S3 Library for Mosel

9.7

REFERENCE MANUAL

FICO® Xpress Optimization

FICO

FICO® Xpress Optimization 9.7

Deliverable Version: A

Last Revised: 30 May, 2023

Contents

CHAPTER 1

Introduction

The package and module *s3* allow objects to be uploaded to and downloaded from an Amazon AWS S3 service, or a compatible third-party service that supports the AWS Signature Version 4 authentication method. This functionality is available from local models as well as models running in supported components on the FICO Decision Management Platform (DMP).

When not used from within a FICO® Xpress Insight or FICO® Xpress Executor component in DMP, you will need to provide your own credentials (in the form of an access key id, secret key and optional session token) to allow access to an Amazon S3 service.

This document assumes that the reader is familiar with the basic concepts of the Amazon S3 service. If not, please consult the documentation available online, for example:

https://aws.amazon.com/s3

Getting Started with Amazon Simple Storage Service

CHAPTER 2

Basic Principles

The *s3* library should be loaded into your model or package as follows:

```
uses "s3"
```

When you want to use S3, you should declare a variable of the 's3bucket' type to represent the Amazon S3 bucket you want to access.

```
declarations
  mybucket: s3bucket
end-declarations
```

Once the 's3bucket' variable has been initialized with your bucket's location and credentials (see 3), it can be used with S3 functions such as `s3getobject`, `s3putobject` and `s3listobjects`. If you need to access more than one bucket, you can declare multiple s3bucket variables.

One property of the 's3bucket' is the 'keyprefix', which will be automatically prepended to all keys you pass to functions. This allows you to simulate having a 'working directory' within the S3 bucket, e.g.:

```
s3putobject(mybucket,'objectkey','mydata.dat')  ! Upload to key 'objectkey'
mybucket.keyprefix := 'myprefix/'
s3putobject(mybucket,'objectkey','mydata.dat')  ! Upload to key 'myprefix/objectkey'
```

The examples in this document use a '/' in keys and the keprefix to simulate a structure of folders; this is the convention used in S3 buckets provisioned by DMP, but is not required by the *s3* module.

After calling each function on the *s3* library, you should check the value of `s3status` for any errors - if this is any value other than `S3_OK` then an error has occurred. In an error case, `s3status` will return an error code and `s3getlasterror` will return a description of the error.

CHAPTER 3

Authenticating

3.1 Using s3bucket Properties

The simplest way to initialize the s3bucket with your S3 bucket's URL and access credentials is by directly setting properties of the s3bucket object within the model. For example:

```
model DirectInitExample
  uses "s3"
  declarations
    mybucket: s3bucket
  end-declarations

  mybucket.url := "https://s3-us-west-2.amazonaws.com/nameofmybucket/"
  mybucket.region := "us-west-2"
  mybucket.keyprefix := "myprefix/"                                          ! Optional
  mybucket.accesskeyid := "JKHGDMNAYGWEnbbsJGDI"
  mybucket.secretkey := "jhdfusJasfui;SVFYSIAVS++siufgsuUISNISOWJ"
  mybucket.sessiontoken := "kHUFGBSUjbfusbuioUHDFSIngudblincxubhxop0szofbv"  ! Optional
  ! mybucket now initialized and can be used
end-model
```

The Bucket URL, Region, Access Key ID and Secret Key must always be specified. If the credentials you are given include a Session Token (sometimes referred to as a Security Token), then you must also specify this. Giving a Key Prefix is optional.

Note that the S3 credentials will not be verified until you make a request to the S3 service. The values you give for the url, region and keyprefix can be read back out of the s3bucket, but for security reasons the accesskeyid, secretkey and sessiontoken properties may not be read.

If you are using server-side encryption with AWS-managed keys, you can configure this by setting additional fields on the s3bucket object, as follows:

```
  mybucket.sse := "aws:kms"
  mybucket.ssekmskeyid := "keyid"     ! Replace 'keyid' with ID of yourkey in the AWS key management servi
  mybucket.ssecontext := "x=1"        ! Encryption context; optional
```

If you need to periodically update your credentials, you can set a `refreshfunc` to a function pointer that will update the properties on the `s3bucket` once the `timestamp` has exceeded the credentials `ttl` (time-to-live), e.g.:

```
model DirectInitExample
  uses "s3"
  declarations
    mybucket: s3bucket
  end-declarations

  function refreshbucket(bucket:s3bucket):boolean
    mybucket.accesskeyid := "KHASFGusbf9634kJFGS8"
    mybucket.secretkey := "sndfd++sfa8KAD&*RWLHSjhsdlifgy8gdIA*RHDJ"
```

```
    mybucket.timestamp := timestamp    ! seconds since 1/1/1970
    mybucket.ttl := 3600               ! 1 hour, in seconds
    returned := true  ! Return true on success, false on error
  end-function

  mybucket.url := "https://s3-us-west-2.amazonaws.com/nameofmybucket/"
  mybucket.region := "us-west-2"
  mybucket.accesskeyid := "JKHGDMNAYGWEnbbsJGDI"
  mybucket.secretkey := "jhdfusJasfui;SVFYSIAVS++siufgsuUISNISOWJ"
  mybucket.timestamp := timestamp           ! seconds since 1/1/1970
  mybucket.ttl := 3600                      ! 1 hour, in seconds
  mybucket.refreshfunc := ->refreshbucket
  ! mybucket now initialized and can be used
end-model
```

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 which you assign to the `s3_buckets` parameter. The document should be in the following format:

```
{
  "<bucket-id>": {
    "url": "<URL of bucket>",
    "region": "<AWS Region>",
    "keyPrefix": "<Key Prefix, optional>",
    "accessKeyId": "<AWS Access Key ID>",
    "secretKey": "<AWS Secret Key>",
    "sessionToken": "<AWS Session Token, optional>"
  }
}
```

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

```
{
  "firstbucket": {
    "url": "https://s3-us-west-2.amazonaws.com/nameofmybucket/",
    "region": "us-west-2",
    "keyPrefix": "myprefix/",
    "accessKeyId": "JKHGDMNAYGWEnbbsJGDI",
    "secretKey": "jhdfusJasfui;SVFYSIAVS++siufgsuUISNISOWJ",
    "sessionToken": "kHUFGBSUjbfusbuioUHDFSIngudblincxubhxop0szofbv"
  },
  "secondbucket": {
    "url": "https://s3-us-east-2.amazonaws.com/nameofmyotherbucket/",
    "region": "us-east-2",
    "accessKeyId": "IHFSUGFOSFUHFJSYIFSG",
    "secretKey": "nusduoUHf;sufbuOFUGSFHRHAFFAbvubddsa=jfb",
    "sessionToken": "UHFSUOFIhfushfglhoFGSiguosnoahusfppgjoUFSUFINM"
  }
}
```

Then you can initialize an `s3bucket` in the model with the credentials from the JSON by calling the `s3init` procedure. For example, if you save the above sample JSON in a file called ”buckets.json”:

```
model InitExample
  uses "s3","mmsystem"
  declarations
    public bucketcfg: text
    mybucket: s3bucket
  end-declarations
```

```
  ! Load buckets.json into a variable so it can be passed to a parameter
  fcopy("buckets.json","text:bucketcfg")
  setparam("s3_buckets",string(bucketcfg))

  ! Initialize mybucket using the 'firstbucket' set of credentials
  s3init(mybucket, "firstbucket")
  if s3status(mybucket)<>S3_OK then
    writeln("Bucket initialization error: ", s3getlasterror(mybucket))
    exit(1)
  end-if
  ! mybucket now initialized and can be used
end-model
```

As before, the S3 credentials will not be verified until you make a request to the S3 service.

The `s3_buckets` 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.

After calling `s3init`, the only property on the s3bucket that may be modified is `keyprefix`, which must always start with the value it was given by `s3init`.

If you are using server-side encryption with AWS-managed keys, you can configure this by setting 3 additional properties on the JSON object: `sse` should be the string `"aws:kms"`, `sseKmsKeyId` the identifier of your key stored in the AWS key-management service, and `sseContext` is the (optional) encryption context string.

3.3 Using the DMP Solution Bucket

When using an Xpress Insight, Xpress Workbench, or Xpress Executor DMP component, the Mosel instance will automatically be configured to access an S3 bucket that is shared by all components in the solution. To use this, call `s3init` with the constant `S3_DMP_SOLUTIONDATA`:

```
model DmpInitExample
  uses "s3"
  declarations
    mybucket: s3bucket
  end-declarations

  ! Initialize mybucket using the 'solutionData' set of credentials
  s3init(mybucket, S3_DMP_SOLUTIONDATA)
  if s3status(mybucket)<>S3_OK then
    writeln("Bucket initialization error: ", s3getlasterror(mybucket))
    exit(1)
  end-if
  ! mybucket now initialized and can be used
end-model
```

By default you will access the solutionData folder matching your component's current DMP lifecycle stage (design, staging or production). Alternatively, you can initialize your s3bucket with the folder of a different lifecycle stage by using the `s3solutiondata` function and passing one of the constants `S3_DMP_DESIGN`, `S3_DMP_STAGING` or `S3_DMP_PRODUCTION`, as follows:

```
model DmpInitExample
  uses "s3"
  declarations
    mybucket: s3bucket
  end-declarations

  ! Initialize mybucket using the 'solutionData' credentials for the 'staging' lifecycle
  s3init(mybucket, s3solutiondata(S3_DMP_STAGING))
  if s3status(mybucket)<>S3_OK then
    writeln("Bucket initialization error: ", s3getlasterror(mybucket))
```

```
    exit(1)
  end-if
  ! mybucket now initialized and can be used
end-model
```

Please note that in Xpress Workbench, only access to the ”solution data” folder for the current lifecycle stage is supported, and the S3 credentials will only be usable within the first 45 minutes of the model’s execution.

3.4 Using the DMP Tenant Bucket

When using an Xpress Insight or Xpress Executor DMP component, the Mosel instance will automatically be configured to access an S3 bucket that is shared by all components in your tenant. To use this, call `s3init` with the constant `S3_DMP_TENANTSHARED`:

```
model DmpInitExample
  uses "s3"
  declarations
    mybucket: s3bucket
  end-declarations

  ! Initialize mybucket using the 'tenantShared' set of credentials
  s3init(mybucket, S3_DMP_TENANTSHARED)
  if s3status(mybucket)<>S3_OK then
    writeln("Bucket initialization error: ", s3getlasterror(mybucket))
    exit(1)
  end-if
  ! mybucket now initialized and can be used
end-model
```

Unlike the solution bucket, the tenant bucket is shared by all component lifecycle stages - there are not separate folders for design, staging, and production.

Please note that the shared tenant bucket cannot be accessed from Xpress Workbench.

3.5 Using the DMP Solution Revision Bucket

When using an Xpress Insight or Xpress Executor DMP component, you can read from the S3 folder for a previously committed solution revision. (A solution revision is created by committing the solution, and a location in S3 will be created to store assets of this revision. This mechanism is frequently used by non-Xpress components to share DMP function implementations and other resources. There is no way to write to a solution revision bucket from an Xpress component.) To use this, call `s3init` with the return value from calling the function `s3solutionrevision` with the ID of the solution revision you want to access, e.g.:

```
model DmpInitExample
  uses "s3"
  parameters
    REVISION_ID="7djfgs9287"
  end-parameters
  declarations
    mybucket: s3bucket
  end-declarations

  ! Initialize mybucket using the 'solutionRevision' set of credentials for revision
  s3init(mybucket, s3solutionrevision(REVISION_ID))
  if s3status(mybucket)<>S3_OK then
    writeln("Bucket initialization error: ", s3getlasterror(mybucket))
    exit(1)
  end-if
```

```
  ! mybucket now initialized and can be used
end-model
```

Please note that the solution revision buckets cannot be accessed from Xpress Workbench.

CHAPTER 4

Usage examples

4.1 Downloading from an S3 object

This example demonstrates downloading the content of an S3 object with the key `MyFile.txt` into a local file `MyDownloadedFile.txt`.

You will need to fill in the model parameters with your own Amazon S3 access credentials.

```
model S3DownloadExample
uses "s3"

parameters
  S3_BUCKET_URL = ""
  S3_REGION = ""
  S3_ACCESS_KEY_ID = ""
  S3_SECRET_KEY = ""
end-parameters

declarations
  LOCAL_FILE="MyDownloadedFile.txt"
  OBJECT_KEY="MyFile.txt"
  mybucket: s3bucket
end-declarations

! Configure 'mybucket' with our S3 access credentials
mybucket.url := S3_BUCKET_URL
mybucket.region := S3_REGION
mybucket.accesskeyid := S3_ACCESS_KEY_ID
mybucket.secretkey := S3_SECRET_KEY

! Download remote object to local file
s3getobject( mybucket, OBJECT_KEY, LOCAL_FILE )

! Check for errors
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3 service: ", s3getlasterror(mybucket))
  exit(1)
end-if

end-model
```

4.2 Uploading to an S3 object

This example demonstrates uploading the content of the file ”MyLocalFile.txt” to the S3 bucket with the object key ”MyFile.txt”.

You will need to fill in the model parameters with your own Amazon S3 access credentials.

```
model S3UploadExample
uses "s3"
```

```
parameters
  S3_BUCKET_URL = ""
  S3_REGION = ""
  S3_ACCESS_KEY_ID = ""
  S3_SECRET_KEY = ""
end-parameters

declarations
  LOCAL_FILE="MyLocalFile.txt"
  OBJECT_KEY="MyFile.txt"
  mybucket: s3bucket
end-declarations

! Configure 'mybucket' with our S3 access credentials
mybucket.url := S3_BUCKET_URL
mybucket.region := S3_REGION
mybucket.accesskeyid := S3_ACCESS_KEY_ID
mybucket.secretkey := S3_SECRET_KEY

! Upload local file to remote object
s3putobject( mybucket, OBJECT_KEY, LOCAL_FILE )

! Check for errors
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3 service: ", s3getlasterror(mybucket))
  exit(1)
end-if

end-model
```

4.3 Listing S3 object keys

This example demonstrates listing the key names and last-modified dates of all object keys in our bucket that start with `"MyPrefix/"`.

Note that the `s3listobjects` procedure fetches object keys in batches of up to `1000` - you will need to keep calling this procedure until the `istruncated` field is `false` to ensure you process all the keys.

You will need to fill in the model parameters with your own Amazon S3 access credentials.

```
model S3ListExample
uses "s3"

parameters
  S3_BUCKET_URL = ""
  S3_REGION = ""
  S3_ACCESS_KEY_ID = ""
  S3_SECRET_KEY = ""
end-parameters

declarations
  PREFIX="MyPrefix/"
  mybucket: s3bucket
  objslistresult: s3objectlist
end-declarations

! Configure 'mybucket' with our S3 access credentials
mybucket.url := S3_BUCKET_URL
mybucket.region := S3_REGION
mybucket.accesskeyid := S3_ACCESS_KEY_ID
mybucket.secretkey := S3_SECRET_KEY

repeat
  ! Request objects list
  s3listobjects( mybucket, PREFIX, "", objslistresult )
```

```
  ! Check for errors
  if s3status(mybucket)<>S3_OK then
    writeln("Error returned by S3 service: ", s3getlasterror(mybucket))
    exit(1)
  end-if

  ! Process objects returned
  forall (o in objslistresult.objects) do
    writeln('Object key:',o.key,', last-modified:',o.lastmodified)
  end-do

  ! Repeat until we've fetched the last batch of object keys
until not objslistresult.istruncated

end-model
```

CHAPTER 5

Types

s3bucket : `type`

Represents an S3 bucket, including bucket URL, key prefix, login credentials and error state.

url : `text`

The URL of the S3 bucket

region : `text`

The AWS region name for the S3 bucket

keyprefix : `text`

The key prefix to use; will be prepended to all key names you pass to S3 functions using this s3bucket.

accesskeyid : `text`

The access key ID to use for authenticating against the AWS S3 service. Write-only; value cannot be read.

secretkey : `text`

The secret key to use for authenticating against the AWS S3 service. Write-only; value cannot be read.

sessiontoken : `text`

The session token value to use for authenticating against the AWS S3 service. Write-only; value cannot be read.

sse : `text`

The server-side encryption type to use for storing data in S3, if any. Write-only; value cannot be read.

ssekmskeyid : `text`

The ID of the AWS KMS key to use for server-side encryption of data stored in S3, if any. Write-only; value cannot be read.

ssecontext : `text`

The context valueto use for server-side encryption of data stored in S3, if any. Write-only; value cannot be read.

refreshfunc : `function(s3bucket):boolean`

A function pointer that will be called to update the S3 bucket properties when more than `ttl` seconds have elapsed since `timestamp`. The function should update the required properties of the `s3bucket`, including `timestamp`, and return `true` on success or `false` on error. Optional.

timestamp : `real`

The timestamp at which the credentials were generated, expressed in seconds since 1/1/1970.

ttl : `integer`

The number of seconds for which the bucket credentials will remain valid.

s3objectinfo : `record`

Record type representing basic information about an S3 object

key : `text`

The object key. If the s3bucket is configured with a keyprefix, the prefix will not be included in this value.

etag : `text`

The 'entity tag' - a hash reflecting the content of the object.

lastmodified : `datetime`

The Last-Modified date for the object

owner : `s3owner`

The owner of the S3 object

size : `real`

The size of the S3 object, in bytes.

storageclass : `text`

The class of storage being used. Currently known values are STANDARD, STANDARD_IA, GLACIER and RRS.

s3objectlist : `record`

Record type representing a list of S3 objects.

objects : `list of s3objectinfo`

List of basic information records for a collection of S3 objects, as returned by s3listobjects

commonprefixes : `list of text`

List of the 'common prefixes' returned by s3listobjects, if any

istruncated : `boolean`

Flag indicating whether the list of objects has been truncated. If true, you should call 's3listobjects' again, passing the same arguments and s3objectlist record structure, after processing the data in the 'objects' and 'commonprefixes' fields.

s3objectmetadata : `record`

Record type representing object meta-data for an S3 object

cachecontrol : `text`

The Cache-Control header used in requests for the object

contentdisposition : `text`

The Content-Disposition header used in requests for the object

contentencoding : `text`

The Content-Encoding header used in requests for the object

contentlength : `real`

The Content-Length header used in requests for the object. This value is ignored by the 's3putobject' subroutine.

contenttype : `text`

The Content-Type header used in requests for the object

lastmodified : `text`

The Last-Modified date for the object, in text format as returned by the server, e.g. "Wed, 12 Oct 2009 17:50:00 GMT". This value is ignored by the 's3putobject' subroutine.

expiration : `boolean`

Boolean value set to 'true' if the object has a configured expiration action. This value is ignored by the 's3putobject' subroutine

expirationdetails : `text`

If the expiration field is 'true', the text content of the 'expiration' header. This includes an 'expiry-date' component and a URL-encoded 'rule-id' component. This value is ignored by the 's3putobject' subroutine

deletemarker : `boolean`

Flag indicating whether the object was a delete-marker. This value is ignored by the 's3putobject' subroutine

usermetadata : `array(s3usermetadatakeys) of text`

Array of user-defined meta-data fields

replicationstatus : `string`

Replication status, if the object is in a bucket which is the source or destination of a cross-region replication. When the object is in the source bucket, this will be PENDING, COMPLETED or FAILED. When the object is in the destination bucket, this will be REPLICA if the object is a replica created by Amazon S3. When the object is not in a replication bucket, this will be an empty string. This value is ignored by the 's3putobject' subroutine.

serversideencryption : `text`

Where server-side encryption is enabled, the name of the encryption algorithm used, otherwise an empty string.

ssekmskeyid : `text`

Where server-side encryption type 'aws:kms' is used, the ID of the encryption key in the Amazon Key Management Service

storageclass : `string`

The class of storage being used. Currently known values are STANDARD, STANDARD_IA, GLACIER and RRS.

taggingcount : `integer`

The count of tags associated with the object. This value is ignored by the 's3putobject' subroutine.

versionid : `text`

If the object has a unique version ID, that version ID, otherwise empty string. This value is ignored by the 's3putobject' subroutine.

etag : `text`

The 'entity tag' - a hash reflecting the content of the object. This value is ignored by the 's3putobject' subroutine.

websiteredirect : `text`

When bucket is configured as a website, this metadata will evaluate the request as a 301 redirect to another object in the same bucket, or an external URL.

s3owner : `record`

Record type representing information about the owner of an S3 object

id : `text`
displayname : `text`

s3tag : `record`

Record type representing a single S3 object tag.

key : `string`
value : `text`

CHAPTER 6

Variables

s3usermetadatakeys : `set of string`

Set of all keys used in user-defined object metadata

Chapter 7

Subroutines

s3init

Purpose

Initializing an S3 bucket from JSON configuration or within DMP

Synopsis

```
procedure s3init(bucket:s3bucket, cfgid:text)
procedure s3init(bucket:s3bucket, type:text, env:text)
```

Arguments

`bucket`	S3 bucket to be configured
`cfgid`	The ID of the bucket configuration to use. When used within DMP, this may be one of the constants `S3_DMP_SOLUTIONDATA` and `S3_DMP_TENANTSHARED`, or the return value from the `s3solutiondata` or `s3solutionrevision` functions.
`type`	Within DMP, the type of the folder you want to use; the only supported value is the constant `S3_DMP_SOLUTIONDATA`
`env`	Within DMP, the lifecycle stage to access; one of the constants `S3_DMP_DESIGN`, `S3_DMP_STAGING`, `S3_DMP_PRODUCTION`.

Example

Example of initializing an s3bucket to use the DMP `solutionData` folder.

```
declarations
  mybucket: s3bucket
end-declarations
s3init(mybucket,S3_DMP_SOLUTIONDATA)
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
```

Further information

1. After calling, check the value of `s3status` for any errors.
2. If the supplied `s3bucket` has already been initialized, the previous configuration will be overwritten.
3. After a s3bucket has been initialized using `s3init`, it will not be possible to change the `url` or `region` field directly.
4. When used within DMP, this function allows you to access S3 folders provided by the platform. You will pass the constant `S3_DMP_SOLUTIONDATA` or `S3_DMP_TENANTSHARED` to access the default solution-data or tenant-global folders respectively, or the return value of the functions `s3solutiondata` or `s3solutionrevision` to access solution data of a specific environment or assets of a solution revision.
5. The 3-argument version of this function is deprecated since version 0.1.0 of the S3 module. It was previously used to specify the environment of the solution data folder to access; new code should use the 2-argument `s3init` procedure with the return value from `s3solutiondata` instead.
6. When the model is running in Xpress Workbench within DMP, only the S3_DMP_SOLUTIONDATA folder type is supported and the third parameter must not be specified (ie only access to the ”solution data” folder for the current lifecycle stage is supported).

Related topics

`s3status`, `s3getlasterror`, `s3solutiondata`, `s3solutionrevision`

s3delobject

Purpose

Deletes an object from the S3 bucket

Synopsis

```
procedure s3delobject(bucket:s3bucket, objectkey:text)
```

Arguments

`bucket`	The s3bucket object describing the bucket to access
`objectkey`	The key of the object to delete

Example

```
s3delobject(mybucket,"my/file.dat")
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
```

Further information

1. After calling, check the value of `s3status` for any errors.
2. The procedure will not return until the object has been deleted or an error detected.
3. If the s3bucket has a configured keyprefix, it will be prepended to the objectkey passed in.

Related topics

`s3status`

s3getlasterror

Purpose

Returns a string describing the error that occurred during the most recent operation on the s3bucket.

Synopsis

```
function s3getlasterror(bucket:s3bucket):text
```

Return value

A text value containing the error message, which will be empty if the most recent operation on the s3bucket succeeded.

Further information

1. After every call to an S3-related function or procedure, you should check the value of `s3status` 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 `s3_verbose` to true, or inspecting the return value of `s3geterrormsg`.

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 `s3status` instead.

Related topics

`s3status` `s3_verbose`

s3getobject

Purpose

Copies an object from the S3 bucket to a local Mosel file.

Synopsis

```
procedure s3getobject(bucket:s3bucket, objectkey:text, dstfname:text,
      dstobjectmetadata:s3objectmetadata)
procedure s3getobject(bucket:s3bucket, objectkey:text, dstfname:text)
```

Arguments

`bucket`	The s3bucket object describing the bucket to access
`objectkey`	The key of the object to retrieve
`dstfname`	Filename into which to download the object, e.g. ”myfile.dat” or ”mmsystem.text:filedata”
`dstobjectmetadata`	Record into which the object meta-data is copied

Example

```
declarations
  myfilecontent: text
end-declarations
s3getobject(mybucket,"my/file.dat","mmsystem.text:myfilecontent")
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
writeln("Fetched data: ",myfilecontent)
```

Further information

1. After calling, check the value of `s3status` for any errors.
2. The procedure will not return until the object has been downloaded or an error detected.
3. If the s3bucket has a configured keyprefix, it will be prepended to the objectkey passed in.

Related topics

`s3status` `s3putobject` `s3objectmetadata`

s3getobjectmetadata

Purpose

Requests the meta-data for the given object

Synopsis

```
function s3getobjectmetadata(bucket:s3bucket,
       objectkey:text):s3objectmetadata
```

Arguments

`bucket`	The s3bucket object describing the bucket to access
`objectkey`	The key of the object to retrieve

Example

```
declarations
  mymetadata: s3objectmetadata
end-declarations
mymetadata := s3getobjectmetadata(mybucket,"my/file.dat")
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
writeln("my/file.dat last modified: ",mymetadata.lastmodified)
```

Further information

1. After calling, check the value of `s3status` for any errors.
2. The procedure will not return until the object metadata has been downloaded or an error detected.
3. If the s3bucket has a configured keyprefix, it will be prepended to the objectkey passed in.

Related topics

`s3status` `s3putobject` `s3objectmetadata`

s3getobjecttagging

Purpose

Requests the tagset for an object

Synopsis

```
function s3getobjecttagging(bucket:s3bucket, objectkey:text):list of s3tag
```

Arguments

`bucket`	The s3bucket object describing the bucket to access
`objectkey`	The key of the object to query

Example

```
declararations
  tags: list of s3tag
end-declarations
tags := s3getobjecttagging(mybucket,"my/file.dat")
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
```

Further information

1. After calling, check the value of `s3status` for any errors.
2. The procedure will not return until the tags have been fetched or an error detected.
3. If the s3bucket has a configured keyprefix, it will be prepended to the objectkey passed in.

Related topics

`s3status` `s3setobjecttagging`

s3listobjects

Purpose

Lists the objects in the bucket

Synopsis

```
procedure s3listobjects(bucket:s3bucket, prefix:text, delimiter:text,
       result:s3objectlist)
procedure s3listobjects(bucket:s3bucket, prefix:text, result:s3objectlist)
```

Arguments

`bucket`	The s3bucket object describing the bucket to access
`prefix`	The object prefix to search for
`delimiter`	The object delimiter, or "" for no delimiter. If a delimiter is specified, then any objects containing this string after the prefix will not be returned in result.objects, but each unique string between the prefix and the next occurrence of this delimiter will be returned in result.commonprefixes
`result`	Record structure to retrieve

Example

```
declarations
  myresult: s3objectlist
end-declarations
repeat
  s3listobjects( mybucket, "myprefix/", myresult )
  if s3status(mybucket)<>S3_OK then
    writeln("Error returned by S3: ",s3getlasterror(mybucket))
    exit(1)
  end-if
  forall (o in myresult.objects) do
    writeln('Found object with key ',o.key)
  end-do
until not myresult.istruncated
```

Further information

1. After calling, check the value of `s3status` for any errors.
2. As the number of objects in an S3 bucket can be very large, this function may return a truncated list of the keys. If this is the case, the `istruncated` flag of the `s3objectlist` will be set to 'true' and you should call `s3listobjects` again, passing the same record, to retrieve the next batch of object keys.
3. If the s3bucket has a configured keyprefix, it will be prepended to the prefix specified by the `s3listobjects` request
4. If the s3bucket has a configured keyprefix, it will be stripped from the keys and prefixes returned in the `s3objectlist`

Related topics

`s3status` `s3_maxkeys` `s3objectlist`

s3newtag

Purpose

Creates an s3tag record with the given key and value

Synopsis

```
function s3newtag(key:string, value:text):s3tag
function s3newtag(key:text, value:text):s3tag
```

Example

```
s3setobjecttagging(mybucket,"my/file.dat",[
  s3newtag("firstname","John"),
  s3newtag("lastname","Smith")])
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
```

Further information

This would most often be used when constructing a tag list to pass to `s3setobjecttagging`

Related topics

`s3setobjecttagging`

s3objectexists

Purpose

Checks if an S3 object exists

Synopsis

```
function s3objectexists(bucket:s3bucket, objectkey:text):boolean
```

Arguments

`bucket`	The s3bucket object describing the bucket to access
`objectkey`	The key of the object to check

Return value

true if an object with the given key exists in the bucket, false if it does not.

Example

```
declarations
  ifexists: boolean
end-declarations
ifexists := s3objectexists(mybucket,"my/file.dat")
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
if ifexists then
  writeln("Object exists")
else
  writeln("Object does not exist")
end-if
```

Further information

1. After calling, check the value of `s3status` for any errors.
2. If the s3bucket has a configured keyprefix, it will be prepended to the objectkey passed in.

Related topics

`s3status`

s3putobject

Purpose

Copies an object from a local Mosel file to an object in the S3 bucket

Synopsis

```
procedure s3putobject(bucket:s3bucket, objectkey:text, srcfname:text,
      metadata:s3objectmetadata)
procedure s3putobject(bucket:s3bucket, objectkey:text, srcfname:text)
```

Arguments

`bucket`	The s3bucket object describing the bucket to access
`objectkey`	The key of the object to write to
`srcfname`	Filename containing the object content, e.g. ”myfile.dat” or ”mmsystem.text:filedata”
`metadata`	Record describing the object meta-data.

Example

```
s3putobject(mybucket,"my/file.dat","mmsystem.text:myfilecontent")
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
```

Further information

1. After calling, check the value of `s3status` for any errors.
2. The procedure will not return until the object has been uploaded or an error detected.
3. If the s3bucket has a configured keyprefix, it will be prepended to the objectkey passed in.
4. The local file may be opened & read multiple times; this procedure cannot be used with an I/O driver that does not return the same content each time the filename is opened.
5. When you pass an object meta-data record, only some of the fields are sent to the server - see the `s3objectmetadata` documentation for full details. In addition, any fields that are empty will not be sent to the server, with the exception of entries in the `usermetadata` array

Related topics

`s3status` `s3getobject` `s3getobjectmetadata` `s3objectmetadata`

s3setobjecttagging

Purpose

Updates the tagging of the given object.

Synopsis

```
procedure s3setobjecttagging(bucket:s3bucket, objectkey:text, tags:list of
      s3tag)
```

Arguments

`bucket`	The s3bucket object describing the bucket to access
`objectkey`	The key of the object to write to
`tagging`	The collection of tags to assign to the object. Any pre-existing tags not in this list will be removed.

Example

```
declarations
  lst: list of s3tag
end-declarations
s3setobjecttagging(mybucket,"my/file.dat",lst)
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
```

Further information

1. After calling, check the value of `s3status` for any errors.
2. The procedure will not return until the tags have been updated or an error detected.
3. If the s3bucket has a configured keyprefix, it will be prepended to the objectkey passed in.

Related topics

`s3status` `s3getobjecttagging`

s3solutiondata

Purpose

Generate an ID referring to a DMP solutionData folder for a given lifecycle stage, for use with `s3init`

Synopsis

```
function s3solutiondata(env:text):text
```

Argument

`env` The lifecycle environment of the solutionData folder; one of the constants `S3_DMP_DESIGN`, `S3_DMP_STAGING`, `S3_DMP_PRODUCTION`

Return value

A bucket configuration ID that can be passed to `s3init`

Example

```
declarations
  mybucket: s3bucket
end-declarations
s3init(mybucket,s3solutiondata(S3_DMP_STAGING))
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
```

Further information

1. Within DMP, this function can be used to generate an ID for the ”solution data” folder of a given lifecycle environment.
2. This function is not available in the Xpress Workbench version 3.3.3 component in DMP.

Related topics

`s3init`

s3solutionrevision

Purpose

Generate an ID referring to a DMP solution revision folder for a given revision, for use with `s3init`

Synopsis

```
function s3solutionrevision(revision:text):text
```

Argument

`revision` The ID of a revision of the solution in which the component is running

Return value

A bucket configuration ID that can be passed to `s3init`

Example

```
declarations
  mybucket: s3bucket
end-declarations
s3init(mybucket,s3solutionrevision('cl07nssyoo'))
if s3status(mybucket)<>S3_OK then
  writeln("Error returned by S3: ",s3getlasterror(mybucket))
  exit(1)
end-if
```

Further information

1. Within DMP, this function can be used to generate an ID that refers to the assets folder of a specific committed revision of the current solution.
2. This function is not available in the Xpress Workbench version 3.3.3 component in DMP.

Related topics

`s3init`

s3status

Purpose

Indicates the status of the most recent request this model has made using the s3bucket

Synopsis

```
function s3status(bucket:s3bucket):integer
```

Return value

One of the following constants:

`S3_OK` The operation completed successfully.

`S3_NOT_FOUND` The requested object was not found in the S3 bucket.

`S3_ACCESS_DENIED` User is not authorized to access the S3 bucket.

`S3_CONNECTION_ERROR` Unable to connect to the S3 service.

`S3_SERVICE_ERROR` The S3 service returned an unexpected error code.

`S3_IO_ERROR` The S3 module encountered an error reading or writing local files.

`S3_PARSE_ERROR` The S3 module did not understand the response from the S3 service

Further information

After every call to an S3-related function or procedure, you should check the value of `s3status` 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 's3_verbose' to true, or inspecting the return value of `s3getlasterror`.

Related topics

`s3getlasterror`

CHAPTER 8

Parameters

Via the `getparam` function and the `setparam` procedure it is possible to access the following control parameters of module *s3* :

s3_verbose

Description Set to 'true' to activate additional logging of S3-related actions and errors, to the model's error stream.

Type Boolean, read/write

Default value `false`

s3_trace

Description Set to 'true' to activate logging of all S3-related HTTP requests, and corresponding responses, to the model's error stream.

Type Boolean, read/write

Default value `false`

Note This can be useful when troubleshooting S3 authentication or other communication issues.

Note For security reasons, this feature is disabled when Mosel is used within DMP.

s3_maxkeys

Description Sets the maximum number of keys that can be returned by `s3listobjects`. If the query would return more than this number of keys, the istruncated field of the s3objectslist record will be set to 'true' and you should call s3listobjects again to retrieve the next batch of keys.

Type Integer, read/write

Default value `1000`

Note Values less than 1, or greater than 1000, will be ignored.

s3_buckets

Description This allows bucket credentials 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.

s3_max_retries

Description Set the maximum number of times a 'failed' S3 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 `s3_retry_error_codes` parameter.

s3_retry_error_codes

Description Comma-separated list of the AWS error codes that should be retried.

Type String, read/write

Default value

`XpressCommunicationError,ExpiredToken,AccessDenied,InternalError,InvalidAccess`

Note Controls which errors to retry when `s3_max_retries` is non-zero.

Note Supported error codes are as given in the AWS REST API documentation, plus the custom error code `XpressCommunicationError`.

Note This list will only be used in a model compiled against S3 version 0.0.10 or greater (included with Xpress 8.5.3 and greater), and if the `s3_parse_error_response` parameter is `true`.

s3_parse_error_response

Description Whether to try to parse the AWS error code out of the AWS error response document.

Type Boolean, read/write

Default value `true`

Note If this is false, then you will not be able to control which errors to retry using the `s3_retry_error_codes` parameter

Note Disable this parameter if your S3 service is returning invalid error responses.

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