DMP Module for Mosel

Developers Guide

9.7

REFERENCE MANUAL

FICO® Xpress Optimization

FICO

FICO® Xpress Optimization 9.7

Deliverable Version: A

Last Revised: 18 February, 2025

Contents

CHAPTER 1

Introduction

The dmp module allows a Mosel model running in a FICO® Xpress Insight or FICO® Xpress Executor component in Decision Management Platform (DMP) to interact with components and webservices within the same DMP Solution, and services on DMP Manager. Using the dmpresource type supplied by the dmp module, you will be able to make HTTP requests to these resources that will be automatically authorized with appropriate credentials, for example:

```
declarations
  mycomponent: dmpresource
end-declarations
dmpinitcomp(mycomponent, "Xpress Executor")
if mycomponent.status<>DMP_OK then
  writeln('Component not found')
else
  httpStatusCode := dmphttpget(mycomponent, "/rest/runtime/status", "status.json")
end-if
```

The dmp module is the only way to communicate with DMP services in Xpress Insight 4 or Xpress Executor. In Xpress Insight 5 it is not required to use the dmp module, as authorization tokens will be returned by the insightgetcontext function of mminsight and can then be used to authorize webservice calls using mmhttp or some other method, but in many cases it is still easier to use the functions of the dmp module instead.

This manual will not cover how to interact with specific DMP webservices; the developer should consult the documentation for the resource they are trying to call for details of how to formulate the HTTP request and interpret the response.

CHAPTER 2

Using the DMP module

2.1 Accessing DMP environmental data

One of the simplest uses of the `dmp` module is to access information about the component instance that's executing your model. You can use `getdmpcompid`, `getdmpsolid` and `getdmplifecycleenv` to retrieve the DMP component ID, solution ID and lifecycle stage respectively. You can also use the `isdmp` function to check whether or not the model is being executed by a DMP component. For example:

```
if isdmp then
  writeln('Component ID:', getdmpcompid)
  writeln('Solution ID:', getdmpsolid)
  writeln('Lifecycle Stage:', getdmplifecycleenv)
else
  writeln('Not within DMP')
end-if
```

2.2 Making HTTP Requests

You can use the functions `dmphttpget`, `dmphttppatch`, `dmphttppost`, `dmphttpput`, `dmphttpdel` and `dmphttphead` to send HTTP requests to a DMP component, webservice or DMP Manager. These functions work much the same as the corresponding functions in `mmhttp`, except instead of passing a destination URL, you pass a destination `dmpresource` and path relative to this.

In event of an unexpected failure, the request can be retried a number of times until it succeeds, the delay between each retry growing progressively larger. When a `dmphttp` request returns an HTTP status code found in `dmp_retry_error_codes`, it will be retried, up to the number of times specified by `dmp_max_retries`-a value of zero disables the feature.

2.3 Calling a DMP Component

The most common use of the `dmp` module will be to send HTTP requests to another component in the same DMP solution. To do this, you first initialize a `dmpresource` value with the details of the target component by calling `dmpinitcomp`, then use the `dmphttp` functions to send HTTP requests in a similar way to using plain `mmhttp` (see section 2.2 for details). For example, in an Xpress Insight app, you can send a webservice request to the **status** endpoint of an Xpress Executor component in the same solution as follows:

```
declarations
  mycomp: dmpresource
end-declarations
dmpinitcomp(mycomp,'Xpress Executor')
if mycomp.status<>DMP_OK then
  writeln('Failed to find component due to error:', mycomp.lasterror)
  exit(1)
end-if
```

```
httpStatusCode := dmphttpget(mycomp,'/rest/runtime/status','result.json')
if httpStatusCode<>200 then
  writeln('Error returned by component: ',httpStatusCode)
  exit(1)
end-if
```

The path you pass to the the `dmphttp` function will be relative to the root of the component instance URL.

When executed from a DMP component, the `dmp` module will take care of obtaining appropriate authorization credentials and including them in the outgoing HTTP requests.

2.4 Finding DMP Components

In most cases, when you want to communicate with a DMP component in the same solution, you will either know the name of the component or you know there will only be a single component of that type, and you can pass those details directly to `dmpinitcomp`. But sometimes you may need to use more complex logic for finding the component; for these cases, `getdmpcomponents` can be used to return a list of all the components in the DMP solution, and you can iterate through them. For example, if you want to initialize an "Xpress Executor" component with a name that ends with "Simulation":

```
declarations
  comps: list of dmpcomponent
  mycomp: dmpresource
end-declarations

comps := getdmpcomponents
forall (c in comps) do
  if endswith(c.type,"Xpress Executor") and endswith(c.name,"Simulation") then
    dmpinitcomp(mycomp,c)
    if mycomp.status<>DMP_OK then
      writeln('Failed to find component due to error:', mycomp.lasterror)
      exit(1)
    end-if
    break
  end-if
end-do
```

2.5 Calling a DMP Webservice

You can also use the `dmp` module to send HTTP requests to a `SERVERLESS_REST` type webservice in the same solution. To do this, you first initialize a `dmpresource` value with the details of the webservice you want to talk to by calling `dmpinitwebservice`, then use the `dmphttp` functions to send HTTP requests in a similar way to using plain `mmhttp` (see section 2.2 for details). For example, you can send a request to a webservice called **processLoanApps** as follows:

```
declarations
  myservice: dmpresource
end-declarations
dmpinitwebservice(myservice,'processLoanApps')
if myservice.status<>DMP_OK then
  writeln('Failed to find service due to error:', myservice.lasterror)
  exit(1)
end-if

httpStatusCode := dmphttppost(myservice,'','request.json','result.json')
if httpStatusCode<>200 then
  writeln('Error returned by webservice: ',httpStatusCode)
  exit(1)
end-if
```

When calling a webservice, you will typically leave the path field as an empty string. In Xpress Insight 4 and Xpress Executor you may only make HTTP POST requests to this path and HTTP GET requests to the paths `/api/swagger.json` and `/api/log`; in Xpress Insight 5 there is no restriction on which paths may be accessed.

2.6 Calling DMP Manager

You can also use the `dmp` module to send HTTP requests to DMP Manager. Initialize a `dmpresource` value with the details of DMP Manager by calling `dmpinitmanager`, then use the `dmphttp` functions to send HTTP requests in a similar way to using plain `mmhttp` (see section 2.2 for details). For example, you can send a request to create a commit of the current solution as follows:

```
public declarations
  dmpmanager: dmpresource
  REQUEST_BODY = '{"label":"EXAMPLE","comment":"this is an example"}'
end-declarations
dmpinitmanager(dmpmanager)
if dmpmanager.status<>DMP_OK then
  writeln('Failed to find DMP Manager due to error:', dmpmanager.lasterror)
  exit(1)
end-if

httpStatusCode := dmphttppost(dmpmanager,'/rest/dmp/runtime/solutions/'+getdmpsolid+'/revisions',
  'text:REQUEST_BODY','result.json')
if httpStatusCode<>200 then
  writeln('Error returned by DMP Manager: ',httpStatusCode)
  exit(1)
end-if
```

In Xpress Insight 4 and Xpress Executor, the set of paths you can call on DMP Manager is restricted as follows:

- You may only make requests to paths pertaining to the solution containing the component executing the model.
- You may make HTTP POST and HTTP GET requests to the `/rest/dmp/runtime/solutions/SOLUTIONID/revisions` endpoint.
- You may make HTTP GET requests to the `/rest/dmp/runtime/solutions/SOLUTIONID/revisions/REVISIONID` endpoint.
- You may make HTTP GET requests to the `/rest/dmp/runtime/solutions/SOLUTIONID/revisions/REVISIONID/descriptor` endpoint.
- You may make HTTP GET requests to the `/rest/dmp/runtime/solutions/SOLUTIONID/lifecycle` endpoint.
- You may make HTTP GET requests to the `/rest/dmp/runtime/solutions/SOLUTIONID/functions` endpoint.
- You may make HTTP POST and HTTP GET requests to the `/rest/dmp/runtime/solutions/SOLUTIONID/services` endpoint.
- You may make HTTP DELETE requests to the `/rest/dmp/runtime/solutions/SOLUTIONID/services/SERVICEID` endpoints.
- You may make any requests to paths starting `/rest/adm/`
- You may not make any HTTP requests to any other paths.

In Xpress Insight 5 there are no restrictions on the paths that can be called, but the requests will be made using the solution token for a given lifecycle environment, which may alter the data returned from some endpoints. In this case, a different environment may be specified as an argument of `dmpinitmanager`.

When calling DMP Manager using the `dmphttp` functions, it is not necessary to start the path with '`/com.fico.dmp.manager`'.

2.7 Calling the Data Pipelines service

This functionality is not available in Insight 5.

You can use the `dmp` module to send HTTP requests to a FICO Data Pipelines service within the solution. To do this, you first initialize a `dmpresource` value by calling `dmpinitdatapipelines`. Then you can read the Data Pipelines service instance ID from the `id` attribute of the `dmpresource`, and use the `dmphttp` functions to send HTTP requests in a similar way to using plain `mmhttp` (see section 2.2 for details). For example, to retrieve the list all the Data Pipelines jobs:

```
declarations
  datapipelines: dmpresource
end-declarations
dmpinitdatapipelines(datapipelines)
if datapipelines.status<>DMP_OK then
  writeln('Failed to find Data Pipelines service due to error:', datapipelines.lasterror)
  exit(1)
end-if

httpStatusCode := dmphttpget(datapipelines,'/rest/service/dmp-system/'+
    datapipelines.id+'/api/v1/jobs','request.json')
if httpStatusCode<>200 then
  writeln('Error returned by Data Pipelines: ',httpStatusCode)
  exit(1)
end-if
```

When calling Data Pipelines, the query path is relative to the root of the Data Pipelines service provider.

The `dmpresource.status` value will be `DMP_NOT_FOUND` if a Data Pipelines service instance does not exist within the solution containing the component.

2.8 Use Outside the Cloud

When not being called from within a DMP Component, the `dmp` module does not give you additional functionality over that supplied by `mmhttp`. However, some users may find it useful to use the `dmp` module outside the cloud when writing libraries to work inside and outside the cloud.

You can use `dmpiniturl` to initialize a DMP resource with a set of user-supplied authorization headers, and then make HTTP requests to that resource using `dmphttp` functions. For example, if you have a local webservice with the URL `http://localhost:8860/` that requires a cookie for authorization, you can call it as follows:

```
declarations
  myurl: dmpresource
  headers: dynamic array(set of string) of text
end-declarations
headers("Cookie") := "SESSIONID=8364825249"
dmpiniturl(myurl, "http://localhost:8860/", headers)
if myurl.status<>DMP_OK then
  writeln('ERROR: ',myurl.lasterror)
  exit(1)
end-if

httpStatusCode := dmphttpget(myurl,'/rest/runtime/status','result.json')
```

```
if httpStatusCode<>200 then
  writeln('Error returned from webservice request: ',httpStatusCode)
  exit(1)
end-if
```

If you need to specify dynamic authorization headers, then you can supply a function pointer instead, and the `dmp` module will automatically call your function when it requires fresh authorizaition headers. For example:

```
function getheaders(res:dmpresource, headers:array(set of string) of text): integer
  headers("X-Request-Timestamp") := string(datetime(SYS_NOW))
  headers("Cookie") := "SESSIONID=8364825249"
  returned := 300  ! use for the next 300 seconds, then call getheaders again
end-function

declarations
  myurl: dmpresource
end-declarations
dmpiniturl(myurl, "http://localhost:8860/", ->getheaders)
if myurl.status<>DMP_OK then
  writeln('ERROR: ',myurl.lasterror)
  exit(1)
end-if
```

You should report a failure within the 'getheaders' function by setting the error status on the `dmpresource`, e.g.:

```
function getheaders(res:dmpresource, headers:array(set of string) of text): integer
  if AUTHORIZATION_TOKEN='' then
    seterror(res, DMP_AUTH_ERROR, "AUTHORIZATION_TOKEN not set")
  else
    headers("Authorization") := "Bearer "+AUTHORIZATION_TOKEN
    returned := 1800  ! Use for 30 minutes
  end-if
end-function
```

2.9 Absolute Request Paths

Normally, the path provided to any of the `dmphttp*` functions is relative to the URL of the `dmpresource`. For example, if the `dmpresource` 'res' references a component with root URL `https://app.dms.usw2.ficoanalyticcloud.com/16m0jgaeei/`, then the following code:

```
httpStatusCode := dmphttpget(res,'/rest/runtime/executions','result.json')
```

will initiate a request to `https://app.dms.usw2.ficoanalyticcloud.com/16m0jgaeei/rest/runtime/executions`. When you are constructing the request path in the model, you will usually be constructing the path relative to the resource. But in some cases, you may want to use a path that was returned by a previous call to the resource, and that path may be relative to the hostname rather than the resource's root. (e.g. `/16m0jgaeei/rest/runtime/executions`) To support this case, if you specify a path that starts with the path from the resource URL, this will be treated as an absolute path and resolved relative to the hostname.

For example:

```
! 'res' is a dmpresource with URL https://app.dms.usw2.ficoanalyticcloud.com/16m0jgaeei

! Fetch https://app.dms.usw2.ficoanalyticcloud.com/16m0jgaeei/rest/runtime/executions
httpStatusCode := dmphttpget(res,'/16m0jgaeei/rest/runtime/executions','result.json')

! Fetch https://app.dms.usw2.ficoanalyticcloud.com/16m0jgaeei/rest/runtime/executions
httpStatusCode := dmphttpget(res,'/rest/runtime/executions','result.json')
```

```
! Fetch https://app.dms.usw2.ficoanalyticcloud.com/16m0jgaeei/16m0j/rest/runtime/executions
httpStatusCode := dmphttpget(res,'/16moj/rest/runtime/executions','result.json')
```

This behaviour can be disabled by setting the `abspath` attribute of the `dmpresource` to 'false', e.g.:

```
! Fetch https://app.dms.usw2.ficoanalyticcloud.com/16m0jgaeei/16m0jgaeei/rest/runtime/executions
res.abspath := false
httpStatusCode := dmphttpget(res,'/16m0jgaeei/rest/runtime/executions','result.json')
```

2.10 Error handling

You should always check the `status` attribute of the `dmpresource` after every `dmpinit` call, to see if it succeeded or not. This status will be `DMP_OK` on success, `DMP_NOT_FOUND` if the resource you requested did not exist, `DMP_ACCESS_DENIED` if you don't have access to the resource you requested; any other values indicate internal errors. When the status is not `DMP_OK`, you can find a human-readable error message in the `lasterror` attribute. For example:

```
declarations
  myservice: dmpresource
end-declarations
dmpinitwebservice(myservice,'processLoanApps')
if myservice.status=DMP_OK then
  writeln('Service found OK')
elif myservice.status=DMP_NOT_FOUND then
  writeln('Service not found')
else then
  writeln('Failed to find service due to error:', myservice.lasterror)
end-if

httpStatusCode := dmphttppost(myservice,'','request.json','result.json')
if httpStatusCode<>200 then
  writeln('Error returned by webservice: ',httpStatusCode)
  exit(1)
end-if
```

Any previous error will always be removed from the `dmpresource` on the next call to a `dmpinit*` or `dmphttp*` subroutine. Code can also use the `seterror` and `clearerror` procedures to set and clear the error status of a `dmpresource` directly.

2.11 Limitations

2.11.1 Use within Submodels

Within an Insight 4 or Executor component, you can only use the `dmp` module from a submodel if the master model also uses the `dmp` module. In the event that your master does not need the `dmp` module, I recommend setting `uses 'dmp'` to the top of it and accessing one of the parameters to prevent the Mosel compiler removing it as an unreferenced module, e.g.

```
uses 'dmp'
setparam('dmp_verbose',false)
```

This is not necessary when the model is executing in Insight 5.

2.11.2 Use within Xpress Workbench

The 'dmp' module cannot be used if the model is running locally in an Xpress Workbench component on

the cloud. However, the 'dmp' module will function correctly when Workbench is being used to debug an Xpress Insight scenario.

2.12 Differences between Insight 4 and 5

Xpress Insight 5 executes scenarios in a new 'trusted code execution' model; as a result, there are small differences between how the DMP module behaves in Insight 5 compared to how the same code behaved in Insight 4.

- Existing Insight 4 apps must be recompiled with at least Xpress 9.2.0 to use the DMP module functionality in Insight 5.
- Insight 4 uses the "component token" to authorize requests to DMP Manager. Insight 5 uses the "solution token" for a single lifecycle, which may affect the data returned by certain endpoints. By default the current lifecycle is used, but a different one may be specified to `dmpinitmanager`.
- Insight 5 does not restrict which endpoints of DMP manager or a DMP solution webservice may be accessed, whereas Insight 4 will only allow calls to a small number of whitelisted paths.
- When the `dmp` module is used from a submodel in Insight 4, it must also be referenced from the master model (see section 2.11.1 above). In Insight 5, this is not required.

CHAPTER 3

Examples

3.1 Calling a REST endpoint on an Xpress Executor component

This example demonstrates fetching the list of executions from an Xpress Executor component in the same solution.

```
model countexecutions
  uses 'dmp','mmxml'

  declarations
    xecomp: dmpresource
    httpstatus: integer
    doc: xmldoc
    nodes: list of integer
  end-declarations

  ! Initialize dmpresource
  dmpinitcomp(xecomp, 'Xpress Executor')
  if xecomp.status<>DMP_OK then
    writeln('ERROR finding Xpress Executor component: ',xecomp.lasterror)
    exit(1)
  end-if

  ! Make request
  httpstatus := dmphttpget(xecomp, '/rest/runtime/execution', 'executions.json')
  if httpstatus<>200 then
    if xecomp.status<>DMP_OK then
      writeln('ERROR authorizing Xpress Executor component: ',xecomp.lasterror)
    else
      writeln('ERROR returned by Xpress Executor component: ',httpstatus)
    end-if
    exit(1)
  end-if

  ! Use mmxml to parse response
  jsonload(doc,'executions.json')
  getnodes(doc,'/jsv/jsv',nodes)
  writeln('Executions found: ',nodes.size)

end-model
```

The use of Xpress Executor webservices here is for example purposes only; we recommend using the `executor` module to interact with Xpress Executor components.

3.2 Calling a REST endpoint on DMP Manager

This example demonstrates calling the REST endpoint on DMP Manager to commit a solution revision.

```
model commitsolution
  uses 'dmp','mmsystem'

  declarations
    xecomp: dmpresource
    httpstatus: integer
    public REQUEST_BODY='{"label":"commitSolutionTest","comment":"dmp module example"}'
  end-declarations

  ! Initialize dmpresource
  dmpinitmanager(xecomp)
  if xecomp.status<>DMP_OK then
    writeln('ERROR finding DMP Manager: ',xecomp.lasterror)
    exit(1)
  end-if

  ! Make request
  httpstatus := dmphttppost(xecomp, '/rest/dmp/runtime/solutions/'+getdmpsolid+'/revisions?async=false',
    'text:REQUEST_BODY', 'response.dat')
  if httpstatus<>200 then
    if xecomp.status<>DMP_OK then
      writeln('ERROR making DMP Manager request: ',xecomp.lasterror)
    else
      writeln('ERROR returned by DMP Manager request: ',httpstatus)
    end-if
    exit(1)
  end-if

  writeln('Committed new solution revision')

end-model
```

Chapter 4

Types

4.1 The `dmpcompinst` type

The `dmpcompinst` type contains metadata about a DMP component instance - ie a copy of a component in a specific lifecycle stage. This is intended for advanced users; the attributes of this type correspond directly to the values returned from the 'componentinstances' endpoint in the DMP Manager webservices and users should consult DMP Manager documentation to understand how to use them.

dmpcompinst : `type`

- **componentid** : `text`
- **id** : `text`
- **name** : `text`
- **description** : `text`
- **url** : `text`
- **internalurl** : `text`
- **proxyurl** : `text`
- **healthcheckrelativeurl** : `text`
- **enginetype** : `text`
- **engineprofile** : `text`
- **engineversion** : `text`
- **scalecount** : `int`
- **parentinstanceid** : `text`
- **systemdata** : `text`
- **containerproviderdata** : `text`
- **containerproviderid** : `text`
- **containerserviceproviderid** : `text`
- **env** : `string`
- **envid** : `text`
- **rowversion** : `int`
- **createdby** : `text`
- **createdts** : `text`
- **lastmodifiedby** : `text`
- **lastmodifiedts** : `text`
- **state** : `string`
- **statedescription** : `text`
- **alternateinstance** : `bool`

multidatacenterinstance : `bool`
currentrevision : `dmpcomprev`

4.2 The `dmpcompinstrev` type

The `dmpcompinstrev` type contains metadata about a revision of a DMP component instance. This is intended for advanced users; the attributes of this type correspond directly to the values returned in the revision structure from the 'componentinstances' endpoint in the DMP Manager webservices and users should consult DMP Manager documentation to understand how to use them.

dmpcompinstrev : `type`

id : `text`
label : `text`
version : `text`
createdts : `text`
createdby : `text`
comment : `text`
componentinstanceid : `text`
componentprototyperevisionid : `text`
componentrevisionid : `text`
fromcomponentinstancerevisionid : `text`
state : `string`
lcmstate : `string`
lcmstatedescription : `text`

4.3 The `dmpcomponent` type

The `dmpcomponent` type contains metadata about a DMP component - the component ID, name, type and the lifecycle environments in which it's available. A `dmpcomponent` contains no authorization token and cannot be used to communicate with the component directly but can be passed to `dmpinitcomp` to initialize a `dmpresource`.

dmpcomponent : `type`

id : `text`

The component ID; note this is different from the component *instance* ID.

name : `text`

The component name.

type : `text`

The component type, e.g. "FICO Xpress Executor".

envs : `set of string`

The DMP lifecycle stages in which instances of this component are available; represented by the following constants:

Values		
	`DMP_ENV_DESIGN`	Design or Root
	`DMP_ENV_STAGING`	Staging
	`DMP_ENV_PRODUCTION`	Production

instances : `list of dmpcompinst`

A list with one entry for each instance of the component.

4.4 The `dmpresource` type

The `dmpresource` type represents a URL and associated authorization credentials. For example, the `dmpresource` initialized by `dmpinitcomp` will contain the URL of the component as well as an appropriate bearer token for authorizing requests. The URL and some other attributes can be read, but the authorization token may not. Other than "abspath", all the attributes of the `dmpresource` type are read-only and cannot be set directly.

dmpresource : `type`

resourcetype : `integer`

The resource type referred to by the object-the value will correspond to one of the constants listed here:

Values	`DMP_RESOURCE_DMP_MANAGER`	DMP Manager
	`DMP_RESOURCE_COMPONENT`	Component
	`DMP_RESOURCE_WEBSERVICE`	Web service
	`DMP_RESOURCE_DATA_PIPELINES`	Data Pipelines service
	`DMP_RESOURCE_USER`	User-specified URL
	`DMP_RESOURCE_UNINITIALIZED`	Uninitialized

url : `text`

The root URL of the DMP resource.

id : `text`

For component resources, this is the component ID. For webservice resources, this is the solution service instance ID. For Data Pipelines resources, this is the solution service ID. For other resource types, this is unset.

name : `text`

For component resources, this is the component name. For webservice resources, this is the function name. For Data Pipelines resources, this is the solution service name. For other resource types, this is unset.

type : `text`

For component resources, this is the component type, e.g. "FICO Xpress Insight". For webservice resources, this is the constant string `"SERVERLESS_REST"`. For Data Pipelines resources, this is the constant string `"DataPipelines"`. For other resource types, this is unset.

env : `string`

The DMP lifecycle stage of the resource being referenced; an empty string when not relevent, otherwise one of the following constants:

Values	`DMP_ENV_DESIGN`	Design or Root
	`DMP_ENV_STAGING`	Staging
	`DMP_ENV_PRODUCTION`	Production

abspath : `boolean`

Flag indicating whether request paths are treated as absolute when they start with the same path as the DMP component's URL, as described in section 2.9. Defaults to true.

status : `integer`

The status of the last `dmpinit` operation called on this `dmpresource`. One of the following constants:

Values

`DMP_OK`	Success
`DMP_ACCESS_DENIED`	Resource found but you do not have access to it
`DMP_NOT_FOUND`	Resource not found
`DMP_IO_ERROR`	Internal error reading or writing data
`DMP_PARSE_ERROR`	Internal error parsing data
`DMP_AUTH_ERROR`	Error fetching authorization credentials from function

context : `any`

A value that can be saved to identify this `dmpresource` or attach additional data. The context value is not used by the `dmp` module directly, but may be to make additional data available to the function that supplies authorization headers.

lasterror : `text`

The error message generated by the last `dmpinit` operation on this `dmpresource`, if it failed.

CHAPTER 5

Procedures and functions

clearerror

Purpose

Clears any error in the `dmpresource`, so that the `status` attribute will be `DMP_OK`.

Synopsis

```
procedure clearerror(res:dmpresource)
```

Argument

`res` The `dmpresource`.

Further information

It is not typically necessary to call this procedure, as the error state of the `dmpresource` is cleared automatically on any call to functions that may set it.

getdmpcomponents

Purpose

Returns a list of metadata about the DMP components available in the current solution.

Synopsis

```
function getdmpcomponents:list of dmpcomponent
```

Return value

List of components, in no defined order

Further information

1. In event of an unexpected failure, the request will be retried the number of times specified by `dmp_max_retries`.
2. The model will terminate with an error if this function is called outside of a supported DMP component.

Example

```
declarations
  components: list of dmpcomponent
end-declarations
components := getdmpcomponents
forall (comp in components) do
  writeln('Found component: ',comp.name)
end-do
```

dmphttpdel

Purpose

Make an HTTP DELETE request to path within an initialized `dmpresource`.

Synopsis

```
function dmphttpdel(res:dmpresource, path:text, result:text):integer
function dmphttpdel(res:dmpresource, path:text, result:text,
       xhdr:text):integer
```

Arguments

`res`	The `dmpresource` we want to access.
`path`	The path we want to access within the DMP resource
`result`	File to store the result of the request
`xhdr`	Additional headers to add to the request

Return value

The HTTP status code of the request, or 999 on a connection error.

Further information

1. This function corresponds to the `httpdel` function of the `mmhttp` module and works in a similar way.
2. The request path is relative to the DMP resource's URL, unless it starts with the same path as the DMP resource URL and the `abspath` attribute has not been set to `false`, in which case it will be treated as an absolute path.
3. The outgoing HTTP request will be automatically augmented with the appropriate credentials for this type of `dmpresource`.
4. In event of an unexpected failure, the request will be retried the number of times specified by `dmp_max_retries` if it returned a HTTP status code listed in `dmp_retry_error_codes`.
5. If this function returns an unexpected HTTP status code, you should check the `status` attribute of the `dmpresource` to see whether this was returned from the HTTP webservice or was the result of an error obtaining credentials.

Example

```
declarations
  res: dmpresource
  httpstatus: integer
  EXEC_ID='5f164857-f27f-44f8-a773-50dedf4f2724'
end-declarations
dmpinitcomp(res,"Xpress Executor")

httpstatus := dmphttpdel(res,'/rest/runtime/execution/'+EXEC_ID,'result.txt')
if httpstatus<>200 then
  if res.status<>DMP_OK then
    writeln('DMP Resource error: ',res.lasterror)
  else
    writeln('Unexpected HTTP status code: ',httpstatus)
  end-if
end-if
```

dmphttpget

Purpose

Make an HTTP GET request to path within an initialized `dmpresource`.

Synopsis

```
function dmphttpget(res:dmpresource, path:text, result:text):integer
function dmphttpget(res:dmpresource, path:text, result:text,
       xhdr:text):integer
```

Arguments

`res`	The `dmpresource` we want to access.
`path`	The path we want to access within the DMP resource
`result`	File to store the result of the request
`xhdr`	Additional headers to add to the request

Return value

The HTTP status code of the request, or 999 on a connection error.

Further information

1. This function corresponds to the `httpget` function of the *mmhttp* module and works in a similar way.
2. The request path is relative to the DMP resource's URL, unless it starts with the same path as the DMP resource URL and the `abspath` attribute has not been set to `false`, in which case it will be treated as an absolute path.
3. The outgoing HTTP request will be automatically augmented with the appropriate credentials for this type of `dmpresource`.
4. In event of an unexpected failure, the request will be retried the number of times specified by `dmp_max_retries` if it returned a HTTP status code listed in `dmp_retry_error_codes`.
5. If this function returns an unexpected HTTP status code, you should check the `status` attribute of the `dmpresource` to see whether this was returned from the HTTP webservice or was the result of an error obtaining credentials.

Example

```
declarations
  res: dmpresource
  httpstatus: integer
end-declarations
dmpinitcomp(res,"Xpress Executor")

httpstatus := dmphttpget(res,'/rest/runtime/execution','executions.json')
if httpstatus<>200 then
  if res.status<>DMP_OK then
    writeln('DMP Resource error: ',res.lasterror)
  else
    writeln('Unexpected HTTP status code: ',httpstatus)
  end-if
end-if
```

dmphttphead

Purpose

Make an HTTP HEAD request to path within an initialized `dmpresource`.

Synopsis

```
function dmphttphead(res:dmpresource, path:text, result:text):integer
function dmphttphead(res:dmpresource, path:text, result:text,
       xhdr:text):integer
```

Arguments

`res`	The `dmpresource` we want to access.
`path`	The path we want to access within the DMP resource
`result`	File to store the result of the request
`xhdr`	Additional headers to add to the request

Return value

The HTTP status code of the request, or 999 on a connection error.

Further information

1. This function corresponds to the `httphead` function of the `mmhttp` module and works in a similar way.
2. The request path is relative to the DMP resource's URL, unless it starts with the same path as the DMP resource URL and the `abspath` attribute has not been set to `false`, in which case it will be treated as an absolute path.
3. The outgoing HTTP request will be automatically augmented with the appropriate credentials for this type of `dmpresource`.
4. In event of an unexpected failure, the request will be retried the number of times specified by `dmp_max_retries` if it returned a HTTP status code listed in `dmp_retry_error_codes`.
5. If this function returns an unexpected HTTP status code, you should check the `status` attribute of the `dmpresource` to see whether this was returned from the HTTP webservice or was the result of an error obtaining credentials.

Example

```
declarations
  res: dmpresource
  httpstatus: integer
end-declarations
dmpinitcomp(res,"Xpress Executor")

httpstatus := dmphttphead(res,'/rest/runtime/execution','null:')
if httpstatus<>200 then
  if res.status<>DMP_OK then
    writeln('DMP Resource error: ',res.lasterror)
  else
    writeln('Unexpected HTTP status code: ',httpstatus)
  end-if
end-if
```

dmphttppatch

Purpose

Make an HTTP PATCH request to path within an initialized `dmpresource`.

Synopsis

```
function dmphttppatch(res:dmpresource, path:text, data:text,
       result:text):integer
function dmphttppatch(res:dmpresource, path:text, data:text, result:text,
       xhdr:text):integer
```

Arguments

`res`	The `dmpresource` we want to access.
`path`	The path we want to access within the DMP resource
`data`	Data file to be sent as the request body
`result`	File to store the result of the request
`xhdr`	Additional headers to add to the request

Return value

The HTTP status code of the request, or 999 on a connection error.

Further information

1. This function corresponds to the `httppatch` function of the `mmhttp` module and works in a similar way.
2. The request path is relative to the DMP resource's URL, unless it starts with the same path as the DMP resource URL and the `abspath` attribute has not been set to `false`, in which case it will be treated as an absolute path.
3. The outgoing HTTP request will be automatically augmented with the appropriate credentials for this type of `dmpresource`.
4. In event of an unexpected failure, the request will be retried the number of times specified by `dmp_max_retries` if it returned a HTTP status code listed in `dmp_retry_error_codes`.
5. If this function returns an unexpected HTTP status code, you should check the `status` attribute of the `dmpresource` to see whether this was returned from the HTTP webservice or was the result of an error obtaining credentials.

Example

```
declarations
  res: dmpresource
  httpstatus: integer
end-declarations
dmpinitcomp(res,"Xpress Executor")

httpstatus := dmphttppatch(res,'/rest/runtime/execution',
  'executionrequest.json', 'execution.json')
if httpstatus<>200 then
  if res.status<>DMP_OK then
    writeln('DMP Resource error: ',res.lasterror)
  else
    writeln('Unexpected HTTP status code: ',httpstatus)
  end-if
end-if
```

dmphttppost

Purpose

Make an HTTP POST request to path within an initialized `dmpresource`.

Synopsis

```
function dmphttppost(res:dmpresource, path:text, data:text,
       result:text):integer
function dmphttppost(res:dmpresource, path:text, data:text, result:text,
       xhdr:text):integer
```

Arguments

`res`	The `dmpresource` we want to access.
`path`	The path we want to access within the DMP resource
`data`	Data file to be sent as the request body
`result`	File to store the result of the request
`xhdr`	Additional headers to add to the request

Return value

The HTTP status code of the request, or 999 on a connection error.

Further information

1. This function corresponds to the `httppost` function of the `mmhttp` module and works in a similar way.
2. The request path is relative to the DMP resource's URL, unless it starts with the same path as the DMP resource URL and the `abspath` attribute has not been set to `false`, in which case it will be treated as an absolute path.
3. The outgoing HTTP request will be automatically augmented with the appropriate credentials for this type of `dmpresource`.
4. In event of an unexpected failure, the request will be retried the number of times specified by `dmp_max_retries` if it returned a HTTP status code listed in `dmp_retry_error_codes`.
5. If this function returns an unexpected HTTP status code, you should check the `status` attribute of the `dmpresource` to see whether this was returned from the HTTP webservice or was the result of an error obtaining credentials.

Example

```
declarations
  res: dmpresource
  httpstatus: integer
end-declarations
dmpinitcomp(res,"Xpress Executor")

httpstatus := dmphttppost(res,'/rest/runtime/execution',
  'executionrequest.json', 'execution.json')
if httpstatus<>200 then
  if res.status<>DMP_OK then
    writeln('DMP Resource error: ',res.lasterror)
  else
    writeln('Unexpected HTTP status code: ',httpstatus)
  end-if
end-if
```

dmphttpput

Purpose

Make an HTTP PUT request to path within an initialized `dmpresource`.

Synopsis

```
function dmphttpput(res:dmpresource, path:text, data:text,
       result:text):integer
function dmphttpput(res:dmpresource, path:text, data:text, result:text,
       xhdr:text):integer
```

Arguments

`res`	The `dmpresource` we want to access.
`path`	The path we want to access within the DMP resource
`data`	Data file to be sent as the request body
`result`	File to store the result of the request
`xhdr`	Additional headers to add to the request

Return value

The HTTP status code of the request, or 999 on a connection error.

Further information

1. This function corresponds to the `httpput` function of the `mmhttp` module and works in a similar way.
2. The request path is relative to the DMP resource's URL, unless it starts with the same path as the DMP resource URL and the `abspath` attribute has not been set to `false`, in which case it will be treated as an absolute path.
3. The outgoing HTTP request will be automatically augmented with the appropriate credentials for this type of `dmpresource`.
4. In event of an unexpected failure, the request will be retried the number of times specified by `dmp_max_retries` if it returned a HTTP status code listed in `dmp_retry_error_codes`.
5. If this function returns an unexpected HTTP status code, you should check the `status` attribute of the `dmpresource` to see whether this was returned from the HTTP webservice or was the result of an error obtaining credentials.

Example

```
declarations
  res: dmpresource
  httpstatus: integer
end-declarations
dmpinitcomp(res,"Example Component")

httpstatus := dmphttpput(res,'/rest/design/model',
  'modelbody.dat', 'response.dat')
if httpstatus<>200 then
  if res.status<>DMP_OK then
    writeln('DMP Resource error: ',res.lasterror)
  else
    writeln('Unexpected HTTP status code: ',httpstatus)
  end-if
end-if
```

dmphttprequest

Purpose

Make an HTTP request to path within an initialized `dmpresource`.

Synopsis

```
function dmphttprequest(res:dmpresource, method:integer, path:text,
      data:text, result:text):integer
function dmphttprequest(res:dmpresource, method:integer, path:text,
      data:text, result:text, xhdr:text):integer
```

Arguments

`res`	The `dmpresource` we want to access.
`method`	Code representing the HTTP method to use for the request. Must be one of the following constants exported by `mmhttp`: `HTTP_DELETE`, `HTTP_HEAD`, `HTTP_GET`, `HTTP_PATCH`, `HTTP_POST` or `HTTP_PUT`
`path`	The path we want to access within the DMP resource
`data`	Data file to be sent as the request body; will be ignored for methods that do not send a body.
`result`	File to store the result of the request
`xhdr`	Additional headers to add to the request

Return value

The HTTP status code of the request, or 999 on a connection error.

Further information

1. This function provides a way to make HTTP requests where the method (`GET`, `POST`, etc) is not known at compile time.
2. The request path is relative to the DMP resource's URL, unless it starts with the same path as the DMP resource URL and the `abspath` attribute has not been set to `false`, in which case it will be treated as an absolute path.
3. The outgoing HTTP request will be automatically augmented with the appropriate credentials for this type of `dmpresource`.
4. In event of an unexpected failure, the request will be retried the number of times specified by `dmp_max_retries` if it returned a HTTP status code listed in `dmp_retry_error_codes`.
5. If this function returns an unexpected HTTP status code, you should check the `status` attribute of the `dmpresource` to see whether this was returned from the HTTP webservice or was the result of an error obtaining credentials.

Example

```
declarations
  res: dmpresource
  httpstatus: integer
end-declarations
dmpinitcomp(res,"Xpress Executor")

httpstatus := dmphttprequest(res,HTTP_POST,'/rest/runtime/execution',
  'executionrequest.json', 'execution.json')
if httpstatus<>200 then
  if res.status<>DMP_OK then
    writeln('DMP Resource error: ',res.lasterror)
  else
    writeln('Unexpected HTTP status code: ',httpstatus)
  end-if
end-if
```

dmpinitcomp

Purpose

Initialize a `dmpresource` to access a DMP component.

Synopsis

```
procedure dmpinitcomp(res:dmpresource, type:text)
procedure dmpinitcomp(res:dmpresource, id:text, type:text, name:text,
       env:text)
procedure dmpinitcomp(res:dmpresource, comp:dmpcomponent)
procedure dmpinitcomp(res:dmpresource, comp:dmpcomponent, env:text)
procedure dmpinitcomp(res:dmpresource, inst:dmpcompinst)
```

Arguments

`res`	The `dmpresource` value to initialize.
`id`	The ID of the component you want to access (or empty string)
`type`	The type of the component you want to access (or empty string)
`name`	The name of the component you want to access (or empty string)
`comp`	Value containing the ID, type and name of the component
`inst`	Value containing the component ID and lifecyce environment you want to access
`env`	The lifecycle stage of the component instance you want to access (or empty string), one of: `DMP_ENV_DESIGN` Design or Root `DMP_ENV_STAGING` Staging `DMP_ENV_PRODUCTION` Production

Further information

1. This procedure will initialize a `dmpresource` to access another component in the same solution as the component executing the model.
2. You can pass an empty string if you don't want to specify any of the arguments, but you must specify at least one of `id`, `type`, `name` or `comp`.
3. If you specify a component type and a component of this type does not exist in the solution, this function will look for a component whose type contains the type string you specified (e.g. passing type *Xpress Insight* may return a component of type *Xpress Insight 5* if one exists).
4. Where multiple components in the solution match the values you specified, the `dmpresource` will be initialized to access an arbitrarily selected component.
5. If you don't specify a lifecycle environment, the lifecycle of the component executing the model will be used.
6. When running in a solution with isolated lifecycles, you will only be able to access component instances from the same or lower lifecycle as the current component - e.g. a model run by a component in Staging can access resources in Staging and Design but not resources in Production.
7. You should check the `status` attribute of the `dmpresource` after calling this procedure.

Example

```
declarations
  res: dmpresource
end-declarations
dmpinitcomp(res,"Xpress Executor")
if res.status<>DMP_OK then
  writeln('ERROR: ',res.lasterror)
end-if
```

dmpinitdatapipelines

Purpose

Initialize a `dmpresource` to access the FICO Data Pipelines service.

Synopsis

```
procedure dmpinitdatapipelines(res:dmpresource)
```

Argument

`res` The `dmpresource` value to initialize.

Further information

1. You should check the `status` attribute of the `dmpresource` after calling this procedure.
2. Solutions do not contain a Data Pipelines service by default and this procedure will not create one if it does not already exist. Please consult the FICO Data Pipelines documentation for instructions on adding a Data Pipelines service instance to your solution.
3. HTTP requests using this `dmpresource` will be relative to the root of the Data Pipelines service provider. The ID of the service instance, which typically needs to be invluded in the path of any webservice requests, can be read from the resource's `id` attribute.
4. It is not possible to use this function to locate the Data Pipelines service from Insight 5.

Example

```
declarations
  res: dmpresource
end-declarations
dmpinitdatapipelines(res)
if res.status<>DMP_OK then
  writeln('ERROR: ',res.lasterror)
end-if
```

dmpinitmanager

Purpose

Initialize a `dmpresource` to access DMP Manager.

Synopsis

```
procedure dmpinitmanager(res:dmpresource)
procedure dmpinitmanager(res:dmpresource, env:text)
```

Arguments

`res` The `dmpresource` value to initialize.

`env` The lifecycle stage of the solution credentials to use to access DMP Manager (or empty string) from an Xpress Insight 5 app. Ignored when called within Xpress Insight 4 or Xpress Executor. One of:

`DMP_ENV_DESIGN`	Design or Root
`DMP_ENV_STAGING`	Staging
`DMP_ENV_PRODUCTION`	Production

Further information

1. In Xpress Insight 4 and Xpress Executor, requests to DMP Manager will be authorized using ”component” credentials. In Xpress Insight 5, requests will be authorized using ”solution” credentials for the lifecycle environment passed in the `env` parameter, or the currently executing environment if none is given.

2. You should check the `status` attribute of the `dmpresource` after calling this procedure.

Example

```
declarations
  res: dmpresource
end-declarations
dmpinitmanager(res)
if res.status<>DMP_OK then
  writeln('ERROR: ',res.lasterror)
end-if
```

dmpiniturl

Purpose

Initialize a `dmpresource` with user-supplied URL and credentials

Synopsis

```
procedure dmpiniturl(res:dmpresource, url:text, authtokentype:text,
      authtoken:text)
procedure dmpiniturl(res:dmpresource, url:text, authheaders:array(set of
      string) of text)
procedure dmpiniturl(res:dmpresource, url:text,
      getauthheaders:function(res:dmpresource, authheaders:array(set of
      string) of text):integer
```

Arguments

`res`	The `dmpresource` value to initialize.
`url`	The resource root URL. All dmphttp calls through this resource will be relative to this URL.

`authtokentype`	The type of authorization token, expressed as one of the following constants:
	`DMP_TOKEN_TYPE_BEARER`
	`DMP_TOKEN_TYPE_JWT`

`authtoken`	The authorization token to add to outgoing HTTP requests made through this resource.
`authheaders`	A collection of HTTP Headers to add to outgoing HTTP requests made through this resource.
`getauthheaders`	A function pointer which will be called to fetch the HTTP headers to add to outgoing requests. This function should populate the `authheaders` array with the headers to use and return the number of seconds for which they can be used, or 0 if fresh headers should be requested for every request.

Further information

1. This procedure is intended to assist users to write code interacting with DMP resources that will work both within Xpress DMP components and when executed locally.
2. If you pass an `authtoken` value, all HTTP requests made through this resource will be decorated with an appropriate `Authorization:` header.
3. If you pass an `authheaders` value, all HTTP requests made through this resource will be decorated with HTTP headers read from this array (indices are the header names).
4. If you pass a `getauthheaders` value, this function pointer will be called on the first HTTP request to this resource, and the HTTP request will be decorated with the headers the function writes to the `authheaders` array (indices are header names). The value returned by the function will be a number of seconds; once this timespan has elapsed, the `getauthheaders` function will be called again on the next HTTP request to fetch fresh headers.
5. The `getauthheaders` function can itself make an HTTP request to the `dmpresource` in the usual way (e.g. to request a bearer token); no authorization headers will be added to requests made from within a `getauthheaders` function.
6. In the event of a failure to populate the headers, the `getauthheaders` function should call the `seterror` function to set the error status on the `dmpresource`. Failures to populate authorization headers will automatically be retried up to `max_retries` times.
7. You should check the `status` attribute of the `dmpresource` after calling this procedure.

Example 1

This example demonstrates passing fixed authorization headers to the dmpresource

```
declarations
  res: dmpresource
  headers: dynamic array(set of string) of text
end-declarations
headers("Cookie") := "SESSIONID=8364825249"
headers("Authorization") := "Bearer 834692364jdnvdjfvb7t3jkd78"
dmpiniturl(res, "http://fakeresource.example.com/", headers)
if res.status<>DMP_OK then
  writeln('ERROR: ',res.lasterror)
end-if
```

Example 2

This example demonstrates dynamically requesting a bearer token to attach to requests

```
declarations
  CLIENT_ID="orughu8sbgofb954g6htg0rwehgosdfg"
  SECRET="guhsogbosfgosdngds"
  public request,response: text
  res: dmpresource
end-declarations

function addauthheader(res:dmpresource,
    headers:array(set of string) of text): integer

  ! Make REST request to get bearer token for given client_id/secret
  request := '{"clientId":"'+CLIENT_ID+'", "secret":"'+SECRET+'"}'
  if dmphttppost(res, '/rest/auth/token', "text:request",
      "text:response")<>200 or res.status<>DMP_OK
  then
    writeln('ERROR: Failed to request bearer token')

  else
    ! Bearer token will be in response body
    headers("Authorization") := "Bearer "+response
    ! Bearer token will be usable for 30 minutes
    returned := 1800
  end-if
end-function

dmpiniturl(res, "http://fakeresource.example.com/", ->addauthheader)
if res.status<>DMP_OK then
  writeln('ERROR: ',res.lasterror)
end-if
```

dmpinitwebservice

Purpose

Initialize a `dmpresource` to access a DMP webservice.

Synopsis

```
procedure dmpinitwebservice(res:dmpresource, functionid:text)
procedure dmpinitwebservice(res:dmpresource, id:text, functionid:text,
      env:text)
```

Arguments

`res`	The `dmpresource` value to initialize.
`id`	The ID of the solution service you want to access (or empty string)
`functionid`	The ID of the function you want to access, ie the function name (or an empty string)
`env`	The lifecycle stage of the service you want to access (or empty string), one of: `DMP_ENV_DESIGN` Design or Root `DMP_ENV_STAGING` Staging `DMP_ENV_PRODUCTION` Production

Further information

1. This procedure will initialize a `dmpresource` to access a DMP function deployed as a `SERVERLESS_REST` type solution service. Typically this mean a DMP function deployed to AWS Lambda. It cannot be used with functions deployed as services in any other ways.
2. You can pass an empty string if you don't want to specify any of the arguments, but you must specify at least one of `id` and `functionid`.
3. The `functionid` value is usually the function name.
4. If you don't specify a lifecycle environment, the lifecycle of the component executing the model will be used.
5. When running in a solution with isolated lifecycles, you will only be able to access component instances from the same or lower lifecycle as the current component - e.g. a model run by a component in Staging can access resources in Staging and Design but not resources in Production.
6. You should check the `status` attribute of the `dmpresource` after calling this procedure.

Example

```
declarations
  res: dmpresource
end-declarations
dmpinitwebservice(res,"getBestPolicy")
if res.status<>DMP_OK then
  writeln('ERROR: ',res.lasterror)
end-if
```

getdmpauthfunc

Purpose

In an Insight 5 app, returns a function pointer that can be passed to `dmpiniturl` to supply authorization headers based on the solution token for the given DMP lifecycle environment.

Synopsis

```
function getdmpauthfunc:function(dmpresource,array(string) of text):integer
function getdmpauthfunc(env:string):function(dmpresource,array(string) of
      text):integer
```

Argument

`env` The environment for which to request the function, or empty string for current env.

Return value

A function pointer that can be passed to `dmpiniturl`.

Further information

1. This function should only be used in an Insight 5 app running on DMP.
2. When called from outside an Insight 5 app on DMP, the authorization function pointer returned will generate an error in the `dmpresource` when it is used.

Example

```
declarations
  res: dmpresource
  DMP_MANAGER_URL='https://console.dms.int.usw2.ficoanalyticcloud.com/com.fico.dmp.manager/'
end-declarations
dmpiniturl(res, DMP_MANAGER_URL, getdmpauthfunc(DMP_ENV_DESIGN))
if res.status<>DMP_OK then
  writeln('ERROR: ',res.lasterror)
end-if
```

getdmpcompctx

Purpose

Gets the values from the context structure of the current DMP component.

Synopsis

```
procedure getdmpcompctx(ctx: array(set of string) of text)
procedure getdmpcompctx(env: string, ctx: array(set of string) of text)
```

Arguments

`ctx` An array into which values from the context structure will be written. Existing array elements will not be modified, except when they have the same indexes as the context values.

`env` The DMP lifecycle environment for which to request context information - one of ”design”, ”staging” and ”production”. If not specified, the environment of the component instance executing the Mosel code will be used. Has no effect when the model is not running within DMP.

Example

This example demonstrates reading the `dmp.tenantId` value from the component context.

```
declarations
  names: set of string
  context: array(names) of text
end-declarations
getdmpcompctx(context)
if 'dmp.tenantId' in names then
  writeln('Executing in DMP tenant ', context('dmp.tenantId'))
else
  writeln('DMP tenant not detected')
end-if
```

Further information

1. This procedure allows easy access to information a DMP component or other application environment publishes about where it is executing. You should consult the component documentation for the expected context values and their meanings, but in Insight 5/Executor 4, they may include:
   - `insightUrl`
   - `traceId`
   - `parentSpanId`
   - `spanId`
   - `sampleId`
   - `dmp.managerUrl`
   - `dmp.environment`
   - `dmp.tenantId`
   - `dmp.solutionId`
   - `dmp.componentId`
   - `dmp.componentInstanceId`
   - `dmp.solutionToken`
   - `dmp.solutionTokenEnvironment`
   - `dmp.solutionTokenExpiryTime`
2. Xpress Insight 5 apps should access the context structure by calling the `insightgetcontext` function of the *mminsight*, which returns the same values in an easy-to-use record.
3. If called from an Xpress Insight 5 app running outside DMP, this procedure will return only those context values relevent outside DMP.
4. If called from Xpress Workbench or Xpress Executor version 3, this procedure will fail with an IO error.
5. Context fields without values will be omitted from the array.

getdmpcompid

Purpose

Gets the ID of the DMP component executing this model.

Synopsis

```
function getdmpcompid:text
```

Return value

The component ID.

Further information

1. This function returns the component ID (the same for design, staging and production lifecycle environments) and not the component instance ID (which is different for each environment).
2. When not running within DMP, this function will fail with a runtime error if called from an Insight 5 app. Use `isdmp` to check if the model is running from a DMP component.

getdmpcompurl

Purpose

Gets the URL of the DMP component executing this model.

Synopsis

```
function getdmpcompurl:text
```

Return value

The component URL.

Further information

The value returned may be an internal address that can only be accessed from within DMP.

getdmplifecycleenv

Purpose

Gets the DMP lifecycle environment of the component instance executing this model, or converts an environment name into one of the constant values.

Synopsis

```
function getdmplifecycleenv:string
function getdmplifecycleenv(env:text):string
```

Argument

`env` The environment name to convert.

Return value

One of the constant values `DMP_ENV_DESIGN`, `DMP_ENV_STAGING` and `DMP_ENV_PRODUCTION`, or an empty string if the value of the `env` parameter was unrecognzed.

Further information

1. When called with no argument, returns the current DMP lifecycle environment.
2. When called with a `text` parameter, will attempt to convert that parameter into one of the constant strings the `dmp` library uses to represent environments. This can be useful when parsing environment strings returned by DMP webservice APIs.
3. If the supplied `text` parameter is not recognized, or an empty string, this function will return an empty string.
4. When called without an argument when not running within DMP, this function will fail with a runtime error if it was called from an Insight 5 app. Use `isdmp` to check if the model is running in a DMP component.

getdmpsoldb

Purpose

Get information about the solution database for the current DMP solution.

Synopsis

```
procedure getdmpsoldb(params:array(set of string) of text)
```

Argument

`params` Array into which will be written various parameters about the solution database. Should be empty when passed in (any existing content will be removed). You can expect at least the below-listed index entries will be written to the array, but other entries may also be present (for example, recommended ODBC connection parameters).

- `drivername` the name of the ODBC driver that can be used to connect to the solution database.
- `server` the hostname of the solution database server.
- `port` the port on which to connect to the solution database server. This entry may be missing if the default port for the driver should be used.
- `username` the username to use for accessing to the solution database.
- `password` the password to use for accessing to the solution database.
- `database` the name of the solution database.

Further information

1. When you will be connecting to the database using `mmodbc`, it is easiest to call `getdmpsoldbconnstr` to fetch the ODBC connection string directly.
2. This procedure can only be used when the model is run within Insight 5 or Executor 4.
3. When not running in an Insight 5 or Executor 4 component in DMP, this procedure will fail with a runtime error.

Related topics

`getdmpsoldbconnstr`

getdmpsoldbconnstr

Purpose

Get connection string for the solution database for the current DMP solution.

Synopsis

```
function getdmpsoldbconnstr:text
```

Return value

An ODBC connection string that can be used to connect to the solution database.

Example

```
SQLconnect(getdmpsoldbconnstr)
if not getparam('SQLsuccess') then
  writeln('ERROR connecting to solution database')
  exit(1)
end-if
```

Further information

1. This function can only be used when the model is run within Insight 5 or Executor 4.
2. When not running in an Insight 5 or Executor 4 component in DMP, this function will fail with a runtime error.

Related topics

`getdmpsoldb`

getdmpsolid

Purpose

Gets the ID of the DMP solution to which the component executing this model belongs.

Synopsis

```
function getdmpsolid:text
```

Return value

The solution ID.

Further information

When not running within DMP, this function will fail with a runtime error if called from an Insight 5 app. Use `isdmp` to check if the model is running from a DMP component.

getinstance

Purpose

Gets information about an instance of a DMP component in a given lifecycle stage

Synopsis

```
function getinstance(comp:dmpcomponent,env:string):dmpcompinst
```

Arguments

`comp` A `dmpcomponent` value.

`env` The lifecycle stage of the instance to look for

`DMP_ENV_DESIGN`	Design or Root
`DMP_ENV_STAGING`	Staging
`DMP_ENV_PRODUCTION`	Production

Return value

A `dmpcompinst` populated with the information for the instance of requested lifecycle stage, or an unpopulated `dmpcompinst` if there is no instance for that stage.

isdmp

Purpose

Checks if the model is running within a DMP component that supports the `dmp` Mosel module.

Synopsis

```
function isdmp:boolean
```

Return value

`true` if running within DMP, `false` if not.

Further information

If this returns false, then any other functions you call from this module will generate errors.

seterror

Purpose

Sets the error status of the `dmpresource`, affecting the values of the `status` and `lasterror` attributes.

Synopsis

```
procedure seterror(res:dmpresource, status:integer, message:text)
```

Arguments

`res`	The `dmpresource`.	
`status`	The error status, one of the following constants	
	`DMP_ACCESS_DENIED`	Resource found but you do not have access to it
	`DMP_NOT_FOUND`	Resource not found
	`DMP_IO_ERROR`	Internal error reading or writing data
	`DMP_PARSE_ERROR`	Internal error parsing data
	`DMP_AUTH_ERROR`	Error fetching authorization credentials from function
`message`	The error message to save into the `lasterror` attribute.	

Further information

It is not typically necessary to call this procedure, as the error state of the `dmpresource` is automatically set on error cases.

CHAPTER 6

Parameters

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

dmp_verbose

Description When set to `true`, the module will produce additional logging output to the model's error stream.

Type Boolean, read/write

Default value `false`

Note This can be useful for diagnosing errors.

dmp_max_retries

Description Set the maximum number of times a failed HTTP 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 `dmp_retry_error_codes` parameter.

dmp_retry_error_codes

Description When a `dmphttp` request returns an HTTP status code found in this comma-separated list, it will be retried, up to the number of times specified by `dmp_max_retries`.

Type String, read/write

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

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