Reference Manual for VisualDOC MATLAB API

INTRODUCTION

Before reading this document, you should be familiar with VisualDOC and VisualDOC database concepts. To become familiar with these, please review the VisualDOC Getting Started document and VisualDOC Online Help.

VisualDOC API functions will allow you to embed all the capabilities of VisualDOC into your own program. These capabilities include Direct Gradient-based Optimization (DGO), Response Surface Approximate Optimization (RSA), and Design Of Experiments (DOE).

NOTE: List of all currently available function calls in alphabetical and functional order is in the end of this manual

REQUIREMENTS FOR VISUALDOC API

To compile and run VisualDOC API you need to be sure that your PATH contains the directory with all VisualDOC DLLs. These include both VDOC_API.DLLs and regular VisualDOC DLLs. VDOC_API.DLLs give access to the VisualDOC API functions. Regular VisualDOC DLLs are used by VisualDOC API functions.

GENERAL SEQUENCE OF CALLS IN VISUALDOC API

The use of VisualDOC API should start by opening a VisualDOC database. The database may be an existing database or a new one. After opening a database, you may perform general "get" and "put" operations with the database. The majority of the VisualDOC API consists of these get/put functions.

To define design problems, you will provide your users with the ability to define "inputs" and "responses". You may supply these inputs and responses to any of the design modules in VisualDOC. For optimization problems (i.e., minimization or maximization using DGO or RSA), your users must specify a design objective. This objective may be one or more of the defined inputs and responses. Your users may provide limits to some of the responses. These will then become constraints when you direct the VisualDOC database to construct a "design task". For design of experiments, your users must define side constraints on the inputs, which the DOE design module uses to place design points throughout the design space (depending on the chosen design). Your users can also specify that the DOE design module creates approximations to the responses. Most of your function calls to define design problems work with the "Interface" portion of the VisualDOC database.

Once your users have defined a design problem, you need to call the function that transforms the Interface objects into Design Task objects before you run the optimization from within your code. This function call checks all data for consistency and returns error/warning codes if problems were encountered that would prevent the design task from running.

There are distinct function calls for running Direct Gradient-based Optimization, Response Surface Approximation optimization, and Design of Experiments.

Once you have instructed the VisualDOC design task to start, you can periodically check the database for the run status and process design data in real time. After VisualDOC is finished, you can retrieve design results from the results objects of the database for further processing in your application.

When you are finished making VisualDOC API calls you should close the database.

NOTE: When accessing data from the database, the primary lookup key is a database ID for each object in the database. The database will return the database ID of an object when you call API functions that create or modify objects in the database.

NAMING CONVENTION

In this documents a certain naming convention was used to help you distinguish between various types of arguments in the function calls. All the argument names are prepend with one letter designating the type of the argument. The meaning of the prepending letters is the following:

m - two-dimensional array
v - one-dimensional row-vector
'…' - string

Here are examples of using this naming convention.

mA - Two-dimensional array "A"
vA - One-dimensional row-vector "A"
'A' - String "A"

All the arguments passed into and returned from the API function calls are assumed to be of the type "double", except for the case of strings.

GENERAL PURPOSE CALL

Error = VDOC_SetPath(*'Directory_name'*)

This function adds the specified directory in front of the users's PATH. So that MATLAB will be able to search for files at first in the specified directory and then in the rest of the path.

The function may take 1 parameter.

Input Argument	Description
Directory_name	String - Name of the directory
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2

GENERAL DATABASE ACCESS CALLS

Error = VDOC_OpenDatabase(*'Database_name', TimeOut, Queue*)

This function opens a database for processing. You must call this function before making any other database API calls. This function either opens an existing database or creates a new database.

The function may take 1, 2, or 3 parameters.

Input Argument	Description
Database_name	String - Name of the database
Queue	Key defining whether to queue the request for VisualDOC license or not when there are no free license available Possible values are: 0 - quit when license is not found 1 - Queue until a valid license is found If all licenses are hanged then the value of "1" will result in queuing for indefinite time. It is possible to free hanged licenses using license manager.

	[0]
TimeOut	Number of seconds until a license shall timeout. When this occurs the license server automatically frees the license. To prevent timeout, applications can set this number to 0. In this case, however, if the application crashes, the license will be hanged. If this number is too small, the license will be freed before the application is completed, and, therefore this will prevent application from completion. If the timeout value is set to less than 3600 (1 hour) it will be automatically set to 3600. Exception is the value of 0, which indicates no timeout. [3600]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2

Error = VDOC_CloseDatabase()

This function closes the currently open database. You must call this function after you are finished making other API calls. You may not call other API functions after this call, unless you call **VDOC_OpenDatabase**.

Return Parameter	Description
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2

Error = VDOC_IsOpen()

This function checks if a database is open.

Return Parameter	Description
Error	Return code. Possible values are: DB_SUCCESS=1

	DB_FAIL=-2

'ErrorMessage'= VDOC_GetDBError()

This function returns the message corresponding to the last error occurred in the database. This function may be called to get details of an error after any unsuccessful API call.

Return Parameter	Description
ErrorMessage	String with the details of the last error occurred in the database.

CALLS TO ACCESS INPUT OBJECTS

[*Error, vDBIDInput*] = VDOC_PutInputAll(*vDBIDInput, vX0, vLB, vUB*)

This function puts initial values, upper, and lower bounds of all input objects into the open database. 1, 2, 3, or 4 input arguments may be specified in the argument list.

This function operates based on the contents of the *vDBIDInput* vector. For each element of this vector that is 0, the database will create a new input object with default values for all attributes except initial values and bounds, which the database sets to the supplied values. For each element of this vector that has a non-zero value, the database will edit the input object that matches that database ID. The database will only modify the initial value and bounds. If a non-zero value does not match any database ID of existing input objects, the error code will be returned.

On return, this function provides current values of the database IDs for all input objects. You will use these database IDs whenever you access the attributes of the corresponding input object.

The transfer indices for input objects are set according to the order in which the input objects appear in the input arguments to this functions call.

The names for the input objects are set to "input_#", where # - is the transfer index of the input object.

The input objects present in the database, but whose database IDs are not specified in *vDBIDInput* will be removed from the database.

Input Arguments	Description
vDBIDInput	Vector of the input objects' database IDs
vX0	Vector of initial values for all input objects. This vector is not required for DOE

	[0]
vLB	Vector of the lower bounds on input objects. This vector is not required to perform optimization [-1.e30]
vUB	Vector of the upper bounds on input objects. This vector is not required to perform optimization [1.e30]
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2
vDBIDInput	Vector of all input objects' database IDs

[*Error, vX0, vLB, vUB, vDBIDInput, vType, vIsObjective*] = VDOC_GetInputAll()

This function gets the current values of all input objects from the database.

Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2
vX0	Vector of current values for all input objects.
vLB	Vector of the lower bounds on input objects.
vUB	Vector of the upper bounds on input objects.
vDBIDInput	Vector of the input objects database IDs
vType	Vector of types of input objects. Possible values are: independent=0, link=1, synthetic=2, constant=3, discrete=4, integer=5
vIsObjective	Vector with flags defining whether the corresponding input object is also an objective. Possible values: TRUE=1, FALSE=0

[*Error, DBIDInput*] = VDOC_PutInputObj(*DBIDInput, Goal, Target, Weight, Worst*)

Function to specify that the given input object (based on the database ID) is an objective. 1, 2, 3, 4, or 5 arguments may be specified.

This function operates based on the contents of the input argument *DBIDInput*. If *DBIDInput* has a non-zero value, the database will modify the attributes specified in the API call for the input object that matches that database ID. If *DBIDInput* has value of 0, a new input object will be created with all the attributes set to their default values, except for the ones specified in the API call. If a non-zero value of *DBIDInput* does not match any database ID of existing input objects, the error code will be returned.

If *DBIDInput* has value of 0, all the input objects present in the database, except for the one created during this functional call will be removed from the database.

Input Arguments	**Description**
DBIDInput	Database ID of the input object, which is an objective. If the Database ID is zero, a new input object will be put into the database. Its initial value, upper and lower bounds will be set to default values.
Goal	Defines the optimization goal for this component of the objective. Possible values are: minimize=0, maximize=1, target=2 [minimize=0]
Target	Target value for the objective [0]
Weight	Weight factor for the objective in multiobjective optimization. [1]
Worst	Worst value for the objective [0]
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2

DBIDInput	Database ID of the input object, which is also an objective. Is equal to the input value of *DBIDInput*, except when input value of *DBIDInput* is equal to zero.

[*Error, Goal, Target, Weight, Worst*] = VDOC_GetInputObj(*DBIDInput*)

This function gets the objective attributes of the specified input object.

This function operates based on the contents of the input argument *DBIDInput*. If *DBIDInput* has a non-zero value, the database will extract the objective attributes of the input object that matches that database ID. If *DBIDInput* has a value of 0, an error code will be returned. If a non-zero value of *DBIDInput* does not match any database ID of existing input objects, the error code will be returned.

Input Arguments	**Description**
DBIDInput	Database ID of the input object. If equal to zero, an error code will be returned.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
Goal	Defines the optimization goal for this component of the objective. Possible values are: minimize=0, maximize=1, target=2
Target	Target value for the objective
Weight	Weight factor for the objective in multiobjective optimization.
Worst	Worst value for the objective

Error = VDOC_ChangeInputType(*DBIDInput, Type*)

Function to change type of the given input object (based on the database ID of the input object). 2 arguments must be specified. The type may be changed to “independent”, “discrete”, or “integer”. (See description of the arguments). If the type of the input object is desired to be discrete, the corresponding discrete set should be previously defined using **VDOC_PutDiscreteSet**. The parameter returned from **VDOC_PutDiscreteSet** should be used as a second argument to **VDOC_ChangeInputType**.

This function operates based on the contents of the input argument *DBIDInput*. *DBIDInput* has to have a positive value. It could be one of the values returned by **VDOC_PutInputAll**.

Input Arguments	Description
DBIDInput	Database ID of the input object which type will be changed.
Type	Defines the type the specified input object should take Possible values are: =0 – continuous <0 – integer >0 – discrete (the value represents the integer set that should be used fore this design variable).
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

CALLS TO ACCESS RESPONSE OBJECTS

[*Error, vDBIDResp*] = VDOC_PutRespAll(*vDBIDResp, vLB, vUB, vLBSc, vUBSc*)

API call to put bounds and scale factors for all response objects in the open database. Some of these responses may be constraints and some – objectives. 1, 2, 3, 4, or 5 arguments may be specified.

This function operates based on the contents of the *vDBIDResp* vector. For each element of this vector that is 0, the database will create a new response object with default values for all attributes except the ones specified in the API call. For each element of this vector that has a non-zero value, the database will replace the response object that matches that database ID. The database will only modify the specified values. If a non-zero value does not match any database ID of existing response objects, the error code will be returned.

On return, this function provides current values of the database IDs for all response objects. You will use these database IDs whenever you access the attributes of the corresponding response object.

The transfer indices for response objects are set according to the order in which the response objects appear in the input arguments to this functions call.

The names for the response objects are set to "response_#", where # - is the transfer index of the response object.

The response objects present in the database, but whose database IDs are not specified in *vDBIDResp* will be removed from the database.

On return, this function provides current values of the database IDs for provided response objects. You will use these database IDs whenever you access the attributes of the corresponding response object.

Input Arguments	Description
vDBIDResp	Vector of the response objects' database IDs
vLB	Vector of lower bounds for all response objects [-1.e30]
vUB	Vector of lower bounds for all response objects [1.e30]
vLBSc	Vector of scale factors for the lower bounds for all response objects [1]
vUBSc	Vector of scale factors for the upper bounds for all response objects [1]
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2
vDBIDResp	Vector of the responses database IDs

[*Error, vDBIDResp, vLB, vUB, vLBSc, vUBSc, vType, vIsObjective, vIsConstraint*] = VDOC_GetRespAll()

API call to get bounds and scale factors for all response objects from the open database.

Return Parameters	Description
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2
vDBIDResp	Vector of the response objects' database IDs

vLB	Vector of lower bounds for all response objects
vUB	Vector of lower bounds for all response objects
vLBSc	Vector of scale factors for the lower bounds for all response objects
vUBSc	Vector of scale factors for the upper bounds for all response objects
vType	Vector of types of response objects. Possible values are: independent=0, link=1, synthetic=2, constant=3, discrete=4, integer=5
vIsObjective	Vector with flags defining whether the corresponding response object is an objective. Possible values: TRUE=1, FALSE=0
vIsConstraint	Vector with flags defining whether the corresponding response object is a constraint. Possible values: TRUE=1, FALSE=0

[*Error, DBIDResp*] = VDOC_PutRespObj(*DBIDResp, Goal, Target, Weight, Worst*)

API call to specify that ONE of the response objects is objective. 1, 2, 3, 4, or 5 input arguments may be specified.

This function operates based on the contents of the input argument *DBIDResp*. If *DBIDresp* has a non-zero value, the database will modify the attributes specified in the API call for the response object that matches that database ID. If *DBIDResp* has a value of 0, the new response object will be created with all the attributes set to their default values, except for the ones specified in the API call. If a non-zero value does not match any database ID of existing response objects, the error code will be returned.

DBIDResp for the input argument may be obtained as one of the elements of the returned vector *vDBIDResp* from the call to **VDOC_PutRespAll.**

On return, this function provides current value of the database ID for the provided response object.

If *DBIDResp* has value of 0, all the response objects present in the database, except for the one created during this functional call will be removed from the database.

Input Arguments	Description
DBIDResp	Database ID of the response object, which is also objective. One of the database IDs returned by

	API call **VDOC_PutRespAll**. If the database ID is zero, the new response object will be created in the database. Its upper and lower bounds, as well as scale factors will be set to default values. If a non-zero value does not match any database ID of existing response objects, the error code will be returned.
Goal	Defines the optimization goal for this component of the objective. Possible values are: minimize=0, maximize=1, target=2 [minimize=0]
Target	Target value for the objective [0]
Weight	Weight factor for the objective in multiobjective optimization. [1]
Worst	Worst value for the objective [0]
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2
DBIDResp	Database ID of the response object, which is also an objective. Is equal to the input value of *DBIDResp*, except when input value of *DBIDResp* is equal to zero.

[*Error, Goal, Target, Weight, Worst*] = VDOC_GetRespObj(*DBIDResp*)

API call to get the objective attributes of the specified response object.

This function operates based on the input value of the database ID *DBIDResp*. If *DBIDResp* has a non-zero value, the database will extract the attributes related to objective in response object that matches that database ID. If *DBIDresp* has a value of 0, the error code will be returned. If a non-zero value does not match any database ID of existing response objects, the error code will be returned.

DBIDresp may be obtained as one of the elements of the returned vector *vDBIDResp* from the call to **VDOC_PutRespAll.**

Input Argument	Description
DBIDResp	Database ID of the response object. One of the database IDs returned by API call **VDOC_PutRespAll**. If the database ID is zero, the error code will be returned. If a non-zero value does not match any database ID of existing response objects, the error code will be returned.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2
Goal	Defines the optimization goal for this component of the objective. Possible values are: minimize=0, maximize=1, target=2
Target	Target value for the objective
Weight	Weight factor for the objective in multiobjective optimization.
Worst	Worst value for the objective

Error **= VDOC_ChangeRespType(** ***DBIDResp, Type*** **)**

Function to change type of the given response object (based on the database ID of the response object). 2 arguments must be specified. The type may be changed to “independent” or “passfail” (See description of the arguments).

This function operates based on the contents of the input argument DBIDResp. DBIDResp has to have a positive value. It could be one of the values returned by **VDOC_PutRespAll**.

Input Arguments	Description
DBIDResp	Database ID of the response object which type will be changed.
Type	Defines the type the specified response object should take Two possible values are subset of enumerator

	"Type" defined in "defs.h": =0 – continuous =3 - passfail
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

CALLS TO ACCESS USER DEFINED POINTS (G-POINTS)

Error = VDOC_PutGPoints(*mDVs, vToAnalyze, vToUse, mResps*)

API call to specify user-defined G-points. These points may be used for DOE run, or as an initial DOE for Response Surface Approximate optimization. 1, 2, 3, or 4 arguments could be supplied for the API call.

This call creates a new G-Points object in the open database.

NOTE: One should distinguish between G-points and D-points. D-points are the points obtained as a result of the run of the task.

Input Arguments	**Description**
mDVs	Two-dimensional array with the values of the design variables for each G-Point. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL design variables.
vToAnalyze	Array with the flags for each point. Each flag indicates whether the corresponding point should be analyzed or not. Possible values for each element: TRUE=1, FALSE=0 [TRUE]
vToUse	Array with the flags for each point. Each flag indicates whether the corresponding point should be used in further calculations or not. Possible values for each element: TRUE=1,

	FALSE=0 [TRUE]
mResps	Two-dimensional array with the values of the responses for each G-Point. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL responses. The values of the responses should be evaluated at the points with the values of the design variables from matrix *mDVs*. The number of rows should be equal to the number of rows in *mDVs*.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

[*Error, mDVs, vToAnalyze, vToUse, mResps*] = VDOC_GetGPoints()

API call to extract user-defined G-points. These points may be used for DOE run, or as an initial DOE for Response Surface Approximate optimization.

This call accesses a current G-Points object in the open database.

Retrun Arguments	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
mDVs	Two-dimensional array with the values of the design variables for each G-Point. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL design variables.
vToAnalyze	Array with the flags for each point. Each flag indicates whether the corresponding point should be analyzed or not. Possible values for each element: TRUE=1, FALSE=0

	[TRUE]
vToUse	Array with the flags for each point. Each flag indicates whether the corresponding point should be used in further calculations or not. Possible values for each element: TRUE=1, FALSE=0 [TRUE]
mResps	Two-dimensional array with the values of the responses for each G-Point. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL responses. The values of the responses are the ones evaluated at the points with the values of the design variables from matrix *mDVs*.

CALLS TO ACCESS AUXILIARY OBJECTS

[*Error, DBIDSet*] = VDOC_PutDiscreteSet(*DBIDSet, vSetValues, `SetName'*)

API call to put the set of discrete values in the open database.

This function operates based on the contents of the input argument *DBIDSet*. If *DBIDSet* has a non-zero value, the database will modify the attributes specified in the API call for the Auxiliary object (discrete set) that matches that database ID. If *DBIDSet* has a value of 0, the new Auxiliary object (discrete set) will be created with all the attributes set to their default values, except for the ones specified in the API call. If a non-zero value does not match any database ID of existing the Auxiliary objects (discrete sets), the error code will be returned.

2 or 3 parameters may be specified.

The current database ID of the discrete set will be returned. You will use these ID when you will need to refer the design variable to take the values from this set.

Input Arguments	Description
DBIDSet	database ID of the set.
vSetValues	Vector of the discrete values for the set
SetName	String defining the name of the set of values

	[']
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
DBIDSet	database ID of the set.

[*Error, vSetValues, `SetName'*] = VDOC_GetDiscreteSet(*DBIDSet*)

API call to get the set of discrete values from the open database.

This function operates based on the contents of the input argument *DBIDSet*. If *DBIDSet* has to have a positive value previously returned by **VDOC_PutDiscreteSet.** The API call will extract the Auxiliary object (discrete set) that matches that database ID. The parameters of this object will be passed out to the user as return parameters. If a non-zero value of *DBIDSet* does not match any database ID of existing the Auxiliary objects (discrete sets), the error code will be returned.

Input Arguments	**Description**
DBIDSet	database ID of the set.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
vSetValues	Vector of the discrete values for the set
SetName	String defining the name of the set of values [']

CALLS TO ACCESS DESIGN CONTROL OBJECT. GENERAL DESIGN CONTROL

Error = VDOC_PutTaskType(*TaskType, 'TaskTitle'*)

API call to specify the general design control parameters: task type and task title. 1 or 2, arguments may be specified.

This call modifies the specified attributes of the current design control object.

Input Arguments	Description
TaskType	Type of the task to be performed. Possible values: DGO=0, RSA=1, DOE=2
TaskTitle	String defining a short task description [']
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

[*Error, TaskType, 'TaskTitle'*] = VDOC_GetTaskType()

API call to get the general design control parameters: task type and task title.

This call extracts the general attributes of the current design control object.

Return Parameters	Description
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2
TaskType	Type of the task to be performed. Possible values: DGO=0, RSA=1, DOE=2 [DGO=0]
TaskTitle	String defining the short task description [']

CALLS TO ACCESS DESIGN CONTROL OBJECT. DGO CONTROL

Error = VDOC_PutDGOControlGeneral(*ConstMethod, UnconstMethod, NItersMax, NitersConv, ToMinimize*)

API call to specify the general design control parameters related to DGO. 1, 2, 3, 4, or 5 arguments may be specified.

This call modifies the specified attributes of the current design control object.

Input Arguments	Description
ConstMethod	Method of constrained optimization. Possible values: SQP=0, SLP=1, MMFD=2 The values are defined in enumerator "Method" in "defs.h"
UnconstMethod	Method of unconstrained optimization. Possible values: BFGS=0, FR=1 The values are defined in enumerator "Method" in "defs.h"
NItersMax	Maximum number of iterations to perform in DGO. Should be a positive number. [100]
NItersConv	Number of iterations where convergence criteria should be satisfied for the optimization to stop. Should be a positive number. [2]
ToMinimize	Flag specifying whether the combined objective function in DGO will be minimized (or maximized) Possible values are: TRUE=1, FALSE=0 (defined in "defs.h") [TRUE]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1

	DB_FAIL=-2, DB_BAD_ARGS=-10

Error = VDOC_PutDGOControlConstr(*ActConstToler, ViolConstToler*)

API call to specify DGO control parameters for active and violated constraints. 1 or 2 arguments may be specified.

This call modifies the specified attributes of the current design control object.

Input Arguments	Description
ActConstToler	A constraint is considered active if its value is more positive than *ActConstToler*. Should be a negative number. [-0.03]
ViolConstToler	A constraint is considered violated if its value is more negative than *ViolConstToler*. Should be a positive number. [0.003]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

Error = VDOC_PutDGOControlConv(*SoftRel, SoftAbs, HardRel, HardAbs*)

API call to specify control parameters for convergence of direct gradient based optimisation (DGO). 1, 2, 3, or 4 arguments may be specified. Convergence is determined comparing corresponding values at the last *NitersConv* consecutive iterations (defined in **VDOC_PutDGOControlGeneral**).

This call modifies the specified attributes of the current design control object.

Input Arguments	Description
SoftRel	A maximum relative change in design variables during the last *NitersConv* consecutive iterations to achieve relative soft convergence. Should be a positive number. [0.001]

SoftAbs	A maximum absolute change in design variables during the last *NitersConv* iterations to achieve absolute soft convergence. Should be a positive number. [0.0001]
HardRel	A maximum relative change in objective function during the last *NitersConv* consecutive iterations to achieve relative hard convergence. Should be a positive number. [0.001]
HardAbs	A maximum absolute change in objective function during the last *NitersConv* iterations to achieve absolute hard convergence. Should be a positive number. [the value is determined after the first evaluation of objective function: 0.0001*ABS(OBJ)]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

Error **= VDOC_PutDGOControlFD(** ***GradCalcMethod, RelFDStep, AbsFDStep*** **)**

API call to specify parameters for calculating gradients in DGO. 1, 2, or 3 arguments may be specified.
This call modifies the specified attributes of the current design control object.

Input Arguments	**Description**
GradCalcMethod	Method of calculating gradients. Possible values are: 0-Forward differences; 1-Central differences; 2-User supplied gradients. (Enumerator Grad in “defs.h”) [0]
RelFDStep	Relative finite difference step when calculating gradients.

	[0.001]
AbsFDStep	Minimum absolute finite difference step when calculating gradients. [0.0001]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

[*Error, vIntDGOControl, vDblDGOControl*] = VDOC_GetDGOControl()

API call to get the current control parameters that govern DGO from a database.

Description of each argument see in the tables below.

This call extracts the attributes of the current design control object.

Return Arguments	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
vIntDGOControl	Vector with the current integer DGO control parameters. At least 6 elements. See description in the table below.
vDblDGOControl	Array with the current double DGO control parameters. At least 8 elements. See description in the table below.

Description of the elements of the array with INTEGER DGO control parameters.

Name	**1-based Index**	**Description**
MethodCons	1	Method of constrained optimization to be used. Possible values are: SQP=0, SLP=1, MMFD=2 (defined in enumerator “Method” in the files “defs.h”)
MethodUncons	2	Method of unconstrained optimization to be used.

Name	1-based Index	Description
		Possible values are: BFGS=0, FR=1 (defined in enumerator "Method" in the files "defs.h")
NumIterConv	3	Number of iterations for convergance
NumIterMax	4	Maximum number of iterations to be performed in optimization.
GradCalcMethod	5	Method of calculating gradients. Possible values are: 0-Forward differences; 1-Central differences; 2-User supplied gradients. (Enumerator Grad in "defs.h") [0]
ToMinimize	6	Flag specifying whether the combined objective function in DGO will be minimized (or maximized) Possible values are: TRUE=1, FALSE=0 (defined in "defs.h") Default is TRUE.

Description of the elements of the array with DOUBLE DGO control parameters.

Name	1-based Index	Description
CT	1	A constraint is considered active if its value is > *CT*
CTViolated	2	Violated constraint tolerance. A constraint is considered violated if its value is > *CTViolated*
HardConvRel	3	Relative hard convergence for optimization (Relative change in objective function).
HardConvAbs	4	Absolute hard convergence for optimization (Absolute change in objective function). Unless specifically set, this value is defined after the first objective function is evaluated in the optimization process. If this parameter was not specifically set 0.0 will be returned as its value.
SoftConvRel	5	Relative soft convergence for optimization (Relative change in design variables).
SoftConvAbs	6	Absolute soft convergence for optimization (Absolute

Name	1-based Index	Description
		change in design variables).
FDStepRel	7	Relative finite difference step to calculate gradients.
FDStepAbsMin	8	Absolute minimum finite difference step to calculate gradients.

CALLS TO ACCESS DESIGN CONTROL OBJECT. RSA CONTROL

Error = VDOC_PutRSAControlIni(*InitialDOE*)

API call to specify the initial starting strategy for RSA (what DOE to create initially) and whether to minimize or maximize the global objective function.

This call modifies the specified attributes of the current Design Control object.

Input Arguments	Description
InitialDOE	DOE from which RSA will start Possible values for constrained optimization: Koshal=5, Simplex=9, Taylor=99, UserDOE=11 If the "UserDOE" is defined, then corresponding G-Points should be set before making the task. Default is Koshal=5. (defined in enumerator "DOEMethod" in the files "defs.h")
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

Error = VDOC_PutRSAControlGeneral(*NitersConv, NPointsMax, NPointsMin, ModelOrder, ToMinimize*)

API call to specify the general design control parameters related to RSA. 1, 2, 3, 4, or 5 arguments may be specified.

This call modifies the specified attributes of the current design control object.

Input Arguments	Description
NItersConv	Number of iterations where convergence criteria should be satisfied for the optimization to stop. Should be a positive number. [2]
NPointsMax	Maximum number of points to be analyzed during RSA. Should be >= 2.
NPointsMin	Minimum number of points to be analyzed during RSA. Should be <= *NpointsMax* and > 0.
ModelOrder	Maximum order of the polynomial model internally created during RSA. Possible values are: Linear = 0, LinearInter = 1, QuadNoInter = 2, FullQuad = 3 The values are taken from DOEModel enumerator in "defs.h" [FullQuad = 3]
ToMinimize	Flag specifying whether the combined objective function in RSA will be minimized (or maximized) Possible values are: TRUE=1, FALSE=0 (defined in "defs.h") [TRUE]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

Error **= VDOC_PutRSAControlConstr(** ***ActConstToler, ViolConstToler*****)**

API call to specify RSA control parameters for active and violated constraints. 1 or 2 arguments may be specified.

This call modifies corresponding attribute of the current design control object.

Input Arguments	Description

ActConstToler	A constraint is considered active if its value is more positive than "ActConstToler". Should be a negative number. [–0.03]
ViolConstToler	A constraint is considered violated if its value is more negative than "ViolConstToler". Should be a positive number. [0.003]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

Error **= VDOC_PutRSAControlConv(** ***SoftRel, SoftAbs, HardRel, HardAbs*** **)**

API call to specify control parameters for convergence of response surface approximate optimization (RSA). 1, 2, 3, or 4 arguments may be specified. Convergence is determined comparing corresponding values at the last *NitersConv* consecutive iterations (defined in **VDOC_PutRSAControlGeneral**).

This call modifies the specified attributes of the current design control object.

Input Arguments	**Description**
SoftRel	A maximum relative change in design variables during the last *NitersConv* consecutive iterations to achieve relative soft convergence. Should be a positive number. [0.001]
SoftAbs	A maximum absolute change in design variables during the last *NitersConv* iterations to achieve absolute soft convergence. Should be a positive number. [0.0001]
HardRel	A maximum relative change in objective function during the last *NitersConv* consecutive iterations to achieve relative hard convergence. Should be a positive number.

	[0.001]
HardAbs	A maximum absolute change in objective function during the last *NitersConv* iterations to achieve absolute hard convergence. Should be a positive number. [the value is determined after the first evaluation of objective function: 0.0001*ABS(OBJ)]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

Error **= VDOC_PutRSAControlMoveLim(** ***RelMLLin, RelMLMix, RelMLQuad, AbsMLLin AbsMLMix, AbsMLQuad*** **)**

API call to specify move limit parameters for RSA optimization. During RSA the approximation a polynomial model used inside is increased in complexity from iterations to iteration from linear to full quadratic. Different move limits for the design variables are used for each particular polynomial model. 1, 2, 3, 4, 5, or 6 arguments may be specified for this API call

This call modifies corresponding attribute of the current design control object.

Input Arguments	**Description**
RelMLLin	Relative move limits for the linear model. [0.2 of the current design variable values]
RelMLMix	Relative move limits for the model that is more than linear but not yet full quadratic [0.3 of the current design variable values]
RelMLQuad	Relative move limits for the full quadratic model. [0.4 of the current design variable values]
AbsMLLin	Absolute move limits for the linear model. [0.02]
AbsMLMix	Absolute move limits for the model that is more than linear but not yet full quadratic

	[0.03]
AbsMLQuad	Absolute move limits for the full quadratic model. [0.04]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

[*Error, vIntRSAControl, vDblRSAControl*] = VDOC_GetRSAControl()

API call to get the current control parameters that govern RSA from a database.

Description of each argument see in the tables below.

This call extracts the attributes of the current design control object.

Return Arguments	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
vIntRSAControl	Vector with the current integer RSA control parameters. At least 6 elements. See description in the table below.
vDblRSAControl	Array with the current double RSA control parameters. At least 12 elements. See description in the table below.

Description of the elements of the array with INTEGER RSA control parameters.

Name	**1-based Index**	**Description**
Design	1	DOE from which RSA wil start Possible values for constrained optimization: Koshal=5, Simplex=9, Taylor=99, UserDOE=11 Default is Koshal=5. (defined in enumerator "DOEMethod" in the files

Name	1-based Index	Description
		"defs.h")
nNItersConv	2	Number of iterations where convergence criteria should be satisfied for the optimization to stop. Should be a positive number. Default is 2.
NPointsMax	3	Maximum number of points to be analyzed during RSA. If this parameter was not specifically set 0 will be returned as its value.
NPointsMin	4	Minimum number of points to be analyzed during RSA. If this parameter was not specifically set 0 will be returned as its value.
ModelOrder	5	Maximum order of the polynomial model internally created during RSA. Possible values are: Linear = 0, LinearInter = 1, QuadNoInter = 2, FullQuad = 3 Default is FullQuad = 3. The values are taken from DOEModel enumerator in "defs.h".
ToMinimize	6	Flag specifying whether the combined objective function in RSA will be minimized (or maximized) Possible values are: TRUE=1, FALSE=0 (defined in "defs.h") Default is TRUE.

Description of the elements of the array with DOUBLE RSA control parameters.

Name	1-based Index	Description
CT	1	A constraint is considered active if its value is > *CT*
CTViolated	2	Violated constraint tolerance. A constraint is considered violated if its value is > *CTViolated*
HardConvRel	3	Relative hard convergence for optimization (Relative change in objective function).

Name	1-based Index	Description
HardConvAbs	4	Absolute hard convergence for optimization (Absolute change in objective function). Unless specifically set, this value is defined after the first objective function is evaluated in the optimization process. If this parameter was not specifically set 0.0 will be returned as its value.
SoftConvRel	5	Relative soft convergence for optimization (Relative change in design variables).
SoftConvAbs	6	Absolute soft convergence for optimization (Absolute change in design variables).
RelMLLin,	7	Relative move limits for the linear model. Default is 0.2 of the current design variable values.
dRelMLMix	8	Relative move limits for the model that is more than linear but not yet full quadratic Default is 0.3 of the current design variable values.
dRelMLQuad	9	Relative move limits for the full quadratic model. Default is 0.4 of the current design variable values.
AbsMLLin,	10	Absolute move limits for the linear model. Default is 0.02.
AbsMLMix	11	Absolute move limits for the model that is more than linear but not yet full quadratic Default is 0.03.
AbsMLQuad	12	Absolute move limits for the full quadratic model. Default is 0.04.

CALLS TO ACCESS DESIGN CONTROL OBJECT. DOE CONTROL

Error = VDOC_PutDOEControlGeneral(*DOEAction, ModelOrder*)

API call to specify the general design control parameters related to DOE. 1 or 2 arguments may be specified.

This call modifies the specified attributes of the current design control object.

Input Arguments	Description
DOEAction	Type of action to be performed by DOE. Possible values are: Points = 0, PointsResps = 1, PointsRespsModels = 2 The values are taken from DOETask enumerator in "defs.h". [Points = 0]
ModelOrder	Maximum order of the polynomial model to be created during DOE run. Possible values are: Linear = 0, LinearInter = 1, QuadNoInter = 2, FullQuad = 3, MixedFwdRegr = 4 The values are taken from DOEModel enumerator in "defs.h" [MixedFwdRegr = 4]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

Error **= VDOC_PutDOEControlDesign(** ***Design, DesignOption, NDoptPoints*** **)**

API call to specify the type of the design to be used when running DOE. 1, 2, or 3 arguments may be specified.

This call modifies the specified attributes of the current design control object.

Input Arguments	Description
Design	Type of the design to be used when running DOE. The values are taken from DOEMethod enumerator in "defs.h". They are also described in the table below. [Composite = 1]

DesignOption	Option for the particular design. Possible values are provided in the table below [0]
NDoptPoints	Number of D-optimal points to be selected. D-optimal points will selected from the original design. D-optimal points will be selected only if *Random* or *User-supplied* design are provided as the original design. [0]
Return Parameter	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10

Description of the design options for the individual designs.

Design	Options
Factorial = 0	0 – Full factorial design 1 – 1/2 Fractional factorial design (2**(NDV-1) points) 2 – 1/4 Fractional factorial design (2**(NDV-2) points) 3 – 1/8 Fractional factorial design (2**(NDV-3) points) etc.
Composite = 1	0 – Full factorial design is used for the fractional portion of the CCD 1 – 1/2 Fractional factorial design (2**(NDV-1) points) is used for the fractional portion of the SCD 2 – 1/4 Fractional factorial design (2**(NDV-2) points) is used for the fractional portion of the SCD 3 – 1/8 Fractional factorial design (2**(NDV-3) points) is used for the fractional portion of the SCD etc.
Box-Behnken = 2	1 - Perturb 1 factor at a time 2 - Perturb 2 factors at a time 3 - Perturb 3 factors at a time 12 - Perturb 1 & 2 factors at a time 13 - Perturb 1 & 3 factors at a time 23 - Perturb 2 & 3 factors at a time 123 - Perturb 1, 2, & 3 factors at a time [1]

Plackett-Burman = 3	The numerical value of the option corresponds to the number of points in the design. Possible values are: 12, 20, 24, 28, 36 [12]
Notz = 4	-1 – "Pure" Notz design: factorial portion is a full factorial design without the point where all the design variables are at their high levels. 0 – Full factorial design is used for the fractional portion of the design 1 – 1/2 Fractional factorial design (2**(NDV-1) points) is used for the fractional portion of the design 2 – 1/4 Fractional factorial design (2**(NDV-2) points) is used for the fractional portion of the design 3 – 1/8 Fractional factorial design (2**(NDV-3) points) is used for the fractional portion of the design etc. [-1]
Koshal = 5	10 – Design for the 1st order model without interactions 11 - Design for the 1st order model with interactions 20 - Design for the 2nd order model with only quadratic terms but without interactions 21 – Design for the full-quadratic model [10]
Rechtschaffner = 6	10 – Design for the 1st order model without interactions 11 - Design for the 1st order model with interactions 20 - Design for the 2nd order model with only quadratic terms but without interactions 21 – Design for the full-quadratic model [10]
Hybrid = 7	3101 – 3 variables, 10 points, design #1 3111 – 3 variables, 11 points, design #1 3112 – 3 variables, 11 points, design #2 4161 – 4 variables, 16 points, design #1 4162 – 4 variables, 16 points, design #2 4163 – 4 variables, 16 points, design #3 6281 – 6 variables, 28 points, design #1 6282 – 6 variables, 28 points, design #2 [3101]
Taguchi = 8	1 – Latin Square 1 2 - Latin Square 2 9 – L9 Taguchi orthogonal array 27 – L27 Taguchi orthogonal array

	81 – L81 Taguchi orthogonal array [1]
Simplex = 9	No options available. The number of point is always equal to the number of design variables + 1. [0]
Random = 10	The numerical value of the option corresponds to the number of points in the design. [0]
User-Supplied = 11	No options available. The points should be supplied as G-points [0]

[*Error, vDOEControl*] = VDOC_GetDOEControl()

API call to get the current control parameters that govern DOE from a database.

Description of each argument see in the tables below.

This call extracts the attributes of the current design control object.

Return Arguments	Description
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
vDOEControl	Vector with the current integer DOE control parameters. At least 5 elements. See description in the table below.

Description of the elements of the array with DOE control parameters.

Name	1-based Index	Description
DOEAction	1	Type of action to be performed by DOE. Possible values are: Points = 0, PointsResps = 1, PointsRespsModels = 2 The values are taken from DOETask enumerator in

Name	1-based Index	Description
		"defs.h". [0]
ModelOrder	2	Maximum order of the polynomial model to be created during DOE run. Possible values are: Linear = 0, LinearInter = 1, QuadNoInter = 2, FullQuad = 3, MixedFwdRegr = 4 The values are taken from DOEModel enumerator in "defs.h" [MixedFwdRegr = 4]
Design	3	Type of the design to be used when running DOE. The values are taken from DOEMethod enumerator in "defs.h". They are also described in the table below. [Composite = 1]
DesignOption	4	Option for the particular design. Possible values are described in the table for the function *VDOC_PutDOEControlDesign* [0]
NDoptPoints	5	Number of D-optimal points to be selected. D-optimal points will selected from the original design. D-optimal points will be selected only if *Random* or *User-supplied* design are provided as the original design. [0]

CALL TO MAKE A TASK

TaskNumber = VDOC_MakeTask()

Function to create a task using the information that is already in the database in the Interface objects (Input, Response, Auxiliary, Design Control objects).

You will use the returned task number whenever you access the results of the corresponding task.

Return Parameter	Description

TaskNumber	The database number of the task that was created. The task number should be always positive. If there was an error during task creation DB_FAIL=-2 will be returned.

CALLS TO RUN FUNCTIONAL MODULES

[*Error, StatusCode, mDVs, mResps, vIndActiveResp*] = VDOC_RunDGO(*TaskNumber, StatusCode, mDVs, mResps, vIndActiveResp*)

Function to run the direct gradient-based optimization (DGO) task with the specified ID. You may need to call this function in a loop, until the returned value of the status code will indicate that the process is completed (F_DONE). After each call to this function you may be asked to evaluate the responses at one or several points with the provided values of design variables (F_RESP, F_GRAD).

When this function is called for the first time StatusCode should be set to F_INIT=0. Later on VisualDOC will be updating this StatusCode and no more changes should be made to its value.

When this function is called NOT for the first time user should provide the values of ALL the design variables that he got as return arguments from the previous call to **VDOC_RunDGO**. The order of the design variables should correspond to the transfer indices obtained from **VDOC_PutInputAll** or **VDOC_GetInputAll**. Thus, only 2 arguments could be provided during the first functional call instead of 4, if desired. If 4 arguments will be provided, the values of the design variables and responses will be ignored.

When responses are supplied to this functional call, ALL the responses should be present. The order of the responses should correspond to the transfer indices obtained from **VDOC_PutRespAll** or **VDOC_GetRespAll**.

Vector of indices of independent response when used as an input argument should be just a copy of the same vector supplied as return arguments from the previous call to **VDOC_RunDGO**. This argument could be omitted in the initial call to **VDOC_RunDGO**.

Only DGO type tasks will be run using this call. An attempt to run DOE or RSA task will result in error.

Input Argument	Description
TaskNumber	The DGO task number to run. One of the task numbers returned by **VDOC_MakeTask**. The task number should be always positive.
StatusCode	Status code of the run. Specifies what step of the run is being currently executed Possible values are: F_INIT=0, F_DONE=1, F_RESP=2, F_GRAD=3, F_GRAD_INIT=4,

	F_USER_GRAD=5 Complete description of all possible codes is given in the file "defs.h" in enumeration "Design_Step"
mDVs	Two-dimensional array with the values of the design variables. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL design variables. During the very first call to this function the values in the array will be ignored.
mResps	Two-dimensional array with the values of the responses. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL responses. The values of the responses should be evaluated at the points with the values of the design variables from array *mDVs*. During the very first call to this function the values in the array will be ignored.
vIndActiveResp	Vector of indices of independent response. Is used only for the case of user-provided gradients (not implemented in the API yet).
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10, DB_EXCEPTION=-91
StatusCode	Status code of the run. Specifies what step of the run is being currently executed Possible return values are: F_DONE=1, F_RESP=2, F_GRAD=3, F_GRAD_INIT=4, F_USER_GRAD=5 Complete description of all possible codes is given in the file "defs.h" in enumeration "Design_Step"
mDVs	Two-dimensional array with the values of the design variables. The number of rows should

	equal to the number of points, the number of columns should equal to the number of ALL design variables. All responses should be evaluated at the provided points.
mResps	Two-dimensional array with the values of the responses. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL responses.
vIndActiveResp	Vector of indices of independent response. Is used only for the case of user-provided gradients (not implemented in the API yet).

[*Error, StatusCode, mDVs, mResps*] = VDOC_RunRSA(*TaskNumber, StatusCode, mDVs, mResps*)

Function to run the response surface approximate optimization (RSA) task with the specified ID. You may need to call this function in a loop, until the returned value of the status code will indicate that the process is completed (F_DONE). After each call to this function you may be asked to evaluate the responses at one or several points with the provided values of design variables (F_RESP).

When this function is called for the first time StatusCode should be set to F_INIT=0. Later on VisualDOC will be updating this StatusCode and no more changes should be made to its value.

When this function is called NOT for the first time user should provide the values of ALL the design variables that he got as return arguments from the previous call to **VDOC_RunRSA**. The order of the design variables should correspond to the transfer indices obtained from **VDOC_PutInputAll** or **VDOC_GetInputAll**. Thus, only 2 arguments could be provided during the first functional call instead of 4, if desired. If 4 arguments will be provided, the values of the design variables and responses will be ignored.

When responses are supplied to this functional call, ALL the responses should be present. The order of the responses should correspond to the transfer indices obtained from **VDOC_PutRespAll** or **VDOC_GetRespAll**.

Only RSA type tasks will be run using this call. An attempt to run DGO or DOE task will result in error.

Input Argument	Description
TaskNumber	The RSA task number to run. One of the task numbers returned by **VDOC_MakeTask**. The task number should be always positive.
StatusCode	Status code of the run. Specifies what step of the

	run is being currently executed Possible values are: F_INIT=0, F_DONE=1, F_RESP=2 Complete description of all possible codes is given in the file “defs.h” in enumeration “Design_Step”
mDVs	Two-dimensional array with the values of the design variables. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL design variables. During the very first call to this function the values in the array will be ignored.
mResps	Two-dimensional array with the values of the responses. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL responses. The values of the responses should be evaluated at the points with the values of the design variables from array *mDVs*. During the very first call to this function the values in the array will be ignored.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10, DB_EXCEPTION=-91
StatusCode	Status code of the run. Specifies what step of the run is being currently executed Possible return values are: F_DONE=1, F_RESP=2 Complete description of all possible codes is given in the file “defs.h” in enumeration “Design_Step”
mDVs	Two-dimensional array with the values of the design variables. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL

	design variables. All responses should be evaluated at the provided points.
mResps	Two-dimensional array with the values of the responses. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL responses.

[*Error, StatusCode, mDVs, mResps*] = VDOC_RunDOE(*TaskNumber, StatusCode, mDVs, mResps*)

Function to run the design of experiments (DOE) task with the specified ID. You may need to call this function in a loop, until the returned value of the status code will indicate that the process is completed (F_DONE). After each call to this function you may be asked to evaluate the responses at one or several points with the provided values of design variables (F_RESP).

When this function is called for the first time StatusCode should be set to F_INIT=0. Later on VisualDOC will be updating this StatusCode and no more changes should be made to its value.

When this function is called NOT for the first time user should provide the values of ALL the design variables that he got as return arguments from the previous call to **VDOC_RunDOE** The order of the design variables should correspond to the transfer indices obtained from **VDOC_PutInputAll** or **VDOC_GetInputAll**. Thus, only 2 arguments could be provided during the first functional call instead of 4, if desired. If 4 arguments will be provided, the values of the design variables and responses will be ignored.

When responses are supplied to this functional call, ALL the responses should be present. The order of the responses should correspond to the transfer indices obtained from **VDOC_PutRespAll** or **VDOC_GetRespAll**.

Only DOE type tasks will be run using this call. An attempt to run DGO or RSA task will result in error.

Input Argument	Description
TaskNumber	The RSA task number to run. One of the task numbers returned by **VDOC_MakeTask**. The task number should be always positive.
StatusCode	Status code of the run. Specifies what step of the run is being currently executed Possible values are: F_INIT=0, F_DONE=1, F_RESP=2 Complete description of all possible codes is given in the file “defs.h” in enumeration “Design_Step”

mDVs	Two-dimensional array with the values of the design variables. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL design variables. During the very first call to this function the values in the array will be ignored.
mResps	Two-dimensional array with the values of the responses. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL responses. The values of the responses should be evaluated at the points with the values of the design variables from array *mDVs*. During the very first call to this function the values in the array will be ignored.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10, DB_EXCEPTION=-91
StatusCode	Status code of the run. Specifies what step of the run is being currently executed Possible return values are: F_DONE=1, F_RESP=2 Complete description of all possible codes is given in the file “defs.h” in enumeration “Design_Step”
mDVs	Two-dimensional array with the values of the design variables. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL design variables. All responses should be evaluated at the provided points.
mResps	Two-dimensional array with the values of the responses. The number of rows should equal to the number of points, the number of columns should equal to the number of ALL responses.

[*Error, BestTaskNumber, BestObj, WorstConstr, BestIterNumber, BestSubIterNumber, StopCode, RunStatus*] = VDOC_GetFinalResultsOptim(*TaskNumber*)

Function to get final results of the optimization for the specified task The obtained best task number and ID numbers of the best iteration and sub-iteration should be used to extract more detailed information.

Note, that when the discrete optimization is performed, the returned value of the parameter *BestTaskNumber* will be different from the task number supplied. This is because discrete optimization internally creates and solves several tasks. You should use the returned value of the parameter *BestTaskNumber* to retrieve more detailed information about best iteration or best point.

When continuous optimization is performed, the returned values of the parameter *BestTaskNumber* is equal to the supplied value of the parameter *TaskNumber*.

Input Argument	Description
TaskNumber	The DGO or RSA task number. One of the task numbers returned by **VDOC_MakeTask**. The task number should be always positive.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
BestTaskNumber	The task number with the best results. When the discrete optimization is performed, the returned value of the parameter *BestTaskNumber* will be different from the task number supplied (*TaskNumber*). When continuous optimization is performed, the returned values of the parameter *BestTaskNumber* is equal to the supplied value of the parameter *TaskNumber.* You should use the returned value of the parameter *BestTaskNumber* to retrieve more detailed information about best iteration or best point.
BestObj	Objective function for the best point found during optimization.
WorstConstr	Worst constraint for the best point found during

	optimization.
BestIterNumber	Iteration number for the best point found during optimization.
BestSubIterNumber	Sub-iteration number for the best point found during optimization.
StopCode	Optimization Stop Code. Possible values are given in enumeration "StopCode" in the file "defs.h"
RunStatus	Run status of the optimization task. Possible values are: NotRun = 0, Running = 1, Done = 2, RunTimeError = 4

[*Error, PointNumber, Obj, Constr, BestPointNumber, BestObj, WorstConstr*] = VDOC_GetSubIter (*TaskNumber, IterNumber, SubIterNumber*)

API call to get the attributes of the specified iteration and subiteration from the specified optimization task. Values of major iteration numbers and sub-iteration numbers always start from 0.

Input Argument	**Description**
TaskNumber	The DGO or RSA task number. The number of the best task returned by **VDOC_GetFinalResultsOptim**. The task number should be always positive.
IterNumber	Iteration number for which the results are desired. Should be non-negative.
SubIterNumber	Sub-iteration number for which the results are desired. Should be non-negative.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
PointNumber	Number of the point corresponding to the desired iteration and sub-iteration.
Obj	Objective function value at the point corresponding to the desired iteration and sub-

	iteration.
Constr	Value of the worst constraint at the point corresponding to the desired iteration and sub-iteration.
BestPointNumber	Number of the best SO FAR point found during optimization.
BestObj	Objective function for the best SO FAR point found during optimization.
WorstConstr	Worst constraint for the best SO FAR point found during optimization.

[*Error, vDVs, vResps*] = VDOC_GetDPoint(*TaskNumber, PointNumber*)

API call to get the array of the input parameter values and the array of responses for the specified point from results of the specified task. Point numbering always starts from 0. Point numbers are unique inside each task.

Input Argument	**Description**
TaskNumber	The DGO or RSA task number. The number of the best task returned by **VDOC_GetFinalResultsOptim**. The task number should be always positive.
PointNumber	Desired Point number.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
vDVs	Vector of design variables at the desired point.
vResps	Vector of responses at the desired point.

[*Error, mDVs, mResps, vToAnalyze*] = VDOC_GetDPointsAll(*TaskNumber*)

API call to get the design variables and responses for all the points visited by VisualDOC in the specified task. The design variables and responses are arranged in the form of matrices. The vector of keys specifying whether the particular point should be analyzed if used again is also provided after this function call.

Input Argument	Description
TaskNumber	The DGO or RSA task number. The number of the best task returned by **VDOC_GetFinalResultsOptim**. The task number should be always positive.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
mDVs	Two-dimensional array with the values of the design variables. The number of rows is equal to the number of points, the number of columns is equal to the number of ALL design variables.
mResps	Two-dimensional array with the values of the responses. The number of rows is equal to the number of points, the number of columns is equal to the number of ALL responses.
vToAnalyze	Vector of flags defining whether this point should be analyzed if used again. The number of elements is equal to the number of points. Possible values are: 1 – The point should be analyzed; 0 – Not.

[*Error, DEffic, AEffic, GEffic, ScVar, LnDet, Resol*] = VDOC_GetDOEEffic (*TaskNumber*)

API call to get the characteristics of the geometrical distribution of points in DOE. The values are extracted from the specified task. The corresponding values are calculated only if the DOE task was to create points only, without calculating responses or model coefficients.

Input Argument	Description
TaskNumber	The DOE task number. The task number should be always positive.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
DEffic	D-efficiencyiy

AEffic	A-efficiency
ScVar	Scaled predicted variance
LnDet	Logarithm of the determinant of the *X'X* matrix, where *X* is the model matrix.
Resol	Resolution of the fractional factorial design if applicable.

[*Error, vParamsANOVA*] = VDOC_GetDOEANOVA(*TaskNumber, DBIDResp*)

API call to get parameters of ANOVA table for the specified response from the DOE run. The corresponding values are calculated only if the DOE task included creating the response surface model. The 15 parameters are returned in one vector.

Input Arguments	**Description**
TaskNumber	The DOE task number. The task number should be always positive.
DBIDResp	Response database ID.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
vParamsANOVA	Vector with the parameters of the ANOVA table for the specified response. At least 15 elements. See description in the table below.

Description of the elements of the array with the ANOVA table parameters.

Name	**1-based Index**	**Description**
SSR	1	Sum of squares due to regression (response surface model)
SSE	2	Sum of squares due to error (Error sum of squares)
SYY	3	Total sum of squares
MSR	4	Mean squared regression
MSE	5	Mean squared error

Name	1-based Index	Description
F0	6	F_0 value for the approximation
PValue	7	*P*-value corresponding to F_0
SE	8	Sum of the errors
PRESS	9	Predicted Error Sum of Squares (*PRESS*)
RMSE	10	Root-mean square error
MeanResp	11	Mean value of the response
CV	12	Coefficient of variation for the whole response surface model
R2	13	Coefficient of multiple determinations, R^2
R2Adj	14	Adjusted R^2
R2PRESS	15	R^2 from *PRESS*

[*Error, vPredResp, vResi, vStdResi, vStuResi, vRStuResi, vPRESSResi, vCookD, vHDiag*] = VDOC_GetDOEResiduals(*TaskNumber, DBIDResp*)

API call to get various residuals (errors) for the specified response from the DOE run for all the points in the DOE run. The corresponding values are calculated only if the DOE task included creating the response surface model.

Input Arguments	Description
TaskNumber	The DOE task number. The task number should be always positive.
DBIDResp	Response database ID.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
vPredResp	Vector with predicted response values for all the points of the current DOE task.
vResi	Vector with the residual values for all the points of the current DOE task.

vStdResi	Vector with the standardized residual values for all the points of the current DOE task.
vStuResi	Vector with the studentized residual values for all the points of the current DOE task.
vRStuResi	Vector with the *R*-studentized residual values for all the points of the current DOE task
vPRESSResi	Vector with the *PRESS*-residual values for all the points of the current DOE task.
vCookD	Vector with Cook's *D*-statistic values for all the points of the current DOE task.
vHDiag	Vector with the values of the diagonal elements of the *H* matrix for all the points of the current DOE task. (*H = X*(X'X)**(-1)*X'*, where *X* is the model matrix)

[*Error, vCoeffs, vModelExp, vStdErr, vTStat, vPVal*] = VDOC_GetDOEModel(*TaskNumber, DBIDResp*)

API call to get parameters of the polynomial response surface model created during the DOE run. The corresponding values are calculated only if the DOE task included creating the response surface model.

Input Arguments	**Description**
TaskNumber	The DOE task number. The task number should be always positive.
DBIDResp	Response database ID.
Return Parameters	**Description**
Error	Return code. Possible values are: DB_SUCCESS=1 DB_FAIL=-2, DB_BAD_ARGS=-10
vCoeffs	Vector with the values of the coefficients in front of the terms of the response surface model. The size of this vector is equal to the number of terms in the response surface model.
vModelExp	Vector of exponents of the independent design variables for each terms of the responses surface model. The size of this vector is equal to the number of terms in the response surface model times the number of independent design variables

	(NDV). The elements in this array are organized as follows: at first NDV exponents corresponding to the design variables in the first term, then NDV exponents corresponding to the design variables in the second terms, etc. The constant terms has all the exponents equal to zero.
vStdErr	Vector with the values of the standard errors of the individual terms of the response surface model. The size of this vector is equal to the number of terms in the response surface model.
vTStat	Vector with the values of the *t*-statistic of the individual terms of the response surface model. The size of this vector is equal to the number of terms in the response surface model.
vPVal	Vector with the *p*-values of the individual terms of the response surface model. The size of this vector is equal to the number of terms in the response surface model.

Functional Listing of VisualDOC MATLAB API functions

GENERAL PURPOSE CALL

VDOC_SetPath

GENERAL DATABASE ACCESS CALLS

VDOC_OpenDatabase
VDOC_CloseDatabase
VDOC_IsOpen
VDOC_GetDBError

CALLS TO ACCESS INPUT OBJECTS

VDOC_PutInputAll
VDOC_GetInputAll
VDOC_PutInputObj
VDOC_GetInputObj
VDOC_ChangeInputType

CALLS TO ACCESS RESPONSE OBJECTS

VDOC_PutRespAll
VDOC_GetRespAll
VDOC_PutRespObj
VDOC_GetRespObj
VDOC_ChangeRespType

CALLS TO ACCESS USER DEFINED POINTS (G-POINTS)

VDOC_PutGPoints
VDOC_GetGPoints

CALLS TO ACCESS AUXILIARY OBJECT

VDOC_PutDiscreteSet
VDOC_GetDiscreteSet

CALLS TO ACCESS DESIGN CONTROL OBJECT. GENERAL DESIGN CONTROL

VDOC_PutTaskType
VDOC_GetTaskType

CALLS TO ACCESS DESIGN CONTROL OBJECT. DGO CONTROL

VDOC_PutDGOControlGeneral
VDOC_PutDGOControlConstr
VDOC_PutDGOControlConv
VDOC_PutDGOControlFD
VDOC_GetDGOControl

CALLS TO ACCESS DESIGN CONTROL OBJECT. RSA CONTROL

VDOC_PutRSAControlIni
VDOC_PutRSAControlGeneral
VDOC_PutRSAControlConstr
VDOC_PutRSAControlConv
VDOC_PutRSAControlMoveLim
VDOC_GetRSAControl

CALLS TO ACCESS DESIGN CONTROL OBJECT. DOECONTROL

VDOC_PutDOEControlGeneral
VDOC_PutDOEControlDesign
VDOC_GetDOEControl

CALL TO MAKE A TASK

VDOC_MakeTask

CALLS TO RUN FUNCTIONAL MODULES

VDOC_RunDGO
VDOC_RunRSA
VDOC_RunDOE

CALLS TO EXTRACT RESULTS INFORMATION FROM A DATABASE

VDOC_GetFinalResultsOptim
VDOC_GetSubIter
VDOC_GetDPoint
VDOC_GetDPointsAll

VDOC_GetDOEEffic
VDOC_GetDOEANOVA
VDOC_GetDOEResiduals
VDOC_GetDOEModel

Alphabetical Listing of VisualDOC MATLAB API functions

VDOC_ChangeInputType
VDOC_ChangeRespType
VDOC_CloseDatabase
VDOC_GetDBError
VDOC_GetDGOControl
VDOC_GetDiscreteSet
VDOC_GetDOEANOVA
VDOC_GetDOEControl
VDOC_GetDOEEffic
VDOC_GetDOEModel
VDOC_GetDOEResiduals
VDOC_GetDPoint
VDOC_GetDPointsAll
VDOC_GetFinalResultsOptim
VDOC_GetGPoints
VDOC_GetInputAll
VDOC_GetInputObj
VDOC_GetRespAll
VDOC_GetRespObj
VDOC_GetRSAControl
VDOC_GetSubIter
VDOC_GetTaskType
VDOC_IsOpen
VDOC_MakeTask
VDOC_OpenDatabase
VDOC_PutDGOControlConstr
VDOC_PutDGOControlConv
VDOC_PutDGOControlFD
VDOC_PutDGOControlGeneral
VDOC_PutDiscreteSet
VDOC_PutDOEControlDesign
VDOC_PutDOEControlGeneral
VDOC_PutGPoints
VDOC_PutInputAll
VDOC_PutInputObj
VDOC_PutRespAll
VDOC_PutRespObj
VDOC_PutRSAControlConstr
VDOC_PutRSAControlConv
VDOC_PutRSAControlGeneral
VDOC_PutRSAControlIni
VDOC_PutRSAControlMoveLim
VDOC_PutTaskType
VDOC_RunDGO
VDOC_RunDOE
VDOC_RunRSA
VDOC_SetPath