-if

load(minst, rmod, "tmp:m.bim")
run(rmod)
wait
dropnextevent
unload(rmod)
disconnect(minst)

terminateInstance(ecf,ainst)                ! terminate the virtual machine
end-model
```

Note the use of function `getConnString` as input parameter for the *mmjobs* function `connect`: it generates a valid connection string for the specified Amazon instance (either for an SSH or *xprmsrv* link depending on the default protocol).

1.2 Setup and creation of an Amazon Machine Image (AMI) for Mosel

The AEC2 settings are stored in a set of configuration files. The `'aec2setup'` program will configure your AWS account as needed and create these configuration files. One of the tasks performed by this program is to create a dedicate AMI running the Linux operating system. To this end, an Xpress Linux package (full setup or patch for x86_64 or aarch64 architecture) together with its associated license file are required to run the program (the community license will be selected if no license is provided).

To run the setup program, copy into the same directory the Xpress Linux archive (file name in the form `"xp?.?.?_linux*_setup.tar"`) and the `"xpauth.xpr"` license (if available), then, from a command prompt type:

```
aec2setup
```

During the process, the program will ask various questions, in most cases a default answer is provided (noted in square brackets): simply hit enter if you do not know what to do. As part of the procedure a virtual machine is started in the Cloud and Xpress is installed on it in order to create the AMI to be used later on. Note that:

- starting a virtual machine and creating the AMI may be quite lengthy operations (up to several minutes): do not interrupt the program during these operations
- in case of failure, you can restart the program: it will recover information already setup.

Once the setup has completed, the following files have been created:

- `"aec2.aid"` the secret key-pair to access your AWS account. This file must be kept secret as it gives full access to your AWS account
- `"aec2.ask"` the private key to log into virtual machines. The information stored in this file cannot be regenerated. If it is lost or corrupted, a new security credential will have to be created (*i.e.* delete the file and re-run `aec2setup`)
- `"aec2.acf"` the main configuration file (that references the 2 others)

These 3 files are required by the AEC2 package when running a model using it.

CHAPTER 2

Main functions of the AEC2 package

AEC2_PENDING = 0

Instance state: starting

AEC2_RUNNING = 16

Instance state: running

AEC2_SHUTTING_DOWN = 32

Instance state: shutting down

AEC2_STOPPED = 80

Instance state: stopped

AEC2_STOPPING = 64

Instance state: stopping

AEC2_TERMINATED = 48

Instance state: terminated

AwsID : `record`

Credentials for AWS access

Note It is populated from an *aid* file that can be generated with `savePrivateKey` and read with `loadPrivateKey`.

EC2Conf : `record`

Conplete configuration for AEC2

endpoint : `text`

AWS EndPoint where to send queries

aidfile : `string`

File containing AWS credentials

aid : `AwsID`

AWS credentials

askfile : `string`

File containing SSH key information

sshid : `SshID`

SSH key information

image : `string`

Id of the image to start

type : string

Type of instance to run (*e.g.* `"t2.mivro"`)

secgrp : string

Id of the security group for the instance

connmode : string

Default connection model (`"ssh"` or `"xprmsrv"`)

xsrvctx : string

xprmsrv context for connection

xsrvpass : text

Password for *xprmsrv* connection

xsrvport : integer

Port to use for *xprmsrv* connection

Note It is populated from an *acf* file that can be loaded with `loadAEC2Config`. Run the program `aec2setup` to create this file.

EC2image : record

An EC2 image

id : string

Id of the image

name : text

Name of the image

description : text

Description

location : text

Location

snapshot : text

Id of the disk snapshot used for this image

available : boolean

Availability status

ispublic : boolean

Report whether the image is public

architecture : string

Architecture (*e.g.* `"x86_64"`)

owner : string

Owner of this image

ownerid : string

Id of the owner

EC2instance : record

An EC2 instance

image : string

Id of the image used to start this instance

state : integer

Current state (see AEC2_* constants)

dns : text

Dns name of the instance

key : text

Id of the SSH key to use for SSH connection

type : string

Type of the instance(*e.g.* "t2.mivro")

launchtime : datetime

Launch time

ipaddr : text

IP address of the instance

EC2secrule : record

An EC2 security rule

protocol : text

Protocol (*e.g.* "tcp")

ports : range

Ports range (*e.g.* 2513..2515)

sources : list of text

Address list (*e.g.* ["0.0.0.0/0"])

SshID : record

SSH key information to connect to the remote instance

Note It is populated from an *ask* file that can be generated with createSSHKeyFile and read with selectSSHKeyFile.

EC2statetext : dynamic array(set of integer) of string

Text representation of instance states

aec2_msg : text

Message associated to the last http query fo AWS

aec2_status : integer

http status of the last AWS query

aec2_verbose : boolean

Enable/disable logging of the package

addSecurityIngress

Purpose

Add a rule to a security group

Synopsis

```
procedure addSecurityIngress(ecf:EC2Conf, name:string, protocol:string,
      ports:range, cidrs:list of string)
```

Arguments

`ecf`	An AEC2 configuration
`name`	Name of a security group
`protocol`	Protocol for the new rule
`ports`	Range of ports for the new rule
`cidrs`	List of CIDRs for the new rule

createImage

Purpose

Create a new image from a running instance

Synopsis

```
function createImage(ecf:EC2Conf, inst:EC2instance, name:string,
      desc:string):text
```

Arguments

`ecf`	An AEC2 configuration to populate
`inst`	An EC2 instance
`name`	Name of the image to create
`desc`	Description of the image to create

Return value

Id of the created image or an empty string in case of failure

createKeyPair

Purpose

Generate a new SSH secret key-pair

Synopsis

```
function createKeyPair(ecf:EC2Conf, name:string, fprint:text):text
```

Arguments

`ecf`	AEC2 configuration
`name`	Name of the newly created key-pair
`fprint`	Returned fingerprint of the key-pair

Return value

The key-pair data in a textual form

Further information

The generated key can be saved to a key-file using `storeSSHKeyFile`. The key-file can also directly be created using `createSSHKeyFile`. A key-pair created with this function can be deleted using `deleteKeyPair`.

createSSHKeyFile

Purpose

Generate then store a SSH key information in a key-file

Synopsis

```
function createSSHKeyFile(ecf:EC2Conf, fname:string,
      keyname:string):boolean
```

Arguments

`ecf` AEC2 configuration

`fname` Destination file name

Return value

`true` if operation succeeded.

Further information

The key data information is generated with `createKeyPair` and the key-file is created using `storeSSHKeyFile` with `"ec2-user"` as the user name.

createSecurityGroup

Purpose

Create a new security group

Synopsis

```
procedure createSecurityGroup(ecf:EC2Conf, name:string, desc:string)
```

Arguments

`ecf`	AEC2 configuration to initialise
`name`	Name of the security group
`desc`	Description for this group

Further information

Use `addSecurityIngress` and `delSecurityIngress` to update the rules of a security group

delSecurityIngress

Purpose

Delete a rule to a security group

Synopsis

```
procedure delSecurityIngress(ecf:EC2Conf, name:string, protocol:string,
      ports:range, cidrs:list of string)
```

Arguments

`ecf`	An AEC2 configuration
`name`	Name of a security group
`protocol`	Protocol of the rule
`ports`	Range of ports of the rule
`cidrs`	List of CIDRs of the rule

deleteKeyPair

Purpose

Delete a SSH secret key-pair

Synopsis

```
procedure deleteKeyPair(ecf:EC2Conf, name:string)
```

Arguments

`ecf`	AEC2 configuration
`name`	name of the key to delete

Further information

This procedure may be used to delete a key previously created with `createKeyPair`.

deleteSecurityGroup

Purpose

Delete a security group

Synopsis

```
procedure deleteSecurityGroup(ecf:EC2Conf, name:string)
```

Arguments

`ecf`	AEC2 configuration to initialise
`name`	Name of the security group to delete

deregisterImage

Purpose

Deregister (or delete) an image

Synopsis

```
function deregisterImage(ecf:EC2Conf, imgid:string):boolean
```

Arguments

`ecf` An AEC2 configuration to populate

`imgid` Id of the image to delete

Return value

`true` if operation succeeded.

describeKeyPair

Purpose

Get the fingerprint of an existing SSH key-pair

Synopsis

```
function describeKeyPair(ecf:EC2Conf, name:string):text
```

Arguments

`ecf`	AEC2 configuration
`name`	Name of the key-pair

Return value

The key-pair fingerprint or an empty string in case of error

describeSecurityGroup

Purpose

Get the description of a security group

Synopsis

```
function describeSecurityGroup(ecf:EC2Conf, name:string):list of EC2secrule
```

Arguments

`ecf`	An AEC2 configuration
`name`	Name of a security group

Return value

The list of rules associated to the security group

getAllInstances

Purpose

Retrieve a list of Instances from AWS.

Synopsis

```
function getAllInstances(ecf:EC2Conf):list of EC2instance
```

Argument

`ecf` An AEC2 configuration

Return value

List of running (or recently terminated) instances.

Further information

The returned list may contain instances that have already terminated. It is important to check the actual state of an instance before using it (field `state` of the `EC2instance` type).

Related topics

`getImages`

getAllSecurityGroups

Purpose

Get the names of all security groups

Synopsis

```
function getAllSecurityGroups(ecf:EC2Conf):list of text
```

Argument

`ecf` An AEC2 configuration

Return value

The list of the names of the current security groups

Further information

The properties of a security group can be obtained with `describeSecurityGroup`

getConnString

Purpose

Generate a connection string.

Synopsis

```
function getConnString(ecf:EC2Conf, inst:EC2instance):string
```

Arguments

`ecf`	An AEC2 configuration
`inst`	Target instance

Return value

A connection string suitable for the `connect` routine of *mmjobs*.

Further information

This function calls either `getConnStringSSH` or `getConnStringXSRV` depending on the value of the variable `ecf.connmode`.

Related topics

`getConnStringXSRV`, `getConnStringSSH`

getConnStringSSH

Purpose

Generate a connection string for an SSH link.

Synopsis

```
function getConnStringSSH(ecf:EC2Conf, inst:EC2instance):string
```

Arguments

`ecf` An AEC2 configuration

`inst` Target instance

Return value

A connection string suitable for the `connect` routine of *mmjobs*.

Further information

The resulting connection string is of the form `rcmd:ssh -o StrictHostKeyChecking=no -i keyfile user@ipaddr mosel -r"`.

Related topics

`getConnString`, `getConnStringXSRV`

getConnStringXSRV

Purpose

Generate a connection string for an *xprmsrv* server.

Synopsis

```
function getConnStringXSRV(ecf:EC2Conf, inst:EC2instance):string
```

Arguments

`ecf`	An AEC2 configuration
`inst`	Target instance

Return value

A connection string suitable for the `connect` routine of *mmjobs*.

Further information

The resulting connection string is of the form `"xssh:ipaddr(port)/ctx/pass"`.

Related topics

`getConnString`, `getConnStringSSH`

getEndPoints

Purpose

Retrieve all endpoint addresses

Synopsis

```
function getEndPoints(aid:AwsID, region:string):list of text
```

Arguments

`aid`	AWS credentials
`region`	Limit search to the specified region (can be an empty string)

Return value

List of available endpoints

getId

Purpose

Get the Id of an instance

Synopsis

```
function getId(inst:EC2instance):text
```

Argument

`inst` An EC2 instance

Return value

The Id of the instance

getImages

Purpose

Retrieve a list of AMI from AWS.

Synopsis

```
function getImages(ecf:EC2Conf, id:string, filt:array(set of string) of
      text, owners:set of string):list of EC2image
function getImages(ecf:EC2Conf, id:string, name:string, desc:string):list
      of EC2image
function getImages(ecf:EC2Conf, id:string):list of EC2image
```

Arguments

`ecf`	An AEC2 configuration
`id`	Image ID (or an empty string)
`name`	Image name (or an empty string)
`desc`	Image description (or an empty string)
`filt`	Array of filter properties
`owners`	Set of owners

Return value

List of images with the specified properties.

Example

The following example retrieves all images created by `aec2setup` and the specification of the default image used to start new instances:

```
xpimgs:=getImages("","","Xpress for AEC2*")
defimg:=getImages(aec2_image,"","")
```

Further information

1. Each of the parameters is used as a filter: the resulting list will keep only images with the corresponding property matching the filter. Filters support wildcard charcaters `"*"` (any sequence of character possibly empty) and `"?"` (any character). An empty string is equivalent to `"*"` (any string).
2. Images created by the program `aec2setup` have the string `"Xpress for AEC2"` included in their description: this can be used to find existing images compatible with the package.

Related topics

`getAllInstances`

getInstances

Purpose

Get and/or start from atl to atm instances

Synopsis

```
function getInstances(ecf:EC2Conf, atl:integer, atm:integer,
      timeout:integer, verbose:boolean):list of EC2instance
```

Arguments

`ecf`	An AEC2 configuration
`atl`	Minimum number of instance to run
`atm`	Maximum number of instance to run (if positive)
`timeout`	Maximum delay to start a new instance
`verbose`	Verbosity option of `waitInstanceReady`

Return value

List of running instances

Further information

This routine gets the list of running instances and starts additional ones if the current number is inferior to `atl`

getKeyFile

Purpose

Get the current key-file (for ssh connections)

Synopsis

```
function getKeyFile(ecf:EC2Conf):text
```

Argument

`ecf` An AEC2 configuration

Return value

The fey-file name of used by the provided configuration

getLaunchPermission

Purpose

Retrieve launch permission of an image

Synopsis

```
function getLaunchPermission(ecf:EC2Conf, imgid:string):set of string
```

Arguments

`ecf` An AEC2 configuration

`imgid` Id of the image

Return value

List of Ids of users having launch permission for this image

importKeyPair

Purpose

Upload to AWS the public part of a local key-pair

Synopsis

```
function importKeyPair(ecf:EC2Conf, name:string, data:text):text
```

Arguments

`ecf`	An AEC2 configuration
`name`	Name of the key
`data`	The public part of the key in a textual form

Return value

The key fingerprint or an empty string in case of error

isValidPrivateKey

Purpose

Test whether the provided AWS credentials are valid

Synopsis

```
function isValidPrivateKey(aid:AwsID):boolean
```

Argument

`aid` AWS credentials

Return value

`true` if the credentails are valid

Further information

This routine checks the validity of AWS crdentials by sending a simple query to EC2.

Related topics

`loadPrivateKey`

loadAEC2Config

Purpose

Load a configuration file and initialise the package.

Synopsis

```
function loadAEC2Config(ecf:EC2Conf, fname:string):boolean
```

Arguments

`ecf` An AEC2 configuration to populate

`fname` Configuration file name

Return value

`true` if operation succeeded.

Further information

This function must be called before any other routine of the package. The configuration files are created by running the "`aec2setup`" program.

loadPrivateKey

Purpose

Initialise an AWS credentials from an aid-file

Synopsis

```
procedure loadPrivateKey(aid:AwsID, fdir:text, fname:string)
procedure loadPrivateKey(aid:AwsID, fname:string)
procedure loadPrivateKey(ecf:EC2Conf, fdir:text, fname:string)
procedure loadPrivateKey(ecf:EC2Conf, fname:string)
```

Arguments

`ecf`	AEC2 configuration to initialise
`aid`	AWS credentials to initialise
`fdir`	Directory for the file
`fname`	Name of the aid-file

Further information

The credentials can also be initialised directly using `setPrivateKey`

Related topics

`isValidPrivateKey`

newParameter

Purpose

Generate a parameter for a query

Synopsis

```
function newParameter(name:text, value:text):text
function newParameter(name:string, value:integer):text
```

Arguments

`name` Parameter name

`value` Parameter value

Return value

An URL encoded string of the form `name=value`

Further information

This function will be used to generate the list of parameters required by `sendQuery`

registerImage

Purpose

Register a new image from an S3 bucket

Synopsis

```
function registerImage(ecf:EC2Conf, bucket:string, name:string,
      desc:string):text
```

Arguments

`ecf`	An AEC2 configuration to populate
`bucket`	S3 bucket containing the image
`name`	Name of the image to create
`desc`	Description of the image to create

Return value

Id of the created image or an empty string in case of failure

runInstance

Purpose

Start a new instance in EC2.

Synopsis

```
function runInstance(ecf:EC2Conf):EC2instance
```

Argument

`ecf` An AEC2 configuration

Return value

Instance record referencing the created instance.

Further information

This function starts a new instance using the current configuration (in particular using 'ecf.image' as the image). Note that an instance needs some time before it is ready to execute commands — this routine does not wait for the instance to be ready: use `waitInstanceReady` for this purpose.

Related topics

`waitInstanceReady`, `updateInstance`

saveAEC2Config

Purpose

Save the provided configuration into a file

Synopsis

```
procedure saveAEC2Config(ecf:EC2Conf, fname:string)
```

Arguments

`ecf` An AEC2 configuration to save

`fname` Destination file name

savePrivateKey

Purpose

Save the provided AWS credentials into an aid-file

Synopsis

```
procedure savePrivateKey(aid:AwsID, fname:string)
procedure savePrivateKey(ecf:EC2Conf, fname:string)
```

Arguments

`ecf`	AEC2 configuration
`aid`	AWS credentials
`fname`	Name of the aid-file to create

Further information

The AWS credentials can be initialised either directly with `setPrivateKey` or from a file with `loadPrivateKey`

selectSSHKeyFile

Purpose

Select the key to use from a key-file

Synopsis

```
function selectSSHKeyFile(ecf:EC2Conf, fdir:text, fname:string):boolean
function selectSSHKeyFile(ecf:EC2Conf, fname:string):boolean
```

Arguments

`ecf`	AEC2 configuration
`fname`	Name of the key-file
`fdir`	Directory name for the file (can be an empty string)

Return value

`true` if operation succeeded.

Further information

After the key-pair has been loaded, the infomation is saved into the provided AEC2 configuration

sendQuery

Purpose

Generate then send a query to the specified endpoint

Synopsis

```
function sendQuery(aid:AwsID, edp:text, lp:list of text,
      fname:string):boolean
function sendQuery(ecf:EC2Conf, lp:list of text, fname:string):boolean
```

Arguments

`aid`	AWS credentials
`edp`	AWS EndPoint where to send the query
`ecf`	AEC2 configuration
`lp`	List of parameters
`fname`	Name of the file that receives the result of the query

Return value

`true` if operation succeeded.

Example

The following example displays all available endpoints:

```
if sendQuery(aid,"ec2.amazonaws.com",
             [newParameter("Action","DescribeRegions")],"tmp:regs.xml") then
 load(doc,"tmp:regs.xml")
 getnodes(doc,"/DescribeRegionsResponse/regionInfo/item/regionEndpoint",elts)
 forall(l in elts) writeln(getvalue(doc,l))
end-if
```

Further information

The query is constructed from the provided parameters (see `newParameter`) to the specified endpoint using the provided credentials. Alternatively the credentials and endpoint might be taken from an aec2 configuration (see `loadAEC2Config`).

setLaunchPermission

Purpose

Add/remove launch permission of an image to a user

Synopsis

```
function setLaunchPermission(ecf:EC2Conf, what:boolean, imgid:string,
      userid:string):boolean
```

Arguments

`ecf`	An AEC2 configuration
`what`	`true`: add permission; `false` remove permission
`imgid`	Id of the image
`userid`	Id of the user to add or remove permission

Return value

`true` if operation succeeded.

setPrivateKey

Purpose

Initialise an AWS credentials

Synopsis

```
procedure setPrivateKey(aid:AwsID, id:text, pk:text, st:text)
```

Arguments

`aid`	AWS credentials
`pk`	AWS Private key
`st`	AWS Session token

Further information

The credentials can be saved to a file using `savePrivateKey`

storeSSHKeyFile

Purpose

Store the SSH key information in a ”key file”

Synopsis

```
procedure storeSSHKeyFile(fname:string, user:string, keyname:string,
      key:text, kp:text)
```

Arguments

`fname`	Destination file name
`user`	User name
`keyname`	Key name
`key`	Key data

Further information

The key data information can be generated with `createKeyPair`. The key-file can also directly be created using `createSSHKeyFile`.

terminateInstance

Purpose

Terminate the specified instance.

Synopsis

```
procedure terminateInstance(ecf:EC2Conf, inst:EC2instance)
```

Arguments

`ecf`	An AEC2 configuration
`inst`	Instance to terminate

Related topics

`runInstance`

updateInstance

Purpose

Update instance information.

Synopsis

```
procedure updateInstance(ecf:EC2Conf, inst:EC2instance)
```

Arguments

`ecf`	An AEC2 configuration
`inst`	Instance object to update

Further information

An `EC2instance` object contains information on an instance running in EC2. This information is not necessarily up to date: this procedure queries AWS and updates properties stored in the corresponding record.

waitInstanceReady

Purpose

Wait for an instance to be ready.

Synopsis

```
function waitInstanceReady(ecf:EC2Conf, inst:EC2instance, delay:integer,
      verbose:boolean):boolean
```

Arguments

`ecf`	An AEC2 configuration
`inst`	Instance to wait for
`delay`	Maximum amount of time (seconds) to wait
`verbose`	Display progress bar if `true`

Return value

`true` if the instance was ready before the expiration of the specified delay.

Further information

This function waits at most `delay` seconds for the instance `inst` to be ready. If `verbose` is `true`, a dot will be displayed at regular intervals.

Related topics

`runInstance`, `updateInstance`

Appendix

APPENDIX A

Contacting FICO

FICO provides clients with support and services for all our products. Refer to the following sections for more information.

Product support

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

The FICO Customer Self Service Portal is a secure web portal that is available 24 hours a day, 7 days a week from the Product Support home page. The portal allows you to open, review, update, and close cases, as well as find solutions to common problems in the FICO Knowledge Base.

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

Product education

FICO Product Education is the principal provider of product training for our clients and partners. Product Education 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 Product Education homepage at www.fico.com/en/product-training or email producteducation@fico.com.

Product 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.

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

Related services

Strategy Consulting: Included in your contract with FICO may be a specified amount of consulting time to assist you in using FICO Xpress Insight to meet your business needs. Additional consulting time can be arranged by contract.

Conferences and Seminars: FICO offers conferences and seminars on our products and services. For announcements concerning these events, go to www.fico.com or contact your FICO account representative.

FICO Community

The FICO Community is a great resource to find the experts and information you need to collaborate, support your business, and solve common business challenges. You can get informal technical support, build relationships with local and remote professionals, and improve your business practices. For additional information, visit the FICO Community (community.fico.com/welcome).

About FICO

FICO (NYSE:FICO) powers decisions that help people and businesses around the world prosper. Founded in 1956 and based in Silicon Valley, the company is a pioneer in the use of predictive analytics and data science to improve operational decisions. FICO holds more than 165 US and foreign patents on technologies that increase profitability, customer satisfaction, and growth for businesses in financial services, telecommunications, health care, retail, and many other industries. Using FICO solutions, businesses in more than 100 countries do everything from protecting 2.6 billion payment cards from fraud, to helping people get credit, to ensuring that millions of airplanes and rental cars are in the right place at the right time. Learn more at www.fico.com.

Index