Virtuoso ADE Explorer and ADE Assembler SKILL Reference

Product Version ICADVM20.1
October 2020

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Contents

Preface

This manual describes the SKILL functions that you can use with Virtuoso ADE Explorer and ADE Assembler. This manual assumes you are familiar with the Cadence SKILL™ language.

This preface describes the following:

Scope

Unless otherwise noted, the functionality described in this guide can be used in both mature node (for example, IC6.1.8) and advanced node (for example, ICADVM18.1) releases.

Label	Meaning
(ICADVM18.1 Only)	Features supported only in the ICADVM18.1 advanced nodes and advanced methodologies releases.
(IC6.1.8 Only)	Features supported only in mature node releases.

Licensing Requirements

Virtuoso ADE Assembler requires the 95260, Virtuoso ADE Assembler license.

In addition, you would require the 95510, Virtuoso_Variation_Analysis_Op license for the following flows:

- The layout-dependent effects flow described in Chapter 13 of this user guide
- The electrically aware design flow described in the Virtuoso® Electrically Aware Flow Guide
- The voltage dependent rules flow described in the Virtuoso® Voltage Dependent Rules Flow Guide

For information on licensing in the Virtuoso design environment, see *Virtuoso Software Licensing and Configuration Guide*.

Related Documentation

What's New and KPNS

- *Virtuoso ADE Assembler What's New*
- *Virtuoso ADE Assembler Known Problems and Solutions*

Installation, Environment, and Infrastructure

- *Cadence Installation Guide*
- *Virtuoso Design Environment User Guide*

SKILL Language

The SKILL programming language is often used with other Virtuoso products or requires knowledge of a special language. The following documents give you more information about these tools and languages.

- *Cadence SKILL Language User Guide*
- *Cadence SKILL Language Reference*
- *Cadence SKILL++ Object System Reference*

- *OCEAN Reference*
- *OCEAN XL Reference*
- *Virtuoso Analog Design Environment L SKILL Reference*
- *Virtuoso Analog Design Environment XL SKILL Reference*
- *Virtuoso Design Environment SKILL Functions Reference*
- *Cadence Application Infrastructure User Guide*

Technology Information

- *Virtuoso Technology Data User Guide*
- *Virtuoso Technology Data ASCII Files Reference*
- *Virtuoso Technology Data SKILL Reference*

Virtuoso Tools

- *Virtuoso ADE Explorer User*
- *Virtuoso Schematic Editor User Guide*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator User Guide*
- *Spectre Circuit Simulator Reference*
- *Spectre Circuit Simulator and Accelerated Parallel Simulator RF Analysis in ADE Explorer User Guide*
- *Virtuoso UltraSim Simulator User Guide*
- *Spectre AMS Designer Simulator User Guide*
- *Virtuoso Parasitic Estimation and Analysis User Guide*
- *Virtuoso Visualization and Analysis Tool User Guide*
- *Component Description Format User Guide*
- *Analog Expression Language Reference*

Additional Learning Resources

Video Library

The Video Library on the Cadence Online Support website provides a comprehensive list of videos on various Cadence products.

To view a list of videos related to a specific product, you can use the *Filter Results* feature available in the pane on the left. For example, click the *Virtuoso Layout Suite* product link to view a list of videos available for the product.

You can also save your product preferences in the Product Selection form, which opens when you click the *Edit* icon located next to *My Products*.

Virtuoso Videos Book

You can access certain videos directly from Cadence Help. To learn more about this feature and to access the list of available videos, see Virtuoso Videos.

Rapid Adoption Kits

Cadence provides a number of Rapid Adoption Kits that demonstrate how to use Virtuoso applications in your design flows. These kits contain design databases and instructions on how to run the design flow.

In addition, Cadence offers the following training courses on the Virtuoso ADE Explorer, Virtuoso ADE Assembler, and the related flows:

- Virtuoso ADE Assembler
- Variation Analysis Using the Virtuoso ADE Assembler
- Virtuoso Analog Design Environment
- Virtuoso Schematic Editor
- Analog Modeling with Verilog-A
- Behavioral Modeling with Verilog-AMS
- Real Modeling with Verilog-AMS
- Spectre Simulations Using Virtuoso ADE

- Virtuoso UltraSim Full-Chip Simulator
- Virtuoso Simulation for Advanced Nodes
- Virtuoso Electrically-Aware Design with Layout Dependent Effects

Cadence also offers the following training courses on the SKILL programming language, which you can use to customize, extend, and automate your design environment:

- SKILL Language Programming Introduction
- SKILL Language Programming
- Advanced SKILL Language Programming

To explore the full range of training courses provided by Cadence in your region, visit Cadence Training or write to training_enroll@cadence.com.

Note: The links in this section open in a separate web browser window when clicked in Cadence Help.

Help and Support Facilities

Virtuoso offers several built-in features to let you access help and support directly from the software.

- The Virtuoso *Help* menu provides consistent help system access across Virtuoso tools and applications. The standard Virtuoso *Help* menu lets you access the most useful help and support resources from the Cadence support and corporate websites directly from the CIW or any Virtuoso application.
- The Virtuoso Welcome Page is a self-help launch pad offering access to a host of useful knowledge resources, including quick links to content available within the Virtuoso installation as well as to other popular online content.

  The Welcome Page is displayed by default when you open Cadence Help in standalone mode from a Virtuoso installation. You can also access it at any time by selecting *Help – Virtuoso Documentation Library* from any application window, or by clicking the *Home* button on the Cadence Help toolbar (provided you have not set a custom home page).

For more information, see Getting Help in *Virtuoso Design Environment User Guide.*

Customer Support

For assistance with Cadence products:

- Contact Cadence Customer Support

  Cadence is committed to keeping your design teams productive by providing answers to technical questions and to any queries about the latest software updates and training needs. For more information, visit https://www.cadence.com/support.

- Log on to Cadence Online Support

  Customers with a maintenance contract with Cadence can obtain the latest information about various tools at https://support.cadence.com.

Feedback about Documentation

You can contact Cadence Customer Support to open a service request if you:

- Find erroneous information in a product manual
- Cannot find in a product manual the information you are looking for
- Face an issue while accessing documentation by using Cadence Help

You can also submit feedback by using the following methods:

- In the Cadence Help window, click the *Feedback* button and follow instructions.
- On the Cadence Online Support Product Manuals page, select the required product and submit your feedback by using the *Provide Feedback* box.

Understanding Cadence SKILL

Cadence SKILL is a high-level, interactive programming language based on the popular artificial intelligence language, Lisp. It lets you customize and extend your design environment. Using SKILL you can validate the steps of your algorithm incrementally before incorporating them into a larger program.

For more information about the SKILL language, see Getting Started in the *SKILL Language User Guide*.

Using SKILL Code Examples

The SKILL APIs in this user manual are explained with illustrative code examples.

You can copy these examples from the manual and paste them directly into the Command Interpreter Window (CIW) or use the code in non-graphical SKILL mode.

Sample SKILL Code

The following code sample shows the syntax of a SKILL API that accepts three arguments.

axlGetRunStatus

```
axlGetRunStatus(
    t_sessionName                          <- Required argument
    [ ?optionName t_optionName ]           <- Optional keyword argument
    [ ?historyName t_historyName ]         <- Optional keyword argument
    )
    => l_statusValues                      <- Return value
```

The first argument `t_sessionName` is a required argument, where `t` signifies the data type of the argument. The second and third arguments `?optionName t_optionName` and `?historyName t_historyName` are optional keyword arguments (identified by a question mark), which are specified in name-value pairs and can be placed in any order during the function call.

The return value is the value that the SKILL API returns after evaluating the expression. In this case, it is a list of status values, *`l_statusValues`*.

Example

```
axlSession=axlGetWindowSession( hiGetCurrentWindow())
=> "session0"
axlGetRunStatus("session0" ?historyName "Interactive.10" ?optionName "tests")
=> (1 2)
```

Return value

Value of the session name argument

Question mark and argument name before the value of the keyword argument

Question mark and argument name before the value of the keyword argument

Accessing API Help

Quick reference information for SKILL APIs is available from the CIW and the SKILL API Finder. To access the reference information for a particular SKILL API, do one of the following:

- Type `help <function_name>` in the CIW.
- Type `startFinder ( [ ?funcName` *`t_functionName`* `] )` in the CIW.
- Start the SKILL API Finder from the CIW by choosing *Tools – Finder* or type `cdsFinder` on the UNIX command line.

  In the *Search in* field of the displayed Cadence SKILL API Finder window, type the SKILL API name for which you want to display the help information and click *Go*.

  The matches for the searched SKILL API appear in the *Results* area.

  To view the complete documentation of the searched SKILL API, select the API name in the *Results* area and click the *More Info* button. The complete documentation of the selected SKILL API appears in a new Cadence Help window.

Typographic and Syntax Conventions

The following typographic and syntax conventions are used in this manual.

text	Indicates names of manuals, menu commands, buttons, and fields.
`text`	Indicates text that you must type exactly as presented. Typically used to denote command, function, routine, or argument names that must be typed literally.
`z_argument`	Indicates text that you must replace with an appropriate argument value. The prefix (in this example, *`z_`*) indicates the data type the argument can accept and must not be typed.
`|`	Separates a choice of options.
`{ }`	Encloses a list of choices, separated by vertical bars, from which you **must** choose one.
`[ ]`	Encloses an optional argument or a list of choices separated by vertical bars, from which you **may** choose one.
`[ ?argName` *`t_arg`* `]`	Denotes a *key argument*. The question mark and argument name must be typed as they appear in the syntax and must be followed by the required value for that argument.
`...`	Indicates that you can repeat the previous argument.
	Used with brackets to indicate that you can specify zero or more arguments.
	Used without brackets to indicate that you must specify at least one argument.
`,...`	Indicates that multiple arguments must be separated by commas.
`=>`	Indicates the values returned by a Cadence® SKILL® language function.
`/`	Separates the values that can be returned by a Cadence SKILL language function.

If a command-line or SKILL expression is too long to fit within the paragraph margins of this document, the remainder of the expression is moved to the next line and indented. In code excerpts, a backslash (\ ) indicates that the current line continues on to the next line.

Identifiers Used to Denote Data Types

Data type identifiers are used to indicate the type of value required by an API argument. These data types are denoted by a single letter that is prefixed to the argument label and is separated from the argument by an underscore; for example, `t` is the data type in `t_viewName`. Data types and underscores are used only as identifiers; they must not be typed when specifying the argument in a function.

Prefix	Internal Name	Data Type
`a`	array	array
`A`	amsobject	AMS object
`b`	ddUserType	DDPI object
`B`	ddCatUserType	DDPI category object
`C`	opfcontext	OPF context
`d`	dbobject	Cadence database object (CDBA)
`e`	envobj	environment
`f`	flonum	floating-point number
`F`	opffile	OPF file ID
`g`	general	any data type
`G`	gdmSpecIlUserType	generic design management (GDM) spec object
`h`	hdbobject	hierarchical database configuration object
`I`	dbgenobject	CDB generator object
`K`	mapiobject	MAPI object
`l`	list	linked list
`L`	tc	Technology file time stamp
`m`	nmpIlUserType	nmpIl user type
`M`	cdsEvalObject	cdsEvalObject
`n`	number	integer or floating-point number
`o`	userType	user-defined type (other)
`p`	port	I/O port
`q`	gdmspecListIlUserType	gdm spec list

Prefix	Internal Name	Data Type
r	defstruct	defstruct
R	rodObj	relative object design (ROD) object
s	symbol	symbol
S	stringSymbol	symbol or character string
t	string	character string (text)
T	txobject	transient object
u	function	function object, either the name of a function (symbol) or a lambda function body (list)
U	funobj	function object
v	hdbpath	hdbpath
w	wtype	window type
sw	swtype	subtype session window
dw	dwtype	subtype dockable window
x	integer	integer number
y	binary	binary function
&	pointer	pointer type

For more information, see *Cadence SKILL Language User Guide*.

1

Introduction to Maestro Functions

This chapter provides an introduction to SKILL scripts that you can write for maestro cellviews. In this chapter, you can find information about

- Creating SKILL Scripts for Maestro Cellviews
- Executing Scripts

Creating SKILL Scripts for Maestro Cellviews

You can create scripts for maestro cellviews by using the various SKILL functions described in the following sections:

- Functions to Migrate ADE L or ADE XL States to Maestro Cellviews
- Functions to Create, View, Edit, and Save Setup
- Functions to Run Simulations
- Functions to Work with Simulation Results

In addition, you can also use the functions defined in the *Analog Design Environment XL SKILL Reference Guide*.

Important Points to Note

- The SKILL functions given in this book can be used with the cellviews of type `maestro` that are saved using either ADE Explorer or ADE Assembler. When these functions are executed from the user interface (CIW), they get executed on the complete cellview, irrespective of the application in which a cellview is currently opened. For example, if you open a cellview that contains multiple tests in the ADE Explorer environment and run the maeGetSetup function to get the setup details, it returns the details of all the tests defined in the cellview instead of returning the details of the test that is opened in the ADE Explorer window.

- The functions listed in this user guide cannot be used to modify or save changes in ADE XL cellviews.

Executing Scripts

If the user interface for Virtuoso ADE Explorer or ADE Assembler is already open, you can directly run the SKILL functions for maestro cellviews from the Command Interpreter Window (CIW).

Alternatively, you can write a script using these functions in a `.il` file and execute by using the `virtuoso` executable, as shown in the following example:

Step 1: Write the following script in `test.il` file

```
maeOpenSetup("solutions" "amptest" "maestro")
maeRunSimulation()
maeWaitUntilDone('All)
maeExportOutputView()
```

Step 2: Execute the script

```
virtuoso -replay test.il -log myLog1 -report
```

The `-report` command line argument prints verbose messages on the command line.

If you include any call to the `printf` function in a script, the output of `printf` is written to the main Virtuoso log.

Important

The execution of the script does not break if any command fails to execute successfuly. Instead, the next command is run. Therefore, it is a good practice to view the report or run log after the complete script is run.

2

Functions for Maestro Cellviews

The SKILL functions described in this chapter are helpful in working with maestro cellviews in ADE Explorer and ADE Assembler. It describes the following SKILL functions:

Functions to Migrate ADE L or ADE XL States to Maestro Cellviews

Use the following functions to migrate ADE L or ADE XL states:

Function	Description
maeConvertAndCombineMultiADELToAssembler	Combines two or more ADE L states into one maestro cellview for ADE Assembler.
maeMigrateADELStateToMaestro	Migrates the given ADE L state to a maestro cellview that can be opened in ADE Explorer.
maeMigrateADEXLToMaestro	Migrates the given adexl view to a new maestro cellview that can be opened in ADE Assembler.

maeConvertAndCombineMultiADELToAssembler

```
maeConvertAndCombineMultiADELToAssembler(
     l_stateList
     t_maestroLib
     t_maestroCell
     [ ?maestroView t_maestroView ]
     [ ?migrateFrom t_migrateFrom ]
     [ ?rootPath t_rootPath ]
     [ ?overwrite g_overwrite ]
     )
     => l_cellviewDetails / nil
```

Description

Combines two or more ADE L states into one maestro cellview for ADE Assembler.

Arguments

l_stateList The list of ADE L states to be combined

t_maestroLib Name of the library in which the maestro cellview is to be saved.

t_maestroCell Name of the cell in which the maestro cellview is to be saved.

?maestroView *t_maestroView*

Name for the maestro cellview to be saved.

Default value: `"maestro"`

?migrateFrom *t_migrateFrom*

The method to find the given ADE L states

Possible values: `'directory`, `'cellview`

Default value: `'cellview`

?rootPath *t_rootPath*

The path that contains the libraries of ADE L states. This argument is used when the `?migrateFrom` argument is set to `'directory`.

Default value: `nil`

?overwrite *g_overwrite*

Boolean value that specifies if an existing view with the same name is to be overwritten by the new cellview

Default value: `t`

Value Returned

`l_cellviewDetails`	A list containing the names of library, cell, and view of the maestro cellview created by this function
`nil`	Unsuccessful operation

Example

The following example code migrates ADE L states to create a maestro cellview:

```
maeConvertAndCombineMultiADELToAssembler((list (list "Two_Stage_Opamp"
"OpAmp_AC_top" "spectre_AC") (list "Two_Stage_Opamp" "OpAmp_TRAN_top"
"spectre_TRAN")) "Two_Stage_Opamp" "OpAmp" ?maestroView "maestro_new")

=> ("Two_Stage_Opamp" "OpAmp" "maestro_new")


maeConvertAndCombineMultiADELToAssembler((list (list "Two_Stage_Opamp"
"OpAmp_AC_top" "spectre_AC" "spectre") (list "Two_Stage_Opamp" "OpAmp_TRAN_top"
"spectre_TRAN" "spectre")) "Two_Stage_Opamp" "OpAmp" ?migrateFrom 'directory
?rootPath "~/.artist_states/"))
```

maeMigrateADELStateToMaestro

```
maeMigrateADELStateToMaestro(
     t_stateLib
     t_stateCell
     t_stateName
     [ ?maestroLib t_maestroLib ]
     [ ?maestroCell  t_maestroCell ]
     [ ?maestroView t_maestroView ]
     [ ?migrateFrom s_migrateFrom ]
     [ ?statePath t_statePath ]
     [ ?simulator t_simulator ]
     [ ?overwrite g_overwrite ]
     )
     => l_cellviewDetails / nil
```

Description

Migrates the given ADE L state to a maestro cellview that can be opened in ADE Explorer.

Arguments

t_stateLib	Name of the library for which the ADE L state is available.
t_stateCell	Name of the cell for which the ADE L state is available.
t_stateView	Name of the view for which the ADE L state is available.
?maestroLib *t_maestroLib*	Name of the library in which the maestro cellview is to be saved. If not specified, the value specified for *t_stateLib* is used. You can specify any other library name. If the specified library does not exist, a library is created with that name.
?maestroCell *t_maestroCell*	Name of the cell in which the maestro cellview is to be saved. If not specified, the value specified for *t_stateCell* is used.
?maestroView *t_maestroView*	Name for the maestro cellview to be saved. Default value: "maestro"

`?migrateFrom` *`s_migrateFrom`*	Specifies if the state is to be migrate from cellview or directory. Possible values: `'cellview` or `'directory` Default value: `'cellview`
`?statePath` *`t_statePath`*	Path to the directory if *`s_migrateFrom`* is set to `'directory`.
`?simulator` *`t_simulator`*	Specifies the simulator name. This value is used when *`s_migrateFrom`* is set to `'directory`. Default value: `"spectre"`
`?overwrite` *`g_overwrite`*	Specifies whether to overwrite the cellview, if already exists.

Value Returned

`l_cellviewDetails`	A list containing the names of library, cell and view of the maestro cellview to which the state is successfully migrated.
`nil`	Unsuccessful operation.

Example

The following example code migrates an ADE L state from a cellview state:

```
maeMigrateADELStateToMaestro("Two_Stage_Opamp" "OpAmp_AC_top" "spectre_state1" )

=> ("Two_Stage_Opamp" "OpAmp_AC_top" "maestro1")
```

The following example code migrates an ADE L state from a state file saved in a directory:

```
maeMigrateADELStateToMaestro("Two_Stage_Opamp" "OpAmp_AC_top" "AC_active"
?maestroLib "aaa" ?migrateFrom 'directory ?statePath
"./libs/Two_Stage_Opamp/OpAmp/adexl/test_states/"  ?overwrite nil)

=> ("Two_Stage_Opamp" "OpAmp_AC_top" "maestro1")
```

maeMigrateADEXLToMaestro

```
maeMigrateADEXLToMaestro(
     t_stateLib
     t_stateCell
     t_stateName
     [ ?maestroLib t_maestroLib ]
     [ ?maestroCell  t_maestroCell ]
     [ ?maestroView t_maestroView ]
     [ ?overwrite g_overwrite ]
     [ ?skipHistory g_skipHistory ]
     )
     => t / nil
```

Description

Migrates the given adexl view to a new maestro cellview that can be opened in ADE Assembler.

Arguments

t_stateLib	Name of the library of the ADE XL view.
t_stateCell	Name of the cell of the ADE XL view.
t_stateView	Name of ADE XL cellview.
?maestroLib *t_maestroLib*	Name of the library in which the new maestro cellview is to be saved. If not specified, the value specified for *t_stateLib* is used. You can specify any other library name. If the specified library does not exist, a library is created with that name.
?maestroCell *t_maestroCell*	Name of the cell in which the maestro cellview is to be saved. If not specified, the value specified for *t_stateCell* is used.
?maestroView *t_maestroView*	Name of the maestro cellview to be created. Default value: "maestro"

`?overwrite` *`g_overwrite`*	Boolean value that specifies whether or not to overwrite an existing maestro view with the same name as that specified by the `?maestroView` argument. Default value: `t` When set to `nil`, the cellview is not migrated if an existing maestro cellview is found with the same name.
`?skipHistory` *`g_skipHistory`*	Boolean value that specifies whether or not to skip histories while copying data from the adexl cellview into the new maestro cellview. Default value: `nil` When set to `t`, the newly created maestro cellview does not contain any history.

Value Returned

`t`	Successful creation of the maestro cellview.
`nil`	Unsuccessful operation.

Example

The following example code migrates the `adexl` cellview to a new cellview named `maestro`.

```
maeMigrateADEXLToMaestro("solutions" "ampTest" "adexl")
=> t
```

Functions to Create, View, Edit, and Save Setup

Use the following functions to work with the setup in maestro cellviews:

Function	Description
maeAddOutput	Adds or updates an output for the given test in the currently active setup.
maeCreateTest	Creates a new test and adds it to the given maestro session. If a source test name is given, it creates a copy of that test. If that is not given, creates a new blank test and sets the design name using the library, cell, and view name.
maeCloseSession	Closes the session that are opened using maeOpenSetup in the SKILL code. This function cannot be used to close the sessions opened from the Virtuoso user interface.
maeConvertViewForReferencedAndLocalRunPlanCorners	Converts all local and referenced corners created in cellviews saved using IC6.1.8 ISR9 or ICADVM18.1 ISR9 to referenced corners supported in earlier versions.
maeConvertViewForIntegratedHistoryManagement	Converts the maestro views in which history setup is integrated with the main setup database to an enhanced format in which separate history management is enabled.
maeConvertViewForSeparateHistoryManagement	Converts the maestro views in which history setup is integrated with the main setup database to an enhanced format in which separate history management is enabled.
maeDeleteCorner	Deletes the specified corner from the setup database.
maeDeleteOutput	Deletes the specified output from the setup database.
maeDeleteParameter	Deletes the specified parameter from the setup database.
maeDeleteVar	Deletes the specified variable from the setup database.

maeExportSetupForExplorer	Displays the Save A Copy form in ADE Explorer, which you can use to save the current setup. It is a callback function.
maeGetAnalysis	Returns a list containing name-value pairs of the options set for the given analysis. By default, the function returns only those options for which the value is not empty. If the `g_includeEmpty` argument is set to `t`, it returns all the options. You can also specify the name of a specific option for which you want to know the value.

maeGetCurrentRunMode	Returns the name of the run mode set in the given session or the given run.
maeGetEnabledAnalysis	Returns a list containing names of all the analyses enabled for the given test.
maeGetEnvOption	Returns a list containing name-value pairs for all the environment option for the given test. By default, it returns all the environment options whose value is not `nil`. If the `includeEmpty` argument is set to `t`, the function returns all the environment options. If a specific option is specified, it returns the value of only that option.
maeGetExplorerTestName	Returns the name of the test opened in ADE Explorer. If you descend into ADE Explorer from ADE Assembler, use this function to get the name of the current test.
maeGetJobPolicy	Returns the details of the job policy attached to the given test. If no test name is given, the function returns the policy attached to the current setup.
maeGetJobPolicyByName	Returns a disembodied property list containing property-value pairs for the given job policy.
maeGetRunPlan	Returns a list of all the runs available in the run plan for the given session or history.
maeGetSetup	Returns the required setup details of variables, parameters, corners, and tests from the given session.
maeGetSessions	Returns a list of valid ADE Explorer or ADE Assembler sessions that are currently open.
maeGetTestEnvVar	Returns the value of the specified environment variable for the given test. This function can be used to get the value set for a particular test using the maeSetTestEnvVar function.
maeGetTestSession	Returns session handle to the given test. You can use this handle to view or modify the test details.
maeGetVar	Returns value of the given variable.

maeImportSetupForExplorer	Displays the Import Setup form in ADE Explorer, which you can use to import a saved `maestro` cellview in the current setup. It is a callback function.
maeIsSetupModified	Checks whether the setup has been modified after the last time it was saved in the given maestro session.
maeIsSingleTest	Returns `t` if the given cellview contains a single test or multiple tests.
maeIsValidMaestroSession	Confirms if the given session is a valid session for ADE Explorer or ADE Assembler.
maeLoadCorners	Loads corners from the given file into the corner setup of the current session or the specified session.
maeOpenSetup	Loads the given cellview and restores the setup details from the specified history. If no history name is specified, the active setup is loaded. If the specified view is already opened in the current Virtuoso session, it is not opened again. However, if the view is already open in some other Virtuoso session, it is opened in read mode in the current session. If the given cellview does not exist, the function creates a new cellview with the same name.
maeLoadSetupState	Loads the given setup state into the given session.
maeLoadStateForTest	Loads the saved ADE state for the given test.
maeSaveSetup	Saves a setup state for the given session.
maeStmGenerateWaveforms	Generates preview stimuli waveforms for the stimuli added for a maestro cellview that contains stimuli definitions added through the Stimuli Assignment form. These stimuli definitions are saved in `<lib-name>/<cell-name>/<view-name>/namedStimuli/stimuli.xml` file. The function examines each of the stimuli definitions for preview waveforms. If a stimuli definition has no associated preview waveform, or the waveform data is out of date, it generates a new SRR waveform data and saves it in the `<project-dir>/<lib-name>/<cell-name>/<view-name>/namedStimuli` directory.

maeSaveSetupState	Saves a setup state for the given session.
maeSetAnalysis	Adds an analysis or changes the enabled status of the specified analysis for the given test. The function can also be used to set the value of different options for the analysis.
maeSetCorner	Adds a new corner and enables/disables it for the given test.
maeSetCurrentRunMode	Updates the current run mode in ADE Assembler.
maeSetDesign	Sets a design for the given test. Use this function to change the design associated to a test.
maeSetEnableTestVar	Enables or disables the specified variables or local sweeps for the given test.
maeSetEnvOption	Sets values for one or more environment options for the given test.
maeSetHistoryLock	Locks or unlocks the given history in the given ADE Assembler session.
maeSetParameter	Adds a new parameter at the global level or corner level. If the parameter already exists, updates its value.
maeSetSpec	Adds a specification to an output defined for a test. You can also use this function to modify an existing specification for an output.
maeSetJobPolicy	Sets the given job policy to the specified setup for the given test.
maeSetSetup	Enables or disables the tests, global variables, parameters, and corners.
maeSetSimOption	Sets values for the specified simulator options of the given test. Multiple options can be specified in a single command.
maeSetTestEnvVar	Sets the value for the given environment variable at the test level. The value is used only by the specified test. Other tests in the session use the value set at the global level or specific values set for them, if any.

maeSetVar	Adds a variable to the given test or corner in the given ADE Explorer or ADE Assembler session. If the variable already exists, its value is updated.

maeAddOutput

```
maeAddOutput(
     t_outputName
     t_testName
     [ ?outputType t_outputType ]
     [ ?signalName t_signalName ]
     [ ?expr t_expr ]
     [ ?plot g_plot ]
     [ ?save g_save ]
     [ ?quote g_quote ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Adds or updates an output for the given test in the currently active setup.

Arguments

t_outputName	Name of the output to be added
t_testName	Name of the test in which output has to be added
?outputType *t_outputType*	Type of the output to be added Possible values: `"point"`, `"corners"`, `"sweeps"`, `"all"`, `"terminal"`, `"net"` Default: `"point"`
?signalName *t_signalName*	Name of the signal to be added. Relevant only when output type is `"terminal"` or `"net"`. Default: ""
?expr *t_expr*	Expression to be used to calculate the output. Relevant only when type is `point`.
?plot *g_plot*	Specifies if the output is to be plotted
?save *g_save*	Specifies if the output is to be saved
?session *t_sessionName*	Name of the session If not specified, output is added in the current session.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

```
maeAddOutput("Output1" "test1" )
```

Adds an output of type expr to `test1`.

```
maeAddOutput("Output2" "test1" ?outputType "point" ?expr "Output1*10")
```

Adds an output of type signal for `test1`.

```
maeAddOutput("Output3" "test1" ?outputType "net" ?signalName "/net2")
```

Adds an output of type net for `test1`.

maeCreateTest

```
maeCreateTest(
     t_testName
     [ ?sourceTest t_sourceTest ]
     [ ?lib t_lib ]
     [ ?cell t_cell ]
     [ ?view t_view ]
     [ ?simulator t_simulator ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Creates a new test and adds it to the given maestro session. If a source test name is given, it creates a copy of that test. If that is not given, creates a new blank test and sets the design name using the library, cell, and view name.

Arguments

t_testName	Name of the test to be created
[?sourceTest *t_sourceTest*]	Name of the source test to be copied to create the new test
[?lib *t_lib*]	Name of the library for the new test is to be created
[?cell *t_cell*]	Name of the cell for the new test is to be created
[?view *t_view*]	Name of the view for the new test is to be created
[?simulator *t_simulator*]	Name of the simulator to be used for the new test Default value: `"spectre"`
[?session *t_sessionName*]	Session name If not specified, the current session is used.

Values Returned

`t`	Returns `t` when the test is successfully created
`nil`	Returns `nil` if there is an error.

Example

```
maeCreateTest("solutions:ampTest:2" ?sourceTest "solution:ampTest:1")
```

Creates a copy of solution:ampTest:1 and name it as solutions:ampTest:2.

```
maeCreateTest("solutions:ampTest:2" ?lib "solutions" ?cell "ampTest" ?view
"schematic" ?simulator "spectre")
```

Creates a new test solutions:ampTest:2 and sets ‘solutions/ampTest/schematic’ as the design and ‘spectre’ as simulator.

maeCloseSession

```
maeCloseSession(
     [ ?session t_sessionName ]
     [ ?forceClose g_forceClose ]
     )
     => t / nil
```

Description

Closes the session that are opened using maeOpenSetup in the SKILL code. This function cannot be used to close the sessions opened from the Virtuoso user interface.

Arguments

`?session` *`t_sessionName`*	Name of the session to be closed. If not specified, the current session is closed.
`?forceClose` *`g_forceClose`*	A boolean value that specifies whether the current session needs to be closed immediately or not if simulations are pending. In both cases, appropriate information messages are added to the log. Possible values: ■ `t`: Immediately stops the simulations that are in progress and closes the session ■ `nil`: Waits for the simulations that are in progress to complete and then closes the session Default value: nil

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

```
; load the setup
maeOpenSetup("solutions" "ampTest" "maestro")
=>"fnxSession2"
```

```
; check which tests are there in setup
maeGetSetup()
=> ("solutions"amptest:1" "solutions:amptest:2")

; Enable a test "solution:amptest:1"
maeSetSetup(?tests '("solutions:amptest:1"))

; Set the value of global variable
"CAP" as 5p: maeSetVar("CAP" 5p)

; Disable corners C0 and C1
maeSetSetup(?corners '("C0"  "C1") ?enabled nil)

; Run Simulation
maeRunSimulation()

maeCloseSession()
=> t

; The tool waits for the simulation to complete before closing the session
```

maeConvertViewForReferencedAndLocalRunPlanCorners

```
maeConvertViewForReferencedAndLocalRunPlanCorners(
     t_libName
     t_cellName
     t_viewName
     )
     => t / nil
```

Description

Converts all local and referenced corners created in cellviews saved using IC6.1.8 ISR9 or ICADVM18.1 ISR9 to referenced corners supported in earlier versions.

Note: Use this function only when you need to use an earlier version of Virtuoso to run simulations for the run plans created in cellviews saved using IC6.1.8 ISR9 or ICADVM18.1 ISR9.

Arguments

t_libName	Name of the library
t_cellName	Name of the cell
t_viewName	Name of the maestro cellview

Value Returned

`t`	Successful conversion of the run plan
`nil`	Unsuccessful conversion

Example

```
; run this function before opening a cellview saved using IC6.1.8 ISR9 into an
; earlier version.

maeConvertViewForReferencedAndLocalRunPlanCorners("solutions" "ampTest"
"maestro")
=t
```

maeConvertViewForIntegratedHistoryManagement

```
maeConvertViewForIntegratedHistoryManagement(
     t_libName
     t_cellName
     t_viewName
     [ ?newLib t_newLibName ]
     [ ?newCell t_newCellName ]
     [ ?newView t_newViewName ]
     [ ?overwrite g_overwrite ]
     )
     => t / nil
```

Description

Converts the enhanced maestro view, in which separate history management is enabled, to the previous format in which histories are integrated with the main setup database.

Arguments

t_libName	Name of the library
t_cellName	Name of the cell
t_viewName	Name of the maestro cellview
?newView *t_newLibName*	Name of the library in which the converted cellview is to be saved.
?newView *t_newCellName*	Name of the cell in which the converted cellview is to be saved.
?newView *t_newViewName*	Name to be used for the converted maestro cellview
?overwrite *g_overwrite*	A boolean value specifying whether to overwrite *t_viewName* with the converted cellview.

Value Returned

t	The cellview is converted successfully.

`nil`	The cellview is not converted.

Example

```
maeConvertViewForIntegratedHistoryManagement("solutions" "ampTest" "maestro"
?newView "maestro_old")
> t=> t
```

maeConvertViewForSeparateHistoryManagement

```
maeConvertViewForSeparateHistoryManagement(
     t_libName
     t_cellName
     t_viewName
     [ ?newLib t_newLibName ]
     [ ?newCell t_newCellName ]
     [ ?newView t_newViewName ]
     [ ?overwrite g_overwrite ]
     )
     => t / nil
```

Description

Converts the maestro views in which history setup is integrated with the main setup database to an enhanced format in which separate history management is enabled.

Arguments

t_libName	Name of the library
t_cellName	Name of the cell
t_viewName	Name of the maestro cellview
?newView *t_newLibName*	Name of the library in which the converted cellview is to be saved.
?newView *t_newCellName*	Name of the cell in which the converted cellview is to be saved.
?newView *t_newViewName*	Name to be used for the converted maestro cellview
?overwrite *g_overwrite*	A boolean value specifying whether to overwrite *t_viewName* with the converted cellview.

Value Returned

t	The cellview is converted successfully.

`nil`	The cellview is not converted.

Example

```
maeConvertViewForSeparateHistoryManagement("solutions" "ampTest" "maestro"
?newView "maestro_new")
=> t
```

maeDeleteCorner

```
maeDeleteCorner(
     t_cornerName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Deletes the specified corner from the setup database.

Note: You cannot delete the nominal corner using this function.

Arguments

t_cornerName	Name of the corner to be deleted from the setup.
?session *t_sessionName*	Name of the session. If not specified, the corner is deleted from the current session.

Value Returned

t	Successful operation
nil	Unsuccessful operation

Example

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeDeleteCorner("C1")
=> t
```

maeDeleteOutput

```
maeDeleteOutput(
     t_outputName
     t_testName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Deletes the specified output from the setup database.

Arguments

t_outputName	Name of the output to be deleted from the setup.
t_testName	Name of the test for which corner is to be deleted.
?session *t_sessionName*	Name of the session. If not specified, the output is deleted from the current session.

Value Returned

t	Successful operation
nil	Unsuccessful operation

Example

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeDeleteOutput("out1" "test1")
=> t
```

maeDeleteParameter

```
maeDeleteParameter(
     t_parameterName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Deletes the specified parameter from the setup database.

Arguments

t_parameterName	Name of the corner to be deleted from the setup.
?session *t_sessionName*	Name of the session. If not specified, the parameter is deleted from the current session.

Value Returned

t	Successful operation
nil	Unsuccessful operation

Example

```
maeDeleteParameter("/I0/l")
=> t
```

maeDeleteVar

```
maeDeleteVar(
     t_varName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Deletes the specified variable from the setup database.

Arguments

t_varName	Name of the variable to be deleted from the setup.
?session *t_sessionName*	Name of the session. If not specified, the output is deleted from the current session.

Value Returned

t	Successful operation
nil	Unsuccessful operation

Example

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeDeleteVar("VDD")
=> t
```

maeExportSetupForExplorer

```
maeExportSetupForExplorer(
     t_session
     )
     => t / nil
```

Description

Displays the Save A Copy form in ADE Explorer, which you can use to save the current setup. It is a callback function.

Arguments

t_session	The current ADE Explorer session.

Values Returned

`t`	Returns `t` when the form is successfully displayed.
`nil`	Returns `nil` if there is an error.

Example

```
maeExportSetupForExplorer(session)
```

where, session is the currently active ADE Explorer session.

maeGetAnalysis

```
maeGetAnalysis(
     t_testName
     t_analysis
     [ ?includeEmpty g_includeEmpty ]
     [ ?option t_option ]
     [ ?session t_sessionName ]
     )
     => l_options / nil
```

Description

Returns a list containing name-value pairs of the options set for the given analysis. By default, the function returns only those options for which the value is not empty. If the *g_includeEmpty* argument is set to `t`, it returns all the options. You can also specify the name of a specific option for which you want to know the value.

Arguments

t_testName	Name of the test
t_analysis	Name of the analysis for which you want to return the settings
`[ ?includeEmpty` *g_includeEmpty* `]`	Includes or excludes the options for which no value is set
`[ ?option` *t_option* `]`	Name of the specific option for which you want to return the value
`[ ?session` *t_sessionName* `]`	Name of the session

Value Returned

l_options	List of options set for the given analysis
`nil`	Unsuccessful operation

Example

The following example code adds a new analysis `tran` and sets its options, stop time 500 and step 10:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeGetAnalysis( "solutions:ampTest:2" "ac" )
=> (("anaName" "ac") ("sweep" "Frequency") ("rangeType" "Start-Stop") ("start" "1")
("stop" "10G" ("incrType" "Logarithmic")  ("stepTypeLin" "Step Size")
("stepTypeLog" "Points Per Decade") ("dec" "20") ("outType" "Voltage") ("srcType"
"isource") ("perturbation" "linear") ("special" "None") ("save" "selected")
("oppoint" "no") ("annotate" "no") )
```

maeGetCurrentRunMode

```
maeGetCurrentRunMode(
     [ ?session t_sessionName ]
     [ ?run t_runName ]
     [ ?Abbreviations g_abbreviation ]
     )
     => t_runModeName / nil
```

Description

Returns the name of the run mode set in the given session or the given run.

Arguments

`?session` *`t_sessionName`*	Name of an ADE Assembler session. If not specified, the currently active session is considered.
`?run` *`t_runName`*	Name of the run from the run plan for which you need to get the run mode. If not specified, the run mode of the session is returned.
`?Abbreviations` *`g_abbreviation`*	Specifies whether to return the short name for the run mode. For example, for the `"Single Run, Sweeps, and Corners"` run mode it returns `"SRSC"`.

Value Returned

`t_runModeName`	Name of the run mode
`nil`	Unsuccessful operation

Example

The following example shows how the function can be used to return the run mode of the current session:

```
maeGetCurrentRunMode()
=> "Single Run, Sweeps and Corners"
```

The following example shows how the function can be used to return the run mode for a specific run in the current session:

```
maeGetCurrentRunMode(?run "Run.1")
=> "Monte Carlo"
```

maeGetEnabledAnalysis

```
maeGetEnabledAnalysis(
     t_testName
     [ ?session t_sessionName ]
     )
     => l_analysisNames / nil
```

Description

Returns a list containing names of all the analyses enabled for the given test.

Arguments

t_testName	Name of the test
?session *t_sessionName*	Name of the session Default: Current session

Value Returned

l_analysisnames	List of the names of analyses enabled for the given test
nil	Unsuccessful operation

Example

The following example code shows how to get the list of enabled analyses for a test:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeGetEnabledAnalysis( "solutions:ampTest:2")
=> ("ac" "tran")
```

maeGetEnvOption

```
maeGetEnvOption(
     t_testName
     [ ?includeEmpty g_includeEmpty ]
     [ ?option t_option ]
     [ ?session t_sessionName ]
     )
     => l_optionList / nil
```

Description

Returns a list containing name-value pairs for all the environment option for the given test. By default, it returns all the environment options whose value is not `nil`. If the `includeEmpty` argument is set to `t`, the function returns all the environment options. If a specific option is specified, it returns the value of only that option.

Arguments

t_testName	Name of the test
?includeEmpty *l_includeEmpty*	Includes or excludes the options for which no value is set
?option *t_option*	Name of the option for which the value is to be returned
?session *t_sessionName*	Name of the session Default: Current session

Value Returned

l_optionList	List of name-value pairs of all or the specified environment options
nil	Unsuccessful operation

Example

The following examples show how this function returns the options and their values:

Example 1:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeGetEnvOption("solutions:ampTest:2")
=> (switchViewList ("spectre" "cmos_sch" "cmos.sch" "schematic" "veriloga" "ahdl")
```

```
stopViewList ("spectre")
controlMode "interactive"
modelFiles (("/home/user1/models/Models/allModels.scs" "FF"))
…..
)
```

Example 2:

```
maeGetEnvOption("solutions:ampTest:2" ?includeEmpty t)
=> ( switchViewList ("spectre" "cmos_sch" "cmos.sch" "schematic" "veriloga" "ahdl")
stopViewList ("spectre")
controlMode "interactive"
modelFiles (("/home/user1/models/Models/allModels.scs" "FF"))
paramRangeCheckFile ""
analysisOrder nil
stimulusFile nil
……)
```

Example 3:

```
maeGetEnvOption("solutions:ampTest:2" ?option "switchViewList")
=> ("spectre" "cmos_sch" "cmos.sch" "schematic" "veriloga" "ahdl")
```

Example 4: The following example code shows how to get the list of DSPF files used for a test

```
dspf = maeGetEnvOption("mytestname" ?option "dspfFile")
(printf "dspf = %L\n" dspf)
```

Example 5: To return the complete list of options including the options that are currently not set for all enabled tests in ADE Assembler

```
tests = maeGetSetup(?typeName "tests" ?enabled t)

(foreach test tests
    (printf "test = %s\n" test)
    dspf = maeGetEnvOption(test ?option "dspfFile")
    (printf "dspf = %L\n" dspf)
    )
```

maeGetExplorerTestName

```
maeGetExplorerTestName(
     [ ?session t_sessionName ]
     )
     => t_testName / nil
```

Description

Returns the name of the test opened in ADE Explorer. If you descend into ADE Explorer from ADE Assembler, use this function to get the name of the current test.

Arguments

`?session` *`t_sessionName`*	Name of an ADE Explorer session If not specified, the currently active session is used.

Value Returned

`t_testName`	Name of the currently opened test in ADE Explorer
`nil`	Unsuccessful operation or when the function is used for an ADE Assembler

Examples

Example 1: The following example shows how to use this function to get the name of the test opened in ADE Explorer:

```
maeGetExplorerTestName()
=> "AC"
```

Example 2: If multiple sessions of ADE Explorer are open, you need to explicitly provide the session name to avoid using the current session always. An example is shown below.

```
maeGetSessions()
=> ("fnxSession0" "fnxSession1")
testName = maeGetExplorerTestName(?session "fnxSession0")
=> "AC"
maeGetTestSession(testName ?session "fnxSession0")
=> stdobj@0x2e48fde8
```

Example 3: The following example shows how to get variable details for a test in ADE Explorer:

```
maeGetSessions()
=> ("fnxSession0" "fnxSession1")
testName = maeGetExplorerTestName(?session "fnxSession0")
=> "AC"

testId = axlGetTest(sdb testName)
;;  Find the total sweeps in variables
varList = axlGetVars(testId)
foreach(var cadr(varList)
   varId = axlGetVar(testId var)
   when(axlGetEnabled(varId)
    ;; Get the total number of sweep points in the desired variable
      if(rexMatchp(":" axlGetVarValue(varId)) then
         sweepList=mapcar('evalstring parseString(axlGetVarValue(varId) ":"))
         varSweepVal= int((car(last(sweepList))-car(sweepList))/cadr(sweepList))
      else
         varSweepVal = length(parseString(axlGetVarValue(varId)))
      )
      sweepVal=sweepVal*varSweepVal
      );when
);foreach varList
```

maeGetJobPolicy

```
maeGetJobPolicy(
     [ ?session t_sessionName ]
     [ ?testName t_testName ]
     [ ?jobType t_jobType ]
     )
     => l_jobPolicyProperties / nil
```

Description

Returns the details of the job policy attached to the given test. If no test name is given, the function returns the policy attached to the current setup.

Arguments

`?session` *`t_sessionName`*

Name of an ADE Assembler session

If not specified, the currently active session is used.

`?testName` *`t_testName`*

Name of the test

If not specified, the job policy attached to the currently active setup is returned.

`?jobType` *`t_jobType`*

Job type for which you want to get policy details.

Possible values:

- `"simulation"`: Jobs that run simulations
- `"netlisting"`: Jobs that create netlists

Value Returned

`l_jobPolicyProperties`	A disembodied property list (DPL) of job policy properties
`nil`	Unsuccessful operation

Examples

Example 1:

The following example shows how to use this function to get the default global job policy for the default job control mode, ICRP:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeGetJobPolicy()
=>'( nil configuretimeout "300" distributionmethod "Local" maxjobs "1" runtimeout
"3600" starttimeout "300" )
```

Example 2:

The following example shows how to use this function to get the job policies set for the simulation and netlistng jobs run for the LSCS job control mode, and make the required modification:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeGetJobControlMode()
=> "ICRP"

maeSetJobControlMode("LSCS")
=> t
; sets the job control mode
;the following function returns the policy for simulation jobs
jpnc = maeGetJobPolicy(?jobType "netlisting")
=> (nil autoresume "0" blockemail "1"
configuretimeout "300" defaultcpuvalue "1" defaultmemoryvalue "1000"
distributionmethod "Local" estimatememoryvalue "" estimationsimulationmode "0"
lingertimeout "300" maxjobs "4" name "Job Policy LPF" preemptivestart "1"
providecpuandmemorydata "1" reconfigureimmediately "0" runpointsvalue "5"
runtimeout "-1" scaleestimatedbycpu "100" scaleestimatedbymemory "100"
showerrorwhenretrying "1" showoutputlogerror "0" startmaxjobsimmed "1"
starttimeout "300" suspenddisklow "0" thresholdvalue "100" usesameprocess "1"
warndisklow "0" warnthresholdvalue "100" )


;the following function returns the policy for simulation jobs
jp = maeGetJobPolicy(?jobType "simulation")
=> (nil autoresume "0" blockemail "1" configuretimeout "300" defaultcpuvalue "1"
defaultmemoryvalue "1000" distributionmethod "Local" estimatememoryvalue ""
estimationsimulationmode "0" lingertimeout "300" maxjobs "2" name "Maestro Default"
preemptivestart "1" providecpuandmemorydata "1" reconfigureimmediately "0"
runpointsvalue "5" runtimeout "-1" scaleestimatedbycpu "100"
scaleestimatedbymemory "100" showerrorwhenretrying "1" showoutputlogerror "0"
startmaxjobsimmed "1" starttimeout "300" suspenddisklow "0" thresholdvalue "100"
usesameprocess "1" warndisklow "0" warnthresholdvalue "100" )

;change Max Jobs for simulation jobs
jp->maxjobs=10
=>10

maeSetJobPolicy(jp ?jobType "simulation")
=> t
jp = maeGetJobPolicy(?jobType "simulation")
```

```
=> (nil autoresume "0" blockemail "1" configuretimeout "300" defaultcpuvalue "1"
defaultmemoryvalue "1000" distributionmethod "Local" estimatememoryvalue ""
estimationsimulationmode "0" lingertimeout "300" maxjobs "10" name "Maestro
Default" preemptivestart "1" providecpuandmemorydata "1" reconfigureimmediately
"0" runpointsvalue "5" runtimeout "-1" scaleestimatedbycpu "100"
scaleestimatedbymemory "100" showerrorwhenretrying "1" showoutputlogerror "0"
startmaxjobsimmed "1" starttimeout "300" suspenddisklow "0" thresholdvalue "100"
usesameprocess "1" warndisklow "0" warnthresholdvalue "100" )
```

Example 3:

The following example shows how to get the job policy for a particular test in the setup:

```
maeGetJobPolicy(?testName "test1")
```

```
=>'( nil configuretimeout "100" distributionmethod "Local" maxjobs "1" runtimeout
"3600" starttimeout "300" )
```

maeGetJobPolicyByName

```
maeGetJobPolicyByName(
     t_policyName
     [ ?session t_sessionName ]
     )
     => l_jobPolicyProperties / nil
```

Description

Returns a disembodied property list containing property-value pairs for the given job policy.

Arguments

t_policyName	Name of the job policy
?session *t_sessionName*	Name of an ADE Assembler session If not specified, the currently active session is used.

Value Returned

l_jobPolicyProperties	A disembodied property list (DPL) of job policy properties
nil	Specified job policy is not found

Example

The following example show how to use this function to get the details of a specific job policy:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"

maeGetJobPolicyByName("myPolicy")
=>'( nil configuretimeout "300" distributionmethod "Local" maxjobs "1" runtimeout
"3600" starttimeout "300" )
```

maeGetRunPlan

```
maeGetRunPlan(
     [ ?session t_sessionName ]
     [ ?hitoryName t_historyName ]
     )
     => l_runPlanNames / nil
```

Description

Returns a list of all the runs available in the run plan for the given session or history.

Note: This function is applicable only for ADE Assembler.

Arguments

`?session` *`t_sessionName`*

Name of an ADE Assembler session.

If not specified, the currently active session is used.

`?historyName` *`t_historyName`*

Name of this history for which you want to get the names of child histories corresponding to each run in the run plan.

Value Returned

`l_runPlanNames`	List of the names of run plans
`nil`	Unsuccessful operation

Example

Example 1: The following code returns the names of runs for the current session:

```
maeGetRunPlan()
=> ("Run.0" "Run.1")
```

Example 2: The following code returns the names of runs for the given session:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeGetRunPlan(?session sess1)
```

```
=> ("Run.0" "Run.1" "Run.2")
```

Example 3: The following code returns the names of runs for the given run plan history:

```
maeGetRunPlan(?historyName "Plan.0")
=> ("Plan.0.Run.0" "Plan.0.Run.1")
```

maeGetSetup

```
maeGetSetup(
     [ ?typeName t_typeName ]
     [ ?enabled g_enabled ]
     [ ?session t_sessionName ]
     )
     => l_setupDetails / nil
```

Description

Returns the required setup details of variables, parameters, corners, and tests from the given session.

Arguments

`?typeName` *`t_typeName`*	Type for which data has to be returned. Possible values: `tests`, `corners`, `variables`, or `parameters`. Default value: `tests`
`?enabled` *`g_enabled`*	Status of variables to be returned Possible values: ■ `t`: Enabled ■ `nil`: Disabled ■ `'all`: Both Enabled and disabled Default value: `'all`
`?session` *`t_sessionName`*	Name of the session. If not specified, the currently active session is used.

Value Returned

l_setupDetails	List containing the names of variables, parameters, corners, and tests.
nil	Unsuccessful operation.

Example

The following example returns the names of all the tests in the current session:

```
maeGetSetup()
```

The following example returns the names of all the enabled tests in the current session:

```
maeGetSetup(?enabled t)
=> ("AC" "TRAN")
```

The following example returns the names of all the enabled corners in the current session:

```
maeGetSetup(?typeName "corners" ?enabled t)
=> ("Nominal" "c2")
```

The following example returns the names of all the enabled variables:

```
maeGetSetup(?typeName "variables" ?enabled t)
=> ("gain" "vcm" "vdd")
```

maeGetSimOption

```
maeGetSimOption(
     t_testName
     [ ?option t_optionName ]
     [ ?includeEmpty g_includeEmpty ]
     [ ?session t_sessionName ]
     )
     => l_options / nil
```

Description

Returns a list containing name-value pairs for the simulator options for the given test. By default, it returns all the simulator options whose value is not `nil`. If the *`includeEmpty`* argument is set to `t`, it returns all the simulator options. If a specific option is specified, the function returns the value of only that option.

Arguments

`t_testName`	Name of the test
`?option` *`t_option`*	Name of the simulator option for which the value is to be returned
`?includeEmpty` *`g_includeEmpty`*	Enables or disables return of options for which value is not set Possible values: `t`, `nil` Default value: `t`
`?session` *`t_sessionName`*	Name of the session Default: Current session

Value Returned

`l_options`	List of name-value pairs of simulator options
`nil`	Unsuccessful operation

Example

The following example returns the values of all the options for test `solutions:ampTest`:

```
maeGetSimOption("solutions:ampTest:2" ?includeEmpty t)
=> (label1: "" reltol: "1e-3" residualtol: "" vabstol: "1e-6" iabstol: "1e-12" )
```

maeGetSessions

```
maeGetSessions()
     => l_sessionNames / nil
```

Description

Returns a list of valid ADE Explorer or ADE Assembler sessions that are currently open.

Arguments

None

Value Returned

l_sessionNames	List of valid ADE Explorer or ADE Assembler sessions that are currently open.
nil	Unsuccessful operation.

Example

The following example shows how to use this function:

maeGetSessions()

```
=> ("fnxSession0" "fnxSession1")
```

```
; you can use the session names returned by this function in other functions, such
as maeSaveSetup, or maeGetSetup.
```

maeGetTestEnvVar

```
maeGetTestEnvVar(
     t_testName
     t_varName
     [ ?session t_sessionName ]
     )
     => g_value / nil
```

Description

Returns the value of the specified environment variable for the given test. This function can be used to get the value set for a particular test using the maeSetTestEnvVar function.

Arguments

t_testName	Name of the test
t_varName	Name of the environment variable for which the value is to be returned
?session *t_sessionName*	Name of the ADE XL session Default: Current session

Value Returned

g_value	Value of the environment variable set for the given test
nil	Unsuccessful operation

Example

The following example code shows how to use this function to get the values set for the *PinCheck Term Mismatch Action* and *PinCheck Term Direction Mismatch Action* environment options for the tests in ADE Assembler:

```
;; Loading setup
sess=maeOpenSetup("opamp090" "full_diff_opamp_AC" "maestro11" ?mode "a")
=> "fnxSession0"
testnames=maeGetSetup()
> ("AC" "TRAN")
maeGetTestEnvVar("AC" "termDirectionMismatch" ?session "fnxSession0")
>"warning"
```

```
maeGetTestEnvVar("TRAN" "termMismatch" )
=>"ignore"
```

maeGetTestSession

```
maeGetTestSession (
     t_testName
     [ ?session t_sessionName ]
     )
     => testSession / nil
```

Description

Returns session handle to the given test. You can use this handle to view or modify the test details.

Arguments

t_testName	Name of the test.
?session *t_sessionName*	Name of the ADE XL session. Default: Current session.

Value Returned

testSession	Handle to the test. **Note:** The handle returned by this function can be used as a session argument to SKILL functions prefixed with `asi`, as shown in the example below.
nil	Unsuccessful operation

Example

```
;; Loading setup
sess=maeOpenSetup("opamp090" "full_diff_opamp_AC" "maestro11" ?mode "a")
=> "session0"
testnames=maeGetSetup()
> ("AC" "TRAN")
>
testHandle=maeGetTestSession(car(testnames))
=>stdobj@0x21989b60

asiAddModelLibSelection(testHandle "../models/spectre/gpdk090.scs" "NN")
=> t
>
maeRunSimulation()
```

```
=> "Interactive.1"
maeWaitUntilDone('All)
=> t
maeGetSimulationMessages(?msgType "ERROR")

maeExportOutputView()
=> "/servers/scratch02/testcases/adexl_workshop/design/Interactive.1.csv"
exit()
```

maeGetVar

```
maeGetVar (
     t_varname
     [ ?typeName t_typeName ]
     [ ?typeValue t_typeValue ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Returns value of the given variable.

Arguments

t_varname	Name of the variable.
?typeName *t_typeName*	Type of the variable. Possible values: `test`, `corner` Default value: `test`
?typeValue *t_typeValue*	Name of the corner or test for which value has to be returned. Default: `"Global"`. **Note:** No value will be returned if the type name is `corner` and the type value is `Global`.
?session *t_sessionName*	Name of the session. Default: Current session.

Value Returned

`t_value`	Value of the given variable.
`nil`	When the given corner, test, or variable is not found in the setup.

Example

```
sess=maeOpenSetup("opamp090" "full_diff_opamp_AC" "maestro11" ?mode "a")
=> "session0"
```

maeGetVar("vdd")
=> 2.2

Returns the value of global Variable `Cap`.

```
maeGetVar("Cap" ?typeName "test" ?typeValue "test1")
```

Returns the value of variable `Cap` for "test1".

```
maeGetVar("Cap" ?typeName "corner" ?typeValue "C1")
```

Returns the value of variable `Cap` for `C1` corner.

maeImportSetupForExplorer

```
maeImportSetupForExplorer(
     t_session
     )
     => t / nil
```

Description

Displays the Import Setup form in ADE Explorer, which you can use to import a saved `maestro` cellview in the current setup. It is a callback function.

Arguments

t_session	The current ADE Explorer session.

Values Returned

`t`	Returns `t` when the form is successfully displayed.
`nil`	Returns `nil` if there is an error.

Example

```
maeImportSetupForExplorer(session)
```

where, session is the currently active ADE Explorer session.

maeIsSetupModified

```
maeIsSetupModified(
     [ ?session t_sessionName ]
     )
     => t /nil
```

Description

Checks whether the setup has been modified after the last time it was saved in the given maestro session.

Arguments

t_sessionName	Name of the session Default: By default, the function uses the current maestro session.

Value Returned

`t`	Returns `t` if the set up of the specified maestro has been modified after the last time it was saved.
`nil`	Returns `nil` if the set up of the specified maestro has not been modified after the last time it was saved.

Example

```
sess= maeOpenSetup("solution" "ampTest" "maestro")
maeSetVar("Cap" "2p")
maeIsSetupModified(?session sess)
=> t
```

maeIsSingleTest

```
maeIsSingleTest(
     t_libraryName
     t_cellName
     t_viewName
     )
     => t /nil
```

Description

Returns `t` if the given cellview contains a single test or multiple tests.

Arguments

t_libName	Name of a library
t_cellName	Name of a cell in the given library
t_viewName	Name of a maestro cellview

Value Returned

`t`	Returns `t` if the specified maestro view contains a single test. This implies that it can be opened in ADE Explorer.
`nil`	Returns `nil` if the specified maestro view contains multiple tests. This implies that it can be opened in ADE Assembler.

Example

```
maeIsSingleTest("solutions" "ampTest" "maestro")
=> t
```

maeIsValidMaestroSession

```
maeIsValidMaestroSession(
     t_sessionName
     )
     => t /nil
```

Description

Confirms if the given session is a valid session for ADE Explorer or ADE Assembler.

Arguments

t_sessionName	Session name to be validated.

Value Returned

t	The specified session is a valid ADE Explorer or ADE Assembler session.
nil	If the specified session is not valid.

Example

```
maeIsValidMaestroSession("fnxSession0")
=> t
```

maeLoadCorners

```
maeLoadCorners(
     t_fileName
     ?operation t_operation
     ?session t_session
     )
     => t /nil
```

Description

Loads corners from the given file into the corner setup of the current session or the specified session.

Arguments

t_fileName	Name of the file from which the corners are to be loaded
?operation *t_operation*	Operation to be performed while loading the corners. Possible values: ■ `"deleteallncreatenew"`: Deletes all existing corners and loads new corners. ■ `"overwrite"`: If the name of the corner being loaded is the same as that of an existing corner, the existing corner is removed and the new one is added. ■ `"rename"`: If the name of the corner being loaded is the same as that of an existing corner, retains the existing corner and renames the new corner being loaded. ■ `"cancel"`: If the name of the corner being loaded is the same as that of an existing corner, cancels the loading process. No corner is loaded. Default value: `"overwrite"`
?session *t_session*	Session name

Value Returned

`t`	When the corners are successfully loaded from the source file.
`nil`	When the corners are not loaded.

Example

```
sess=maeOpenSetup("opamp090" "full_diff_opamp_AC" "maestro11" ?mode "a")
=> "session0"

maeLoadCorners("corners.sdb" ?operation "rename")

=> t

;Load corners from the file "corners.sdb" after changing the conflicting corner
;names.
```

maeOpenSetup

```
maeOpenSetup(
     t_libName
     t_cellName
     t_viewName
     [ ?application t_applicationName ]
     [ ?histName t_historyName ]
     [ ?mode t_mode ]
     )
     => t_sessionName / nil
```

Description

Loads the given cellview and restores the setup details from the specified history. If no history name is specified, the active setup is loaded. If the specified view is already opened in the current Virtuoso session, it is not opened again. However, if the view is already open in some other Virtuoso session, it is opened in read mode in the current session. If the given cellview does not exist, the function creates a new cellview with the same name.

Arguments

t_libName Name of the library

t_cellName Name of the cell

t_viewName Name of the view

?application *t_applicationName*

Specified whether to open the setup in ADE Assembler.

Set this argument to "Assembler" when you need to open a single test environment in ADE Assembler to use the features specific to this product.

Note: This argument is not required to open a test in ADE Explorer.

?histName *t_historyName*

Name of the history to be restored. If not given, the active setup is loaded.

`?mode` *`t_mode`*	Mode in which the maestro view is to be opened Possible values: ■ `"a"`, for append mode ■ `"r"`, for read mode Default value: `"a"` **Note:** You cannot create a new view in read mode. If you are creating a new view, set this argument to `"a"`.

Value Returned

`t_sessionName`	The setup is successfully loaded
`nil`	Unsuccessful operation

Example

```
maeOpenSetup("solutions" "amptest" "maestro")
```

Loads the active setup of solutions/amptest/maestro in append mode.

```
maeOpenSetup("solutions" "amptest" "maestro" ?application "Assembler")
```

Loads the active setup of solutions/amptest/maestro in append mode in ADE Assembler.

```
maeOpenSetup("solutions" "amptest" "maestro" ?histName "Interactive.5")
```

Loads the setup of “Interactive.5” history for solutions/amptest/maestro in append mode.

```
maeOpenSetup("solutions" "amptest" "maestro" ?histname "Interactive.5" ?mode "r")
```

Loads the setup of `Interactive.5` history for `solutions/amptest/maestro` in read only mode.

maeLoadSetupState

```
maeLoadSetupState(
     t_stateName
     [ ?tags l_tagNames ]
     [ ?operation s_operationName ]
     [ ?session t_sessionName ])
     )
     => t_sessionName / nil
```

Description

Loads the given setup state into the given session.

Arguments

t_stateName	Name of the state to be loaded.
l_tagNames	List of tags that defines the components to be copied from the loaded state. Possible values: `tests` - Testbench setups `vars` - Global variables `parameters` - Parameters and their values `currentMode` - Run mode `runOptions` - Simulation options for different run modes and the run distribute options `specs` - Parameter specifications `corners` - Corner details `modelGroups` - Model groups `extensions` - Extensions `relxanalysis` - Reliability analysis setup details `All` - Details of all tests, vars, parameters, currentMode, runOptions, specs, corners, modelGroups, extensions, and relxanalysis Default value: `All`

t_viewName	Name of the view.
?operation *s_operationName*	Operation to handle the existing details in the setup. Possible values: ■ `'merge` - If the existing setup details have the same name as that of the details being loaded, they are deleted and overwritten by the setup information copied from the loaded state. ■ `'retain` - If an existing setup detail has the same name as that of the detail being loaded, it is retained and not overwritten by the setup information copied from the loaded state. ■ `'overwrite` - All existing setup details are deleted and all the imported outputs are used
?session *t_sessionName*	Name of the session to which the state has to be loaded. Default value: Current session

Value Returned

`t`	Successfully loaded the given setup state.
`nil`	Unsuccessful operation.

Example

The following example code loads the corners from state state1 and merges those with the current corner details:

```
maeLoadSetupState("state1" ?tags list("corners") 'merge)
```

maeLoadStateForTest

```
maeLoadStateForTest(
     t_testName
     t_stateName
     [ ?session t_session ]
     [ ?loadFrom s_loadFrom ]
     [ ?statePath s_statePath ]
     [ ?libName t_libName ]
     [ ?cellName t_cellName ]
     [ ?simulator t_simulatorName ]
     [ ?component l_componentList ]
     )
     => t_sessionName / nil
```

Description

Loads the saved ADE state for the given test.

Arguments

t_testName	Name of the test in maestro setup for which the state is to be loaded.
t_stateName	Name of the state to be loaded for the given test. You can provide the name of a state that was saved using ADE L or ADE XL.
?session *t_session*	Name of the session to which the ADE state is to be loaded. Default: Current Session.
?loadFrom *s_loadFrom*	Specifies whether state is to be loaded from cellview or directory. Valid values: ■ `'directory` ■ `'cellview` If no value is specified for this argument, the saveAsCellview environment variable is checked. If the variable is set to `t`, state is loaded from the cellview. Otherwise, the state is loaded from the directory.

`?statePath` *`t_statePath`*	State path from where state is to be loaded when `?loadFrom` is set to `'directory`. It is path where `libName/cellName/simulator/stateName` is located. The default path is taken from the saveDir environment variable.
`?libName` *`t_libName`*	Library name for which state is to be loaded. If this is not given libName from test session is taken.
`?cellName` *`t_cellName`*	Cell name for which state is to be loaded. If this is not given cell name from test session is taken.
`?simulator` *`t_simulatorName`*	simulator name for which state is to be loaded. If this is not given simulator name from test session is taken.

`?component` *`l_componentList`*

List of the components to be loaded from the state. When no value is given all components are loaded.

Valid Values:

- `analyses` for Analyses
- `variables` for Variables
- `outputs` for Outputs
- `subckts` for Subcircuit instances
- `opPoints` for Operating Points
- `modelSetup` for Model Setup
- `simulationFiles` for Simulation Files
- `environmentOptions` for Environment Options
- `simulatorOptions` for Simulator Options
- `convergence` for Convergence Setup
- `waveformSetup_ws` for Waveform Setup
- `graphicalStimuli` for Graphical Stimuli
- `conditionsSetup` for Conditions Setup
- `printSetup` for Results Display Setup
- `devCheckingSetup` for Device Checking Setup
- `relxOptions` for RelXpert Setup
- `cosimOptions` for Cosimulation Options
- `turboOptions` for Performance/Parasitic Reduction
- `mdlOptions` for MDL Control Setup
- `dpSetup` for Distributed Processing
- `paramSetup` for Parameterization Setup
- `all` for all types of components listed above. This is the default value.

Value Returned

t_sessionName	Successfully loaded the given state.
nil	Unsuccessful operation.

Example

```
maeLoadStateForTest("test1" "spectre_state1" ?loadFrom 'cellView ?statePath
"/home/TEST/TRAINING/training/ampTest/adexl/test_states" ?libName "training"
?cellName "ampTest" ?simulator "spectre" ?component '(environmentOptions
simulationFiles))
> t
```

maeSaveSetup

```
maeSaveSetup(
     [ ?lib t_libName ]
     [ ?cell t_cellName ]
     [ ?view t_viewName ]
     [ ?session t_sessionName ])
     => t / nil
```

Description

Saves the setup database file and test state files for the current session in the library, cell, view format. The behavior of this function is similar to File - Save. By default, the setup of the current cellview is saved. If the cellview was opened in the read-only mode, a new library, cell, and view value must be provided for which you need to save the setup details.

Arguments

`?lib` *`t_libName`*	Name of the library.
`?cell` *`t_cellName`*	Name of the cell.
`?view` *`t_viewName`*	Name of the view.
`?session` *`t_sessionName`*	Name of the session. Default: Current session

Value Returned

`t`	Successfully saved the setup.
`nil`	Unsuccessful operation.

Example

```
maeSaveSetup()
```

Saves the current setup.

```
maeSaveSetup(?lib "solutions" ?cell "amptest" ?view "maestro")
```

Saves the current setup as `solutions/amptest/maestro/maestro.sdb`.

maeStmConsolidateStimuli

```
maeStmConsolidateStimuli(
     t_libName
     t_cellName
     t_viewName
     ?reportFile t_reportFile
     )
     => t / nil
```

Description

Removes duplicate stimuli definitions from the given cellview. For this, ADE Explorer or ADE Assembler reviews each test in the cellview and if required, reassigns pins and globals by removing duplicates.

Arguments

t_libName	Name of the library.
t_cellName	Name of the cell.
t_viewName	Name of the view.
?reportFile *t_reportFile*	Prints a report of changes in the specified report file.

Value Returned

t	Successfully saved the setup.
nil	Unsuccessful operation.

Example

Open a maestro cellview and use the following command to consolidated stimuli in the given cellview:

```
maeStmConsolidateStimuli("testLib" "StimuliTest" "maestro" ?reportFile
"consolidationReport.txt")
=> t
```

maeStmGenerateWaveforms

```
maeStmGenerateWaveforms(
     t_libName
     t_cellName
     t_viewName
     [ ?test t_testName ]
     [ ?reportFile t_reportFile ]
     [ ?force g_force ]
     )
     => t / nil
```

Description

Generates preview stimuli waveforms for the stimuli added for a maestro cellview that contains stimuli definitions added through the Stimuli Assignment form. These stimuli definitions are saved in *<lib-name>/<cell-name>/<view-name>*/namedStimuli/stimuli.xml file. The function examines each of the stimuli definitions for preview waveforms. If a stimuli definition has no associated preview waveform, or the waveform data is out of date, it generates a new SRR waveform data and saves it in the <project-dir>/*<lib-name>/<cell-name>/<view-name>*/namedStimuli directory.

The waveform data is considered out of date if the time stamp of a stimuli XML definition is newer than that of the preview waveform data associated with it.

To view the waveforms generated by this function, open the Stimuli Assignment form, right-click on a stimuli and choose the *Plot Transient Waveform* command. For details, refer Setting Up Stimuli in *Virtuoso ADE Explorer User Guide*.

Arguments

t_libName	Name of the library.
t_cellName	Name of the cell.
t_viewName	Name of the view.
?test *t_testName*	Name of the test to which the stimuli is associated. If no test name is specified, the tool uses the first test in the cellview.
?reportFile *t_reportFile*	

	Name of the report file in which the status message for each stimuli waveform is printed. When no file name is provided, the default file path relative to the current working directory is used. Default value: `"./maeStmGenerateWaveforms.<lib>.<cell>.<view>.<test>.log"`.
`?force` *`g_force`*	Forces the waveform generation for each stimuli. By default, the tool generates waveforms only when the stimuli information does not match with the current waveforms, which could happen in the following scenarios: ■ No waveform exists ■ Values of variables have been changed after waveform generation ■ Value of a stimuli parameter has been changed.

Value Returned

`t`	Preview stimuli waveforms are generated successfully.
`nil`	Preview stimuli waveforms are not generated successfully .

Example

Open the Stimuli Assignment form in a maestro cellview and use the following command to generate stimuli waveforms:

```
maeStmGenerateWaveforms("testLib" "StimuliTest" "maestro")
=> t
```

Open the Stimuli Assignment form in a maestro cellview and use the following command to generate stimuli waveforms for all stimuli and print report in the given file:

```
maeStmGenerateWaveforms("testLib" "StimuliTest" "maestro" ?force t ?reportFile
"./reports/stimuliReport.log")
=> t
```

maeSaveSetupState

```
maeSaveSetupState(
     t_stateName
     [ ?tags l_tagNames ]
     [ ?inReadOnly s_readOnlyAction ]
     [ ?session t_sessionName ])
     => t / nil
```

Description

Saves a setup state for the given session.

Arguments

t_stateName	Name of the state in which you need to save the setup details.
?tags *l_tagNames*	List of tag names that specify the details to be saved. Available tags are: `tests` - Testbench setups `vars` - Global variables `parameters` - Parameters and their values `currentMode` - Run mode `runOptions` - Simulation options for different run modes and the run distribute options `specs` - Parameter specifications `corners` - Corner details `modelGroups` - Model groups `extensions` - Extensions `relxanalysis` - Reliability analysis setup details `All` - Details of all the tests, variables, parameters, current run mode, run options, specs, corners, model groups, extensions, and reliability analysis.

`?inReadOnly` *`s_readOnlyAction`*	Specifies the action to be performed in read only mode. Possible values: ■ `'error`: Displays an error. This is the default value. ■ `'useSaveDir`: Saves the setup state in the save directory. ■ `'useprojectDir`: Saves the setup state in the project directory.
`?session` *`t_sessionName`*	Name of the session. Default: Current session

Value Returned

`t`	Successfully saved the setup.
`nil`	Unsuccessful operation.

Example

The following example code saves the corners and variables from the current session to a state named `state1`:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeSaveSetupState("state1" '("corners" "vars"))
```

maeSetAnalysis

```
maeSetAnalysis(
     t_testName
     t_analysis
     [ ?enable g_enabled ]
     [ ?options l_options ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Adds an analysis or changes the enabled status of the specified analysis for the given test. The function can also be used to set the value of different options for the analysis.

Arguments

t_testName	Name of the test to which you need to add an analysis
t_analysis	Name of the analysis
[?enable *g_enabled*]	Status to be set Possible values: `t`, `nil` Default value: `t`
[?options *l_options*]	List of analyses options and the values to be set **Note:** You can use maeGetAnalysis to get the list of analysis options, as shown below. `maeGetAnalysis(t_testName t_analysisName ?includeEmpty t)`
[?session *t_sessionName*]	Name of the session Default: Current session.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

The following example code adds a new analysis `tran` and sets its options, stop time 500 and step 10.:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeSetAnalysis( "solutions:ampTest:2" "tran" ?enable t ?options '(("stop" "500")
("step" "10")))
```

maeSetCorner

```
maeSetCorner(
     t_cornerName
     [ ?enabled g_enabled ]
     [ ?enableTests l_enableTests ]
     [ ?disableTests l_disableTests ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Adds a new corner and enables/disables it for the given test.

Arguments

t_cornerName	Name of the corner to be added/updated.
?enabled *g_enabled*	Specifies whether the given corner is to be enabled or disabled. This option updates the status of the check box for this corner in the Run Summary assistant.
?enableTests l_enableTests	List of the tests for which the given corner has to be enabled. Default: `ALL
?disableTests l_disableTests	List of the tests for which the given corner has to be disabled. Default: `ALL
?session *t_sessionName*	Name of the session. If not specified, the currently active session is used.

Value Returned

t	The given corner is added or updated
nil	Returns nil if there is an error.

Example

The following example code adds a corner C0 and enables it for all tests:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeSetCorner("C0")
```

The following example code adds a corner C1 and enables it for tests AC and TRAN, but disables the corner in the setup (that is, the check box for this corner in the Run Summary assistant is cleared):

```
maeSetCorner("C1" ?enableTests `("AC", "TRAN") ?enabled nil)
```

The following example code adds a corner C2 and disables it for test1 and test3:

```
maeSetCorner("C2" ?disableTests '("test1" "test3"))
```

maeSetCurrentRunMode

```
maeSetCurrentRunMode(
     [ ?session t_sessionName ]
     t_runModeName
     )
     => t / nil
```

Description

Updates the current run mode in ADE Assembler.

Arguments

`?session` *`t_sessionName`*	Name of the session. If not specified, the currently active session is used.
`t_runModeName`	Name of the corner to be added/updated.

Value Returned

`t`	The run mode is changed.
`nil`	Returns `nil` if there is an error.

Example

The following example sets `Fault Simulation` as the current run mode:

```
maeSetCurrentRunMode(?runMode "Fault Simulation")
=> t
```

maeSetDesign

```
maeSetDesign(
     t_testName
     t_libName
     t_cellName
     t_viewName
     [ ?session t_sessionName ]
     )
     => t_testName / nil
```

Description

Sets a design for the given test. Use this function to change the design associated to a test.

Arguments

t_testName	Name of the test for which you need to change the design.
t_libName	Name of the library of the design to be used.
t_cellName	Name of the cell of the design to be used.
t_viewName	Name of the maestro cellview of the design to be used.
?session *t_sessionName*	Name of the session. If not specified, the currently active session is used.

Value Returned

t_testName	Returns the name of the test for which the design is changed.
nil	Returns nil if there is an error.

Example

The following example shows how to set the design for test test1 to the config view of the ampTest cell in library solutions:

```
maeSetDesign("test1" "solutions" "ampTest" config")
=>"test1"
```

maeSetEnableTestVar

```
maeSetEnableTestVar(
     t_testName
     l_variableNames
     [ ?enabled g_enabled ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Enables or disables the specified variables or local sweeps for the given test.

Arguments

t_testName	Name of the test
l_variableNames	List of variable names to be enabled or disabled for the given test
?enabled *g_enabled*	Status to be set for the given variables. Possible values: `t`, `nil` Default value: `t`
?session *t_sessionName*	Name of the session. Default: Current session.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

The following example code enables `var1` and `var3` local variables for test `test1`:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeSetEnableTestVar("test1" '("var1" "var3") ?enabled t)
=> t
```

maeSetEnvOption

```
maeSetEnvOption(
     t_testName
     [ ?options l_options ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Sets values for one or more environment options for the given test.

Arguments

t_testName	Name of the test
?options *l_options*	List of environment options and the values to be set for them. You can use maeGetEnvOption to get the list of environment options, as shown below. `maeGetEnvOption(t_testName ?includeEmpty t)` **Note:** This function does not validate the values specified for the environment options. Therefore, you must ensure that the names of options and their respective values are in correct format.
?session *t_sessionName*	Name of the session Default: Current session

Value Returned

t	Successful operation
nil	Unsuccessful operation

Example

Example 1:

The following example code shows how to set values for multiple environment options:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
```

```
maeSetEnvOption("AC" ?options '(("stopViewList" ("spectre")) ("modelFiles"
(("./models.scs" "FF")))))
;; Sets the stopViewList as ("spectre") and modelFiles as (("./models.scs" "FF")
```

Example 2:

The following example shows how to set multiple model files by specifying a list of model file paths:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"

maeSetEnvOption("AC" ?options '(("modelFiles" (("./models.scs" "FF")
("./TECH/GPDK045/gpdk045/models/spectre/gpdk045.scs" "mc")
("./DESIGNS/GPDK045/FRACNPLL/models/inductor.scs")))))
```

maeSetHistoryLock

```
maeSetHistoryLock(
     t_historyName
     g_lock
     [ ?session t_sessionName ])
     => t / nil
```

Description

Locks or unlocks the given history in the given ADE Assembler session.

Arguments

t_historyName	Name of the history to be locked or unlocked
g_lock	Lock status. Specify `t` to lock, `nil` to unlock the given history
`?session` *t_sessionName*	Name of the session. Default: Current session

Value Returned

`t`	Successfully saved the setup.
`nil`	Unsuccessful operation.

Example

```
maeSetHistoryLock("Interactive.1" t)
=> Lock the history "Interactive.1"
```

maeSetParameter

```
maeSetParameter(
     t_parameterName
     g_parameterValue
     [ ?typeName t_typeName ]
     [ ?typeValue l_typeValue ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Adds a new parameter at the global level or corner level. If the parameter already exists, updates its value.

Arguments

t_parameterName	Name of the parameter to be added or updated. Parameter names must contain at least five non-empty names each separated by a `/` (slash), typically representing the `library/cell/view/instance/property` to be modified. Multiple `"instance/"` strings can be specified to specify hierarchical parameters, such as `library/cell/view/instance0/instance1/instance2/property`.
g_parameterValue	Value to the set for the parameter
`?typeName` *t_typeName*	Type of the parameter that specifies if the parameter is to be added or updated at the global level or only for corners. Possible values: `"test" "corner"` Default value: `"test"`
`?typeValue` *l_typeValue*	List of the corners for which the parameter has to be added. **Note:** If `?typeName` is set to `test`, value given for this argument is ignored. The parameter is added at the global level. Default: `"Global"`
`?session` *t_sessionName*	Name of the session. Default: Current session

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

The following example code adds a parameter at the test level and assigns a value to it:

```
sess= maeOpenSetup("Two_Stage_Opamp" "OpAmp" "maestro")
=> "session0"
maeSetParameter("Two_Stage_Opamp/OpAmp/schematic/R0/r" "8k")
=> t
; the following command sets 6k for the same parameter for corners C0 and C1
maeSetParameter("Two_Stage_Opamp/OpAmp/schematic/R0/r" "6k" ?typeName "corner"
?typeValue '("C0" "C1"))
=> t
```

maeSetSpec

```
maeSetSpec (
     t_outputName
     t_testName
     [ ?minimum g_minValue ]
     [ ?maximum g_maxValue ]
     [ ?gt g_greaterThanValue ]
     [ ?lt g_lessThanValue ]
     [ ?range g_rangeValues ]
     [ ?tolerance g_toleranceValue ]
     [ ?info g_info ]
     [ ?weight g_weightingFactor ]
     [ ?corner g_cornerName ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Adds a specification to an output defined for a test. You can also use this function to modify an existing specification for an output.

Arguments

t_outputName	Name of the output for which the specification is to be set
t_testName	Name of the test to which the output is associated
?minimum *g_minValue*	Value for the min spec.
?maximum *g_maxValue*	Value for the max spec.
?gt *g_greaterThanValue*	Value for the greater than spec.
?lt *g_lessThanValue*	Value for the less than spec.
?range *g_rangeValues*	A range of values for the range spec.
?tolerance *g_toleranceValue*	Value for the tolerance spec.

`?info` *`g_info`*	Any information string for info spec.
`?weight` *`g_weightingFactor`*	A weighting factor for this spec.
`?corner` *`g_cornerName`*	Name of the corner for which the spec is to be enabled. This argument helps to override a specification for a particular corner. By default, a specification defined for a measurement applies to all the corners enabled for the test. To change the specification for a particular corner, specify the name of that corner in this argument. In ADE Assembler, you can view the overridden corner name in the Override Specifications form. **Note:** This argument is considered only for ADE Assembler sessions because ADE explorer does nor support overridden specifications.
`?session` *`t_sessionName`*	Name of the session. Default: Current session

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

The following example code adds the `>1` spec to output `out1` in test `TRAN`:

```
sess= maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeSetSpec("out1" "TRAN" ?gt "1")
=> t
```

maeSetJobPolicy

```
maeSetJobPolicy (
     g_jobPolicyDPL
     [ ?testName t_testName ]
     [ ?jobType t_jobType ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Sets the given job policy to the specified setup for the given test.

Arguments

g_jobPolicyDPL	A distributed property list that can provide values for multiple properties of a job policy.
?testName *t_testName*	Name of the test for which this policy needs to be set. Default: `"Global"`
?jobType *t_jobType*	Type of job for which the policy is to be set. Possible values: ■ `"simulation"`: (Default) Sets the job policy for the simulation jobs ■ `"netlisting"`: Sets the job policy for the netlisting jobs
?session *t_sessionName*	Name of the session. Default: Current session.

Value Returned

t	Successful operation
nil	Unsuccessful operation

Examples

The following example updates the default global job policy in the current maestro session to change the value of the pointlifetime property:

```
Jp = maeGetJobPolicy()
Jp->pointlifetime=1
maeSetJobPolicy(jp)
```

The following example updates the netlisting job policy for test test1:

```
Jp = maeGetJobPolicy(?jobType "netlisting" ?testName "test1")
Jp->pointlifetime=1
maeSetJobPolicy(jp)
```

maeSetSetup

```
maeSetSetup(
     [ ?tests l_testNames ]
     [ ?variables l_variables ]
     [ ?parameters l_params ]
     [ ?corners l_corners ]
     [ ?enabled g_enabled ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Enables or disables the tests, global variables, parameters, and corners.

Arguments

`?tests` *`l_testNames`*	List of the tests which are to be enabled or disabled.
`?variables` *`l_variables`*	List of the global variables to be enabled or disabled.
`?parameters` *`l_params`*	List of the parameters to be enabled or disabled.
`?corners` *`l_corners`*	List of the corners to be enabled or disabled. To enable or disable the Nominal corner, add `Nominal` or `nominal` to the list. **Note:** If all other corners are disabled, the function keeps the Nominal corner enabled.
`?enabled` *`g_enabled`*	Enabled or disabled status. Possible values: `t` , `nil` Default value: `t`
`?session` *`t_sessionName`*	Name of the session. Default: Current session.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Examples

The following example code enables tests, corners, and variables:

```
; load the setup
maeOpenSetup("solutions" "ampTest" "maestro") > t

maeSetSetup(?tests '("test1" "test3"))
; enables test1 and test3.

maeSetSetup(?tests '("test2") ?enabled nil)
; Disable test2

maeSetSetup(?corners '("nominal" "C1") ?variables '("VDD"  "IREF")?enabled nil)
; Disables the nominal and C1 corners, and the VDD and IREF global variables
```

maeSetSimOption

```
maeSetSimOption(
     t_testName
     [ ?options l_options ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Sets values for the specified simulator options of the given test. Multiple options can be specified in a single command.

Arguments

t_testName	Name of the test
?options *l_options*	List of name-value pairs of the simulator options to be set **Note:** You can use maeGetSimOption to get the list of simulator options as shown below. `maeGetSimOption(t_testName ?includeEmpty t)`
?session *t_sessionName*	Name of the session Default: Current session

Value Returned

t	Successful operation
nil	Unsuccessful operation

Examples

The following example code sets the simulator option `temp` as `27` and `maxwarns` as `10` for test `solutions:ampTest:2`:

```
maeSetSimOption("solutions:ampTest:2" ?options '(("temp" "27") ("maxwarns" "10")))
=> t
```

maeSetTestEnvVar

```
maeSetTestEnvVar(
     t_testName
     t_varName
     g_varValue
     [ ?session t_sessionName ]
     )
     => t_value / nil
```

Description

Sets the value for the given environment variable at the test level. The value is used only by the specified test. Other tests in the session use the value set at the global level or specific values set for them, if any.

Arguments

t_testName	Name of the test for which you need to set the value for an environment variable
t_varName	Name of the environment variable
g_varValue	Valid value for the environment variable
?session *t_sessionName*	Name of the session Default: Current session

Value Returned

t_value	When the value of the environment variable is set successfully for the given test
nil	When the value of the environment variable is not set for the given test The function also returns an error if the given value is invalid.

Examples

The following example code sets the values for *PinCheck Term Mismatch Action* and *PinCheck Term Direction Mismatch Action* environment options for test `AC` in ADE Assembler:

```
; load the setup
sess=maeOpenSetup("opamp090" "full_diff_opamp_AC" "maestro11" ?mode "a")
=> "fnxSession0"
testnames=maeGetSetup()
> ("AC" "TRAN")

maeSetTestEnvVar( "AC" "termDirectionMismatch" "ignore" ?session "fnxSession0")
=>"ignore"
```

maeSetVar

```
maeSetVar(
     t_varname
     g_valValue
     [ ?typeName t_typeName ]
     [ ?typeValue l_typeValue ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Adds a variable to the given test or corner in the given ADE Explorer or ADE Assembler session. If the variable already exists, its value is updated.

Arguments

t_varnam	Name of the variable.
g_valValue	Value of the variable.
?typeName *t_typeName*	Type of the variable. Possible values: `test`, `corner` Default value: `test`
?typeValue *t_typeValue*	List of the corners or tests for which variable has to be added. Default: `Global`
?session *t_sessionName*	Name of the Explorer or Assembler session in which you want to make these changes. Default: Current session.

Values Returned

`t`	Successful addition/update.
`nil`	Unsuccessful operation.

Examples

Example 1

The following example statement shows how to set value for a global variable, `Cap`:

```
maeSetVar("Cap" "5p")
```

Example 2

The following example statement shows how to set value for variable `Cap` in test `AC`:

```
maeSetVar("Cap" "5p" ?typeName "test" ?typeValue '("AC"))
```

Example 3

The following example statement shows how to set value for variable `Cap` in corners `C1`, `C2`, and `C3`:

```
maeSetVar("Cap" "5p" ?typeName "corner" ?typeValue '("C1" "C2" "C3"))
```

Example 4

The following example code shows how to get the value of a variable from a previous run and update it for the next run:

```
Res=maeOpenResults(?run "vco_char")
kvco_val=maeGetOutputValue("kvo" "vco_char" ?cornerName "nominal")
maeCloseResults(Res)
maeSetVar("KVCO" kvco_val)
;; This example script can be helpful in run plans.
```

maeUpdateImplicitSignals

```
maeUpdateImplicitSignals(
     o_session
     [?test t_name]
     )
     => t / nil
```

Description

Updates the implicit signals for the tests in the specified maestro session.

Arguments

o_session	Maestro session object.
?test t_name	Name of the test for which implicit signal outputs are updated.

Value Returned

t	Returns t when function call is successful.
nil	Returns nil when function call is unsuccessful.

Example

```
maeUpdateImplicitSignals( maeSession ?testName test1)
```

It generates implicit signals for the expressions in the maestro setup, if the implicit signals do not already exist. It also clears the implicit signals which may have existed but are no longer used in any existing expression.

Example script

The following example shows how to view and edit the setup details:

```
; load the setup
maeOpenSetup("solutions" "ampTest" "maestro") > t

; check which tests are there in setup
maeGetSetup()
=> ("solutions"amptest:1" "solutions:amptest:2")

; Enable a test "solution:amptest:1"
maeSetSetup(?tests '("solutions:amptest:1"))

; Set the value of global variable
"CAP" as 5p: maeSetVar("CAP" 5p)

; Disable corners C0 and C1
maeSetSetup(?corners '("C0"  "C1") ?enabled nil)

; Run Simulation
maeRunSimulation()
```

Functions to Run Simulations

SKILL functions to run and manage simulations

Function	Description
maeGetSimulationMessages	Displays the error messages of the given type thrown during the simulating run.
maeGetMappingForJobAndPoint	Returns a list containing the mapping of job IDs to the point IDs allocated to them for all the simulations run in the current ADE Explorer or ADE Assembler session. After starting a simulation run, you can use this information to debug incomplete simulations.
maeOpenLogViewer	Opens the Log Viewer window where you can view the messages loaded from a database.
maeResumeSimulation	Resumes the simulations that were earlier suspended automatically by ADE Explorer or ADE Assembler according to the settings specified on the Resources tab of the Job Policy Setup form.
maeRunSimulation	Sets the given run mode for the given session and runs simulation.
maeSetPreRunScript	Sets the given script as a pre-run script for the given test and sets its status.
maeSetRunOption	Sets value for a run option.
maeStopSimulation	Stops simulation runs for the given histories.
maeSuspendSimulation	Suspends the simulation run for the specified maestro session. If not specified, the current simulation run is suspended.
maeWaitUntilDone	Specifies the names of history checkpoints for which the tool must wait before proceeding further.
maeWriteScript	Creates a script with the specified setup details. This script can be run from the command line.

maeGetSimulationMessages

```
maeGetSimulationMessages(
     [ ?session t_session ]
     [ ?msgType t_messageType ]
     )
     => t_messages / nil
```

Description

Displays the error messages of the given type thrown during the simulating run.

Arguments

`?session` *`t_session`*	Name of the session.
`?msgType` *`t_messageName`*	Type of messages to be displayed. Valid values: ■ `ERROR`. This is the default value. ■ `WARNING` ■ `INFO` ■ `ALL`

Value Returned

`t_messages`	Returns a string of errors for given simulation.
nil	Unsuccessful operation.

Example

The following example script shows how you can print all warning messages from a simulation run:

```
mySession=maeOpenSetup("solutions" "amptest" "maestro")
=> t
>
maeRunSimulation()
=> "Interactive.1"
>
```

```
maeWaitUntilDone('all)
>
```

maeGetSimulationMessages(?session mySession ?msgType "ERROR")

```
=>"ERROR (ADEXL-1611): Sweep and Corners are using the same variables:vddEither
disable variables in sweeps, or disable corners that use these variables.\nERROR
(ADEXL-1742): Cannot run 'Monte Carlo Sampling' with swept variable/parameters.You
are trying to run Monte Carlo with sweeps enabled.  This is currently not supported
in ADE XL.  Disable all sweeps or enable 'Use Reference Point' in the Monte Carlo
options to continue.\n"
>
```

maeGetMappingForJobAndPoint

```
maeGetMappingForJobAndPoint(
     [ ?session t_session ]
     )
     => l_jobPointMapping / nil
```

Description

Returns a list containing the mapping of job IDs to the point IDs allocated to them for all the simulations run in the current ADE Explorer or ADE Assembler session. After starting a simulation run, you can use this information to debug incomplete simulations.

Arguments

`?session` *`t_session`*	Name of the session.

Value Returned

`l_jobPointMapping`	Returns a list of job ID and point ID mapping.
nil	Unsuccessful operation.

Example

The following example shows the result of a simulation run on an LSF farm. The max jobs value in the job policy is 4.

```
maeGetMappingForJobAndPoint()
=>((1 "1 ") (2 "2 3 ") (4 "2 ") )
```

maeOpenLogViewer

```
maeOpenLogViewer(
     [ t_dbFilePath ]
     )
     => t / nil
```

Description

Opens the Log Viewer window where you can view the messages loaded from a database.

Arguments

t_dbFilePath	Path to the database file in which messages are saved.

Value Returned

`t`	Successfully opens the Log Viewer window.
`nil`	Unsuccessful operation.

Example

The following example code opens the Log Viewer window and loads the given database:

```
maeOpenLogViewer("/home/example/Interactive.0.msg.db")
=> t
```

maeResumeSimulation

```
maeResumeSimulation(
     [ ?session t_session ]
     )
     => t / nil
```

Description

Resumes the simulations that were earlier suspended automatically by ADE Explorer or ADE Assembler according to the settings specified on the *Resources* tab of the Job Policy Setup form.

For more details about those options, see Automatically Suspend Simulations When the Disk Space is Low.

Arguments

`?session` *`t_session`*	Name of the session. If not specified, the current session is used.

Value Returned

`t`	The simulation is successfully resumed.
`nil`	Unsuccessful operation.

Example

The following example script shows how you can print all warning messages from a simulation run:

```
mySession=maeOpenSetup("solutions" "amptest" "maestro")
=> t
>
maeRunSimulation()
=> "Interactive.1"
...
;; assumption that the simulation got suspended due to unavailability of disk space
maeResumeSimulation()
=> t
```

maeRunSimulation

```
maeRunSimulation(
     [ ?session t_sessionName ]
     [ ?runMode t_runMode ]
     [ ?callback t_callback ]
     [ ?run t_runPlan ]
     [ ?waitUntilDone g_waitUntilDone ]
     [ ?returnRunId g_returnRunId ]
     )
     => t_histname / x_runID / nil
```

Description

Sets the given run mode for the given session and runs simulation.

Arguments

`?session` *`t_sessionName`*	Name of the ADE Explorer or ADE Assembler session. Default: Current session

`?runMode` *`t_runMode`*

Name of the run mode that has to be run.

Possible Values:

`"Single Run, Sweeps and Corners"`

`"Monte Carlo Sampling"`

`"Global Optimization"`

`"Local Optimization"`

`"Improve Yield"`

`"Sensitivity Analysis"`

`"Conjugate Gradient Optimization"`

`"Feasibility Analysis"`

`"Create Worst Case Corners"`

`"Manual Optimization"`

`"Manual Tuning"`

`"Run Plan"`

`"Fault Simulation"`

`"Size Over Corners"`

Default: `"Single Run, Sweeps and Corners"`

Note: For ADE Explorer, you can set this argument to only `"Single Run, Sweeps and Corners"` or `"Monte Carlo Sampling"`

`?callback` *`l_callback`*

A callback procedure to be executed after completing the simulation.

`?run` *`t_runPlan`*

Name of the run plan. Use this argument in the scripts you save to run the plans created in the Run Plan assistant in ADE Assembler.

To run simulation for all the runs in the run plan, set this argument to `"All"`.

`?waitUntilDone` *`g_waitUntilDone`*	Boolean value that determines if the tool should wait for this run to complete before executing the next command in the script. Valid values: ■ `t`: Specifies that the tool should wait for the completion of this run. ■ `nil`: Specifies that the tool should not wait for the completion of this run. You can use this option when you intend to run multiple runs in parallel. This is the default value.
`?returnRunId` *`g_returnRunId`*	Boolean value that determines if the tool should return the run ID instead of the default return value, history name. Valid values: ■ `t`: Returns the run ID of the simulation run. You can use this run ID as an argument in other SKILL functions that require run ID instead of history name. ■ `nil`: Returns name of the history name of the simulation run. This is the default value.

Value Returned

`t_histName`	Name of the history created after the simulation run
`x_runID`	Run ID of the simulation run **Note:** This value is returned only when the `?returnRunId` argument is set to `nil`.
`nil`	Unsuccessful operation

Examples

Example 1:

The following example code runs the Single Run, Sweeps and Corners for the current session:

```
maeOpenSetup("solutions" "amptest" "maestro" ?mode "a")
=> "session1"
```

maeRunSimulation()

```
=> "Interactive.1"
;; returns the name of the history created by the run
```

Example 2:

The following example code runs the Monte Carlo Sampling for the current session and executes the code given with the `?callback` argument after completing the run:

maeRunSimulation(?runMode "Monte Carlo Sampling" ?callback "printf(\"Run Complete\")")

```
=> "MonteCarlo.1"
```

Example 3:

The following example code runs a simulation for the current session and executes a callback procedure after completion:

```
;; Loading setup
maeOpenSetup("testLib" "demo_top" "maestro" ?mode "a")

;; Post-simulation callback.  Called after each run.
define( RunFinishedCallback(session runID)
printf("Run ID %L has finished" runID)
)
; Execute simulation
maeRunSimulation(?callback "RunFinishedCallback")
```

Example 4:

The following example code shows how you can run simulations for multiple sessions together:

```
;; Loading setup
session1 = maeOpenSetup("testLib" "demo_top" "maestro" ?mode "a")
session2 = maeOpenSetup("testLib" "block1" "maestro" ?mode "a")

;; Post-simulation callback.  Called after each run.
define( RunFinishedCallback(session runID)
printf("Run ID %L has finished" runID)
)
; Run simulations
history1 = maeRunSimulation(?session session1 ?callback "RunFinishedCallback")
history2 = maeRunSimulation(?session session2 ?callback "RunFinishedCallback")

maeWaitUntilDone('All)

;;Generate unique file names where you want to save results

file1=strcat(history1 "_" axlGetSessionLibName(session1) "_"
axlGetSessionCellName(session1) "_" axlGetSessionViewName(session1))

file2=strcat(history2 "_" axlGetSessionLibName(session2) "_"
axlGetSessionCellName(session2) "_" axlGetSessionViewName(session2))

; file names are now available in file1 and file2 variables

;; Export Results to CSV
```

```
maeExportOutputView(?fileName file1 ?session session1 ?historyName history1)
maeExportOutputView(?fileName file2 ?session session2 ?historyName history2)
exit()
```

Example 5:

The following example code shows how to check the simulation status for each point:

```
maeRunSimulation()
x_history=axlGetHistoryEntry(x_mainSDB "Interactive.3")
rdbPath=axlGetHistoryResults(x_history)
r=axlOpenResDB(rdbPath)
r->testStatus("myTest1_1" 1) ; myTest1 is the test name, 1 is the point ID
; it will return an integer indicating the simulation status of a point

1 - Pending
2- Running
3- Done
10 -Disabled status
```

maeStopSimulation

```
maeStopSimulation(
     [ ?historyName l_historyNames ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Stops simulation runs for the given histories.

Arguments

`?historyName` *`l_historyNames`*	(Optional) List of history names. If this argument is not provided, all the simulations that are currently running are stopped.
`?session` *`t_sessionName`*	Name of the session. If not specified, the currently active session is used.

Value Returned

`t`	Name of the history created after the simulation run.
`nil`	Unsuccessful operation.

Example

The following example code runs the Single Run, Sweeps and Corners for the current session:

```
maeOpenSetup("solutions" "amptest" "maestro" ?mode "a")
=> "session1"
maeRunSimulation()
=> "Interactive.1"
;; returns the name of the history created by the run
maeStopSimulation()
```

maeSuspendSimulation

```
maeSuspendSimulation(
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Suspends the simulation run for the specified maestro session. If not specified, the current simulation run is suspended.

Arguments

`?session` *`t_sessionName`*	Name of the maestro session. Default: Current session.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

The following example shows how to suspend a simulation run:

```
maeRunSimulation()
=> "Interactive.9"

maeSuspendSimulation()
=> t
```

maeSetPreRunScript

```
maeSetPreRunScript(
     t_scriptName
     [ ?tests l_testNames ]
     [ ?enabled g_enabled ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Sets the given script as a pre-run script for the given test and sets its status.

Arguments

t_scriptName	Name of the script.
?tests *l_testNames*	List of the tests for which the script needs to be enabled.
?enabled *g_enabled*	Status of the script. Possible values: ■ `t`: Enabled ■ `nil`: Disabled Default value: `t`
?session *t_sessionName*	Name of the session. If not specified, the currently active session is used.

Value Returned

t	Successful operation
nil	Unsuccessful operation

Example

maeSetPreRunScript("preRun.il")

Sets `preRun.il` as the pre-run script for all tests and marks it enabled for all tests.

```
maeSetPreRunScript("preRun.il" ?tests '("test1" "test2"))
```

Sets `preRun.il` as the pre run script for tests `test1` and `test2`.

maeSetRunOption

```
maeSetRunOption(
     t_mode
     t_runOptionName
     t_runOptionValue
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Sets value for a run option.

Arguments

t_mode	Name of the run mode. Valid Values: `"Sampling"`, `"Global Optimization"`, `"Local Optimization"`, `"Monte Carlo Sampling"`

t_runOptionName

Name of the run option. The run option should be relevant for the run mode specified in the *t_mode* argument.

Possible values when *t_mode* is set to `"Sampling"`:

- `points` - Number of sampling points

Possible values when *t_mode* is set to `"Global Optimization"`:

- `tillsatisfied` - Optimization stops when all goals are met
- `timelimit` - Optimization stops when the program reaches the time limit (in minutes)
- `numpoints` - Optimization stops when the program reaches the number of points
- `ptswithnoimprovement` - Optimization stops when there is no improvement for the number of points

Possible values when *t_mode* is set to `"Local Optimization"`:

- `effort` - Optimization effort
- `tillsatisfied` - Optimization stops when all goals are met
- `timelimit` - Optimization stops when the program reaches the time limit (in minutes)
- `numpoints` - Optimization stops when the program reaches the number of points
- `ptswithnoimprovement` - Optimization stops when there is no improvement for the number of points.

Possible values when *t_mode* is set to `"Monte Carlo Sampling"`:

- `mcmethod` - Monte Carlo Sampling method
- `mcnumpoints` - Number of Monte Carlo sampling points

Note: Typically, this number should be at least the number of statistical variables.

t_runOptionValue

Value of the run option.

`?session` *`t_sessionName`*	Name of the session. If not specified, the currently active session is used.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

```
maeOpenSetup("solutions" "amptest" "maestro" ?mode "a")
=> "session1"
maeSetRunOption("Sampling" "points" "10")
=> t
maeRunSimulation()
=> "Interactive.1"
```

maeWaitUntilDone

```
maeWaitUntilDone(
     g_historyNames
     [ ?session t_sessionName ]
     )
     => t /nil
```

Description

Specifies the names of history checkpoints for which the tool must wait before proceeding further.

Arguments

`g_historyNames`	Specifies one or more history checkpoints for which the tool must wait to complete simulation before proceeding to the next command. Possible values: ■ Name of a history ■ A list of history names for which the simulation must be complete before the tool runs the next command ■ `'All` to specify that the tool must wait for the completion of all active runs
`?session` *`t_sessionName`*	Name of the ADE Assembler or Explorer session. Default value: Current session

Value Returned

`t`	Successful operation.
`nil`	Unsuccessful operation.

Example

```
maeOpenSetup("solutions" "amptest" "maestro")
;; the above function loads the given cellview and returns the session object
=> "session0"
maeSetVar("rload" 10k)
;; sets value for the rload variable
```

```
=> t
maeRunSimulation()
;; the above function runs the simulation for the given setup and returns the name
of the history
=> "Interactive.2"
```
maeWaitUntilDone('All)
```
=> (0)
maeExportOutputView()
;; the above function writes the results in a csv file in the current working
;; directory and returns the file path when successful.
=> "/servers/scratch02/testcases/adexl_workshop/design/Interactive.2.csv"


;;To make Assembler wait for the completion of simulation for a particular history:
hist=maeRunSimulation()
=> "Interactive.3"
```
maeWaitUntilDone(hist)
```
;; To make Assembler wait for the completion of simulation for multiple histories,
;; provide a list of all history names to the maeWaitUntilDone function, as shown
;; below:

hist=maeRunSimulation()
=> "Interactive.3"

hist1=maeRunSimulation()
=> "Interactive.4"

hist2=maeRunSimulation()
=> "Interactive.5"
```
maeWaitUntilDone(list(hist hist1))
```
maeExportOutputView()
=> "/servers/scratch02/testcases/adexl_workshop/design/Interactive.4.csv"
```

maeWriteScript

```
maeWriteScript(
     t_fileName
     [ ?session t_session ]
     [ ?shouldRunActive g_shouldRunActive ]
     [ ?runPlans l_runPlans ]
     [ ?histories l_histories ]
     )
     => h_resultsDBObj / nil
```

Description

Creates a script with the specified setup details. This script can be run from the command line.

Arguments

t_fileName	Name of the script file to be created
?session *t_session*	Name of the session from which setup details are to be written to the script By default, the current session and its active state is used.
?shouldRunActive *g_shouldRunActive*	Boolean value that specifies if the active setup should be run or not. When this argument is set to `t`, `maeRunSimulation()` is added to the script saved by `maeWriteScript`. Default value: `t`
?runPlans *l_runPlans*	List containing the names of run plans to be added to the script. By default, all the run plans defined in the active setup are saved in the script.
?histories *l_histories*	List of histories

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation
	An error message is also displayed in the log.

Example

The following example code creates a script for the current cellview with the given file name:

```
maeOpenSetup("solutions" "amptest" "maestro")
;; the above function loads the given cellview and returns the session object
=> "session0"
maeWriteScript("runPlan.il")
=> t
```

Functions for Setting Up Job Policies

SKILL functions to view and modify job policies

Function	Description
maeClearAllTestJobPolicies	Clears the test-level job policy setup for each test in the setup. The job policy setup at the global level is not removed.
maeClearTestJobPolicy	Clears the job setup of a given test and applies the global job setup.
maeGetAllJobPolicies	Returns the job policies of all the tests in the specified session.
maeGetJobControlMode	Returns the job control mode currently set in the given session.
maeHasTestJobPolicy	Checks if the given test has a test-specific job policy setup.
maeIsEvaluatorProcess	Returns `t` if currently the expression evaluator service process is running for ADE Assembler or ADE Explorer. You can use this function in your `.cdsinit` file or in custom SKILL code.
maeIsNetlistProcess	Returns `t` if currently the netlister service process is running for ADE Assembler or ADE Explorer. You can use this function in your `.cdsinit` file or in custom SKILL code.
maeSetJobControlMode	Sets the job control mode in the given session.
maeStopAllJobs	Stops all the simulation or netlisting jobs you started during the current session regardless of their state (started, getting configured, running).
maeStopJob	Stops the job for the specified job ID regardless of its state.

maeClearAllTestJobPolicies

```
maeClearAllTestJobPolicies(
     [ ?session t_session ]
     )
     => t / nil
```

Description

Clears the test-level job policy setup for each test in the setup. The job policy setup at the global level is not removed.

Arguments

`?session` *`t_sessionName`*	Name of the session. When not specified, it uses the current session.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

```
; Clears the test-level job policies for the current session
maeClearAllTestJobPolicies()
=> t

; Clears the test-level job policies for the given session
maeClearAllTestJobPolicies(?session "fnxSession0")
=> t
```

maeClearTestJobPolicy

```
maeClearTestJobPolicy(
     t_testName
     [ ?session t_session ]
     )
     => t / nil
```

Description

Clears the job setup of a given test and applies the global job setup.

Arguments

t_testName	Name of the test for which the job policies are to be removed.
?session *t_sessionName*	Name of the session. When not specified, it uses the current session.

Value Returned

t	Successful operation
nil	Unsuccessful operation

Example

```
;clear the test-level job setup for the test
maeClearTestJobPolicy("AMSBC:Inv:1")
=> t
```

maeCreateNetlistForCorner

```
maeCreateNetlistForCorner(
     t_testName
     t_cornerName
     t_netlistDir
     [ ?session t_session ]
     )
     => t / nil
```

Description

Generates a netlist for the specified corner. In case the setup contains a corner sweep, the netlist will be created for the first sweep point of the specified corner.

Arguments

t_testName	Name of the test for which the corner netlist is to be created.
t_cornerName	Name of the specified corner.
t_netlistDir	Path of the netlist directory in which the netlist is to be created.
?session *t_session*	Name of the session. When not specified, it uses the current session.

Value Returned

t	Netlist is created.
nil	Netlist is not created.

Example

Creates a netlist in the netlist directory, `C2NetlistDir`, for the corner `C2` in the test, `test1`, for the specified session.

```
axlSession =maeOpenSetup("testLib" "testCell" "maestro")
=> "fnxSession2"
maeCreateNetlistForCorner("test1" "C2" "C2NeltistDir" ?session axlSession)
=> t
```

maeGetAllJobPolicies

```
maeGetAllJobPolicies(
     [ ?session t_session ]
     )
     => l_jobPolicyNames / nil
```

Description

Returns the job policies of all the tests in the specified session.

Arguments

`?session` *`t_sessionName`*

Name of the session. When not specified, it uses the current session.

Value Returned

`l_jobPolicyNames`	List of the names of job policies
`nil`	Unsuccessful operation

Example

```
;returns all the job policies for the session

maeGetAllJobPolicies(?session "fnxSession5")
=> ("Job Policy LPF" "Maestro Default" "Maestro1" "Netlisting Default")
```

maeGetJobControlMode

```
maeGetJobControlMode(
     [ ?session t_session ]
     )
     => t_controlMode / nil
```

Description

Returns the job control mode currently set in the given session.

Arguments

`?session` *`t_sessionName`*

Name of the session. When not specified, it uses the current session.

Value Returned

`t_controlMode`	Returns `"LSCS"` or `"ICRP"` depending on the job control mode set in the given session.
`nil`	If the provided session is invalid.

Example

```
maeGetJobControlMode()
=> "LSCS"
```

maeHasTestJobPolicy

```
maeHasTestJobPolicy(
     t_testName
     [ ?session t_session ]
     )
     => t / nil
```

Description

Checks if the given test has a test-specific job policy setup.

Arguments

t_testName	Name of the specified test.
?session *t_sessionName*	Name of the session. When not specified, it uses the current session.

Value Returned

t	If the given test has a test-specific job policy setup.
nil	If the given test does not have a test-specific job policy setup.

Example

The following example shows how to check if a given test has a test-specific job policy setup:

```
maeHasTestJobPolicy("AMSBC:Inv:1")
=> t
```

maeIsEvaluatorProcess

```
maeIsEvaluatorProcess(
     [ ?session t_session ]
     )
     => t / nil
```

Description

Returns `t` if currently the expression evaluator service process is running for ADE Assembler or ADE Explorer. You can use this function in your `.cdsinit` file or in custom SKILL code.

Argument

None

Value Returned

`t`	If currently the expression evaluator service process is running.
`nil`	If the expression evaluator service process is not running.

Example

```
maeIsEvaluatorProcess( )
=> nil
```

maeIsNetlistProcess

```
maeIsNetlistProcess(
     [ ?session t_session ]
     )
     => t / nil
```

Description

Returns `t` if currently the netlister service process is running for ADE Assembler or ADE Explorer. You can use this function in your `.cdsinit` file or in custom SKILL code.

Argument

None

Value Returned

`t`	If currently the netlister service process is running.
`nil`	If the netlister service process is not running.

Example

```
maeIsNetlistProcess( )
=> t
```

maeSetJobControlMode

```
maeSetJobControlMode
     t_mode(
     [ ?session t_session ]
     )
     => t / nil
```

Description

Sets the job control mode in the given session.

Arguments

t_mode	The job control mode to be set.
?session *t_sessionName*	Name of the session. When not specified, it uses the current session.

Value Returned

t	Successful operation
nil	Unsuccessful operation

Example

```
; Sets the specified job control mode for the current session
maeSetJobControlMode("ICRP")
=>t

;Sets the specified job control mode for the given session
maeSetJobControlMode("ICRP" ?session "fnxSession0")
=>t
```

maeStopAllJobs

```
maeStopAllJobs(
     [ ?jobType t_jobType ]
     [ ?force g_force ]
     [ ?session t_session ]
     )
     => t / nil
```

Description

Stops all the simulation or netlisting jobs you started during the current session regardless of their state (started, getting configured, running).

Arguments

`?jobType` *`t_jobType`*	The type of the job to be stopped. Possible values: ■ "simulation" ■ "netlisting" When no value is specified, both types of jobs are stopped.
`?force` *`g_force`*	If set to `t`, kills the job by sending a signal, such as `SIGKILL`. Otherwise, it kills the job by using an internal communication mechanism.
`?session` *`t_sessionName`*	Name of the session. When not specified, it uses the current session.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

```
;Stops all the jobs for the current session
maeStopAllJobs()
=>t
```

```
;Stops all netlister service jobs for the current session
maeStopAllJobs("netlisting")
=>t

;Stops all jobs by sending a SIGKILL signal
maeStopAllJobs(?force t)
=>t
```

maeStopJob

```
maeStopJob(
     [ ?jobType t_jobType ]
     [ ?force g_force ]
     )
     => t / nil
```

Description

Stops the job for the specified job ID regardless of its state.

Arguments

x_jobID	The ID of the job to be stopped.
?force *g_force*	If set to `t`, kills the job by sending a signal, such as `SIGKILL`. Otherwise, it kills the job by using an internal communication mechanism.

Value Returned

`t`	Successful operation
`nil`	When no job is found for the given job ID

Example

```
;Stops the job for the specified job ID
maeStopJob(3)
=>t
;Stops the job for the specified job ID by sending a SIGKILL signal
maeStopJob(5 ?force t)
=>t
```

Functions to Work with Simulation Results

SKILL functions to view and save simulation results for maestro cellviews

Function	Description
getSimRunInfo	Returns the value of the specified type of simulation information for a given ADE output expression by accessing the related `psf` directory.
maeCloseResults	Closes the results opened by the maeOpenResults function.
maeDeleteSimulationData	Deletes the simulation results data for the given history.
maeExportOutputView	Exports the output or results view to the specified .csv or .html file.
maeGetNBestDesignPoints	Returns the best n design points for the opened results.
maeGetOutputValue	Returns value of the given output.
maeGetParamConditions	Returns the design parameter conditions for the given design point ID.
maeGetResultOutputs	Returns the list of outputs for the opened result.
maeGetResultTests	Returns the list of tests for the opened result.
maeGetSpecStatus	Returns the specification status for the given output, test and point id.
maeGetOverallSpecStatus	Returns the overall specification status for the current history.
maeGetOverallYield	Returns the overall yield for the given history.
maeGetOverallYield	Returns the overall yield for the given history.
maeOpenResults	Opens the result for the given history or run plan, and sets the result pointer to be used by other functions.
maeReadResDB	Returns a handle to the results database for the given history.
maeRestoreHistory	Restores the given history as active setup. The active setup is replaced with the setup from the history being restored.

SKILL functions, ***continued*****to view and save simulation results for maestro cellviews**

Function	Description
maeWriteDatasheet	Writes the results for the given history in a datasheet.

getSimRunInfo

```
getSimRunInfo(
     t_Type
     )
     => s_Value/ nil
```

Description

Returns the value of the specified type of simulation information for a given ADE output expression by accessing the related `psf` directory.

Note the following:

- The specified job control mode must be LSCS.
- This function must be set in the *Outputs Setup* tab.
- The return value can be viewed in the *Results* tab.

Argument

t_Type	The type of the simulation information that you the need the value for. The possible values for this argument are `memory`, `threads`, `nodes`, `host`, `jobId` and `startTime`.

Value Returned

s_Value	The value of the specified type of simulation information for a given ADE expression.
`nil`	If the job control mode is set to ICRP or the specified type of simulation information is invalid.

Example

The following example shows how this function returns the value of the specified type of simulation information for a given ADE output expression.

Set the function in the *Outputs Setup* as shown below.

Outputs Setup | Results

11 rows

Test	Name	Type	Details	EvalType	Plot	Save	
Two_Stage_Opamp:OpAmp_AC_top:1		signal (I)	/V1/PLUS	point	✓	✓	
Two_Stage_Opamp:OpAmp_AC_top:1		signal	/OUT	point	✓	✓	
Two_Stage_Opamp:OpAmp_AC_top:1	Current	expr	abs(IDC("/V1/PLUS"))	point	✓		< 1.5
Two_Stage_Opamp:OpAmp_AC_top:1	UGF	expr	cross(dB20(mag(VF("/OUT"))) 0 ...	point	✓		> 53
Two_Stage_Opamp:OpAmp_AC_top:1	Gain	expr	value(dB20(mag(VF("/OUT"))) 1)	point	✓		> 44
Two_Stage_Opamp:OpAmp_AC_top:1	Voffset	expr	(VDC("/inm") - VDC("/inp"))	point	✓		rang
Two_Stage_Opamp:OpAmp_AC_top:1	CMRR	expr	(value(dB20(mag(VF("/OUT"))) 1...	point	✓		
Two_Stage_Opamp:OpAmp_AC_top:1	GainVsupply1	expr	value(db(getData("/V1" ?result "...	point	✓		
Two_Stage_Opamp:OpAmp_AC_top:1	PSRR	expr	(value(dB20(mag(VF("/OUT"))) 1...	point	✓		
Two_Stage_Opamp:OpAmp_AC_top:1	GainCm	expr	value(db(getData("/Vcm" ?resul...	point	✓		
Two_Stage_Opamp:OpAmp_AC_top:1		expr	getSimRunInfo("memory")	point	✓		

The results are displayed in the Results tab.

Outputs Setup | Results

19 rows

Test	Output	Nominal	Spec	Weight	Pass/Fail	Min	
Two_Stage_Opa...	/V1/PLUS						
Two_Stage_Opa...	/OUT						
Two_Stage_Opa...	Current	1.126m	< 1.5m		pass	1.126m	1
Two_Stage_Opa...	UGF	464.2M	> 533M		fail	464.2M	4
Two_Stage_Opa...	Gain	46.74	> 44		pass	46.74	
Two_Stage_Opa...	Voffset	3.065m	range -10m 10m		pass	3.065m	3
Two_Stage_Opa...	CMRR	eval err					
Two_Stage_Opa...	GainVsupply1	-5.935				-5.935	-
Two_Stage_Opa...	PSRR	52.68				52.68	
Two_Stage_Opa...	GainCm	eval err					
Two_Stage_Opa...	getSimRunInfo("...	98.8M				95.4M	9
Two_Stage_Opa...	sim_Peak_Mem...	98.8M				95.4M	9
Two_Stage_Opa...	sim_Threads_Us...	1				1	

maeCloseResults

```
maeCloseResults()
     => t / nil
```

Description

Closes the results opened by the maeOpenResults function.

Arguments

None

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

```
; Load the results for that history
maeOpenResults(?history "Interactive.1")
=> t
>
; Get the ouput value
maeGetOutputValue("outputA" "solutions:amptest:1")
=> 2.2
;close the results
maeCloseResults()
=> t
```

maeDeleteSimulationData

```
maeDeleteSimulationData(
     t_historyName
     ?session t_sessionName
     ?keepNetlist g_keepNetlist
     ?keepQuickPlot g_keepQuickPlot
     )=> t / nil
```

Description

Deletes the simulation results data for the given history.

Arguments

t_historyName	Name of the history for which you need to delete the simulation data
?session *t_sessionName*	Name of the session
?keepNetlist *g_keepNetlist*	Boolean value that specifies whether to retain the netlist even if simulation data is deleted.
?keepQuickPlot *g_keepQuickPlot*	Boolean value that specifies whether to retain the quick plot data even if simulation data is deleted.

Value Returned

t	Successful operation
nil	Unsuccessful operation

Example

```
maeDeleteSimulationData("Interactive.0")
=> t
;; Deletes simulation data/Netlist and quick plot.
maeDeleteSimulationData("Interactive.0 ?keepNetlist t)
=> t
;;Deletes simulation data while keeping the netlist folder and spectre logs.
```

maeExportOutputView

```
maeExportOutputView(
     [ ?session t_sessionName ]
     [ ?fileName t_fileName ]
     [ ?view t_viewType ]
     [ ?history t_historyName ]
     [ ?testName t_testName ]
     [ ?filterName t_filterName ]
     [ ?clearAllFilters g_clearAllFilters ]
     )
     => t / nil
```

Description

Exports the output or results view to the specified .csv or .html file.

Arguments

`?session` *`t_sessionName`*	Name of the ADE Assembler or ADE Explorer session. Default value: current session
`?fileName` *`t_fileName`*	Path and name of the file to which results are to be exported.
`?view` *`t_viewType`*	Name of the output view to be exported. Valid values: ■ `"Detail"` ■ `"Detail - Transpose"` ■ `"Status"` ■ `"Summary"` ■ `"Yield"` ■ `"Checks/Asserts"` ■ `"Fault"` ■ `"Current"` Default value: `"Current"`

`?historyName` *`t_historyName`*

Name of the history for which outputs are to be exported.

Default value: ""

`?testName` *`t_testName`*

Name of the test for which outputs are to be exported. This argument is useful when you are exporting results from a multi-test cellview.

Note: This argument is supported only for the `Checks/Asserts` and `Fault` result views.

`?filterName` *`t_filterName`*

Name of a saved filter to be applied before exporting outputs.

Note: This argument is supported only for the `Checks/Asserts` and `Fault` result views.

`?clearAllFilters` *`g_clearAllFilters`*

Specifies whether to clear all column filters before exporting outputs (`t`) or not (`nil`).

Note: This argument is supported only for the `Checks/Asserts` and `Fault` result views.

Value Returned

`t`	The given output view is successfully exported.
`nil`	Unsuccessful operation

Examples

The following example code exports the yield view of a Monte Carlo run to a file named `abc.csv`:

```
maeOpenSetup("solutions" "amptest" "maestro")
;; the above function loads the given cellview and returns the session object
=> "fnxSession0"
maeSetVar("rload" 10k)
;; sets value for the rload variable
=> t
maeRunSimulation(?runMode "Monte Carlo Sampling")
;; the above function runs the simulation for the given setup and returns the name
of the history
```

```
=> "MonteCarlo.2"
maeWaitUntilDone('All)
=> (0)
maeExportOutputView(?session "fnxSession0" ?fileName "./abc.csv" ?view "Yield")
=> "/servers/scratch02/testcases/adexl_workshop/design/abc.csv"
```

The following example code saves the results of the Checks/Asserts result view for the current history of test `demo_top:1` to a file named `deviceChecks.csv`. While doing that, it clears the filter named `Device Checks`:

```
maeExportOutputView( ?session "fnxSession0" ?fileName "./deviceChecks.csv" ?view
"Checks/Asserts" ?testName "demo_top:1" ?filterName "Device Checks"
?clearAllFilters t)
```

```
;; for Checks/Asserts view the optional arguments can be used to specify a named
filter
```

maeGetNBestDesignPoints

```
maeGetNBestDesignPoints(
     [ ?count n_designPoints ]
     )
     => l_bestDesignPoints / nil
```

Description

Returns the best n design points for the opened results.

Arguments

`?count` *`n_designPoints`*	Number of best design points required. Default: `1`

Value Returned

`l_bestDesignPoints`	List of best design points
`nil`	Unsuccessful operation

Example

```
; Load the result for history Interactive.1
maeOpenResults(?history "Interactive.1")

; Return best 3 design points for the results loaded.
maeGetNBestDesignPoints(?count 3)
=>(4 2 7)
>
maeCloseResults()
=> t
```

maeGetOutputValue

```
maeGetOutputValue(
     t_outputName
     t_testName
     [ ?cornerName t_cornerName ]
     [ ?pointId x_pointId ]
     [ ?evalType t_evalType ]
     )
     => x_value / nil
```

Description

Returns value of the given output.

Arguments

t_outputName	Name of the output for which value is to be returned.
t_testName	Name of the test.
?cornerName *t_cornerName*	Name of the corner. If not specified, output value is returned for the nominal corner.
?pointId *x_pointID*	Point ID for which you need to get the output value. If not specified, output value is shown for the point with ID = 1.
?evalType *t_evalType*	Evaluation type. Valid values: `"point"`, `"corners"`, `"sweeps"`, `"all"` Default value: `"point"`

Value Returned

x_value	Value of the output
nil	Unsuccessful operation

Example

```
; Load the results for that history
maeOpenResults(?history "Interactive.1")
=> t
>
; Get the ouput value
```

maeGetOutputValue("outputA" "solutions:amptest:1")

```
=> 2.2
;close the results
maeCloseResults()
=> t
```

maeGetParamConditions

```
maeGetParamConditions(
     x_designPointID
     )
     => l_designParamConditions / nil
```

Description

Returns the design parameter conditions for the given design point ID.

Arguments

x_designPointID	ID of the design point for which the parameter conditions are to be returned.

Value Returned

l_designParamConditions	List of design parameter conditions
nil	Unsuccessful operation

Example

```
;create setup for simulation

; Run Simulation
maeRunSimulation()
=> "Interactive.2"

; open the results
maeOpenResults(?history "LocalOpt.1")
=> t

; Find the best design point
bestPt=maeGetNBestDesignPoints(?count 2)
=> (8 10)
; view the conditions for the first best point

maeGetParamConditions(car(bestPt))
=> (("vdd" "3") ("opamp090/full_diff_opamp_AC/schematic/R2/m" "2")

)
; close the results
maeCloseResults()
=> t
```

maeGetParameter

```
maeGetParameter(
     t_paramName
     [ ?typeName t_typeName ]
     [ ?typeValue t_typeValue ]
     [ ?session t_sessionName ]
     )
     => t_value / nil
```

Description

Returns value of the given parameter for the given test or corner.

Arguments

t_paramName	Name of the parameter Parameter names must contain at least five non-empty names each separated by a `/` (slash), typically representing the `library/cell/view/instance/property` to be modified. Multiple `"instance/"` strings can be specified to specify hierarchical parameters, such as `library/cell/view/instance0/instance1/instance2/property`.
`?typeName` *l_typeName*	Type for which paramater details are to be returned. Possible values: `"test"` or `"corner"`. Default value: `"test"`
`?typeValue` *g_typeValue*	Name of the corner for which value has to be returned. This field is irrelevant if `?typeName` is set to `test`. Default value: `"Global"`
`?session` *t_sessionName*	Name of the session. Default: Current session.

Value Returned

t_value	Value of the parameter
nil	Unsuccessful operation

Example

```
maeOpenSetup("solutions" "amptest" "maestro")
;; the above function loads the given cellview and returns the session object
=> "session0"
maeGetParameter("solutions/amptest/schematic/I0/l" ?typeName "corner" ?typevalue
"C0")
;; gets the value of I1/l parameter for corner C0
=> 10n
```

maeGetResultOutputs

```
maeGetResultOutputs(
     [ ?testName t_testName ]
     )
     => l_outputNames / nil
```

Description

Returns the list of outputs for the opened result.

Arguments

`?testName` *`t_testName`*	Test for which outputs required. If not given, return for all tests.

Value Returned

`l_outputNames`	List containing the name of outputs
`nil`	Unsuccessful operation

Example

```
;Load the result for history Interactive.1
maeOpenResults(?history "Interactive.1")
=> t
maeGetResultOutputs()
=> ("Current" "DCGain" "/OUTN" "?OUTP")
>
maeCloseResults()
=> t
```

maeGetResultTests

```
maeGetResultTests()
     => l_testNames / nil
```

Description

Returns the list of tests for the opened result.

Arguments

None

Value Returned

l_testNames	List containing the name of tests
nil	Unsuccessful operation

Example

```
;Load the result for history Interactive.1
maeOpenResults(?history "Interactive.1")
=> t
maeGetResultOutputs()
=> ("Current" "DCGain" "/OUTN" "?OUTP")
>
maeGetResultTests()
>("solutions:ampTest:1" "solutions:amptest:2")
>
maeCloseResults()
=> t
```

maeGetSpecStatus

```
maeGetSpecStatus(
     t_outputName
     t_testName
     [ ?pointId n_pointId ]
     )
     => t / nil
```

Description

Returns the specification status for the given output, test and point id.

Arguments

t_outputName	Name of the output
t_testName	Name of the test
?pointId *n_pointId*	Point ID for which you need to get the spec status Default value: `1`

Value Returned

`"pass"`	If the spec passes
`"fail"`	If the spec fails
`"undefined"`	Invalid case

Example

```
; Load the results for that history
maeOpenResults(?history "Interactive.1")
=> t
>
; Get the output value
maeGetOutputValue("outputA" "solutions:amptest:1")
=> 2.2
; Get the spec status
maeGetSpecStatus("outputA" "solutions:amptest:1")
=>"pass"
;close the results
maeCloseResults()
=> t
```

maeGetTestOutputs

```
maeGetTestOutputs(
     t_testName
     [ ?session t_session ]
     )
     => l_outputs / nil
```

Description

Returns the list of outputs for the specified test in the current session or a specific maestro session.

Arguments

t_testName	Name of the test for which the list of outputs is required.
?session *t_session*	Name of the session. If not specified, the current session is used.

Value Returned

l_outputs	A list containing the outputs for the specified test.
nil	Unsuccessful operation.

Example

; Returns the name of the maestro session.

```
maestroSession = maeOpenSetup("Two_Stage_Opamp" "OpAmp" "maestro_basic")
=> "fnxSession8"
```

; Returns the name of the tests.

```
maeGetSetup(?typeName "tests" ?session maestroSession)
=> ("AC" "TRAN" "AC:1" "AC:2")
```

; Returns the outputs for the test TRAN.

```
outputList = maeGetTestOutputs("TRAN" ?session maestroSession)
=> (sevOutputStruct@0x220138a8 sevOutputStruct@0x220138c0
sevOutputStruct@0x220138d8 sevOutputStruct@0x220138f0 sevOutputStruct@0x22013908
sevOutputStruct@0x22013920 sevOutputStruct@0x22013938 sevOutputStruct@0x22013950
sevOutputStruct@0x22013968)
```

; Prints the name, expression or signal name for each output.

```
foreach( mapcar output outputList or(output->name output->expression
output->signal))
=> ("/OUT" "Swing" "SettlingTime" "RelativeSwingPercent" "PhaseMargin"
    "/net06" "/netAVDD" "/IN" "/gnd!"
)
```

maeGetOverallSpecStatus

```
maeGetOverallSpecStatus(
     [ ?verbose verbose ]
     )
     => t / nil
```

Description

Returns the overall specification status for the current history.

Arguments

`?verbose` *`verbose`*	Specifies whether to print a detailed report including the pass or fail status for each test name and output name combination. Default value: `nil`

Value Returned

`t`	The spec status for each test-output combination. It also returns a list containing the overall status and a sub list of `testName:outputName: <PASSED/FAILED>`. The first value in the list gives the overall status.
`nil`	If the function is not run successfully. For example, when no results are available.

Example

```
; Load the results for a history
maeOpenResults(?history "Interactive.1")
=> t
>
; Get the overall spec status
; when ?verbse is set to t, an overall pass/fail status of each output is shown in
; addition to a detailed report for each output.
;
maeGetOverallSpecStatus(?verbose t)
=>
```

```
Spec Status:
DCGain:lowValue : PASSED
DCGain:highValue : PASSED
DCGain:dcGain : PASSED
DCGain:lowValue2 : PASSED
DCGain:highValue2 : PASSED
DCGain:dcGain2 : PASSED
DCGain:maxCurrent : PASSED
Swing:Swing : PASSED
Swing:SettlingTime : PASSED
Swing:RelativeSwingPercent : PASSED
Swing:lowValue : PASSED
Swing:highValue : PASSED
Swing:Swing2 : PASSED
Swing:lowValue2 : PASSED
Swing:highValue2 : PASSED
Swing:maxCurrent : PASSED
overAll : PASSED
(("overAll" t)
    ("DCGain:lowValue" t)
    ("DCGain:highValue" t)
    ("DCGain:dcGain" t)
    ("DCGain:lowValue2" t)
    ("DCGain:highValue2" t)
    ("DCGain:dcGain2" t)
    ("DCGain:maxCurrent" t)
    ("Swing:Swing" t)
    ("Swing:SettlingTime" t)
    ("Swing:RelativeSwingPercent" t)

; When ?verbose is set to nil, only a list containing the status of each output at
; the test level is returned
```

maeGetOverallSpecStatus(?verbose nil)
=>

```
maeGetOverallSpecStatus(?verbose nil)
(("overAll" t)
    ("DCGain:lowValue" t)
    ("DCGain:highValue" t)
    ("DCGain:dcGain" t)
    ("DCGain:lowValue2" t)
    ("DCGain:highValue2" t)
    ("DCGain:dcGain2" t)
    ("DCGain:maxCurrent" t)
    ("Swing:Swing" t)
    ("Swing:SettlingTime" t)
    ("Swing:RelativeSwingPercent" t)
    ("Swing:lowValue" t)

;close the results
maeCloseResults()
=> t
```

maeGetOverallYield

```
maeGetOverallYield(
     t_historyName
     [ t_sessionName ]
     )
     => t / nil
```

Description

Returns the overall yield for the given history.

Arguments

t_historyName	Name of the history
t_sessionName	Name of the session Default value: Current session

Value Returned

r_yieldInfo	Yield details from the history results
nil	Unsuccessful operation

Example

```
maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"

maeRunSimulation()
=> "Run.2"

maeGetOverallYield("Run.2")
=>(nil Yield 100 PassedPoints 2 ErrorPoints 0 )
```

maeImportHistory

```
maeImportHistory(
     t_libName
     t_cellName
     t_viewName
     [ ?session t_sessionName ]
     [ ?history t_historyName ]
     [ ?copyPSF g_copyPSF ]
     [ ?overwrite g_overwrite ]
     )
     => t / nil
```

Description

Imports the zip file for the given history from one cellview into the current cellview. You can import histories only for cellviews that use the separate history management feature. This feature is enabled by default in ICADVM20.1 and can be ena the bled in IC6.1.8 releases using the useSeparateHistoryFileManagement environment variable.

Arguments

t_libName Name of a library.

t_cellName Name of a cell in the given library.

t_viewName Name of a maestro cellview from which the history is to be imported.

?session *t_sessionName*

Name of the Assembler session.

Default: Current session.

?history *t_historyName*

Name of the history to be imported.

?copyPSF *g_copyPSF* A boolean value that specifies whether to copy the `psf` directory along with the history zip.

Possible values:

- `t`: Copies the `psf` directory along with the history zip
- `nil`: Does not copy the `psf` directory

?overwrite *g_overwrite*

A boolean value that specifies whether to overwrite a history of the same name if any exists in the destination cellview.

Possible values:

- `t`: Overwrites any history that exists with the same name
- `nil`: Does nor import the specified history if another history already exists with the same name

Value Returned

`t`	When the specified history is imported.
`nil`	When the specified history is not imported.

Example

The following example code shows how to import two histories from the `maestro2` cellview to the `maestro` cellview of the `ampTest` cell.

```
maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"

maeImportHistory("solutions" "ampTest" "maestro2" ?history '("Interactive.1"
"Interactive.2"))
=>t
```

maeOpenResults

```
maeOpenResults(
     [ ?session t_sessionName ]
     [ ?history t_historyName ]
     [ ?run t_runName ]
     )
     => t / nil
```

Description

Opens the result for the given history or run plan, and sets the result pointer to be used by other functions.

Arguments

`?session` *`t_sessionName`*	Name of the ADE Assembler or Explorer session. Default value: current session.
`?history` *`t_historyName`*	Name of the history or child history. If a run saves child histories, specify the name of the child history to open results. You can also use this argument to open the results of a history saved for a run in the run plan.
`?run` *`t_runName`*	Name of the run plan. This argument is considered only when the `?historyName` argument is not available. For example, while accessing the results of one run in another run of a run plan. Since the history is not yet created while you are writing the script, you can use the `?run` argument to access the results of the specified run. *Important* Use the `?run` argument only inside the pre-run script of a run to access the results of another run. This argument does not work when the SKILL function is run from CIW.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

Example 1

```
; Load the results for a history
maeOpenResults(?history "Interactive.1")
=> t
>
; Get the ouput value
maeGetOutputValue("outputA" "solutions:amptest:1")
=> 2.2
;close the results
maeCloseResults()
=> t
```

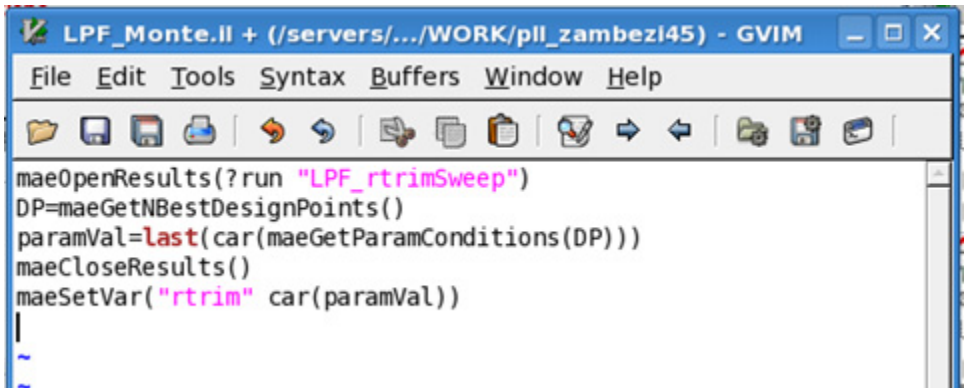
Example 2

```
; Loads the results of a run plan
maeOpenResults(?history "Plan.0.Run.0")
```

; **Note: If you are opening the results saved for a run plan, it is important to specify the name of a child history, Plan.0.Run.0, instead of the parent history, Plan.0.**

Example 3

```
; Loads the results of a run in the pre-run script of another run
```



maeReadResDB

```
maeReadResDB(
     [ ?historyName t_historyName ]
     [ ?session t_sessionName ]
     [ ?run t_runName ]
     )
     => h_resultsDBObj / nil
```

Description

Returns a handle to the results database for the given history.

This handle provides read-only access to the results database that contains objects of the following five types:

- point - a design point
- corner - a corner defined for a particular design point
- test - a test defined for a particular corner
- param - a parameter defined for a particular corner
- output - an output defined for a particular test

There is a hierarchical relationship between the instances of these objects. For example, a point is associated with one or more corners. Each corner is associated with one or more tests. Each test is associated with zero or more outputs, and so on. As a result of this relationship, for an object, you can access the properties of the object itself and other objects related to it.

Each object has:

- Three properties: `name`, which returns the name of the object; `value`, which returns its value; and a property that returns ID of the parent object. For example, an object of type corner has a property `pointID` that returns the ID of the parent point object.
- A set of member functions that return the instances of that object type and other related types. For example, using an instance of type output, you can get the value of an output object and its parent test instance. Using the functions given for a test instance, you can get a corner instance. For a corner instance, you can get the parameters which were used to generate the output, as well as the point and its ID.

In addition, the result database provides a function `help()` for each object of the above mentioned types to displays the list and description of functions that can be called using that particular object. For example, function call `help('corner)` displays a list of all the

functions that can be called using an object of type corner. `help('all)` displays help for all the object types.

Arguments

`?historyName` *`t_historyName`*	Name of the history for which results are required. If not specified last active run will be used.
`?session` *`t_sessionName`*	Name of the session. Default: Current session.
`?run` *`t_runName`*	Name of the run plan. This argument is considered only when the `?historyName` argument is not used.

Value Returned

`h_resultsDBObj`	Handle to the results database.
`nil`	Unsuccessful operation.

Example

Example 1:

The following example returns a handle to access the results for a history named `Interactive.1`:

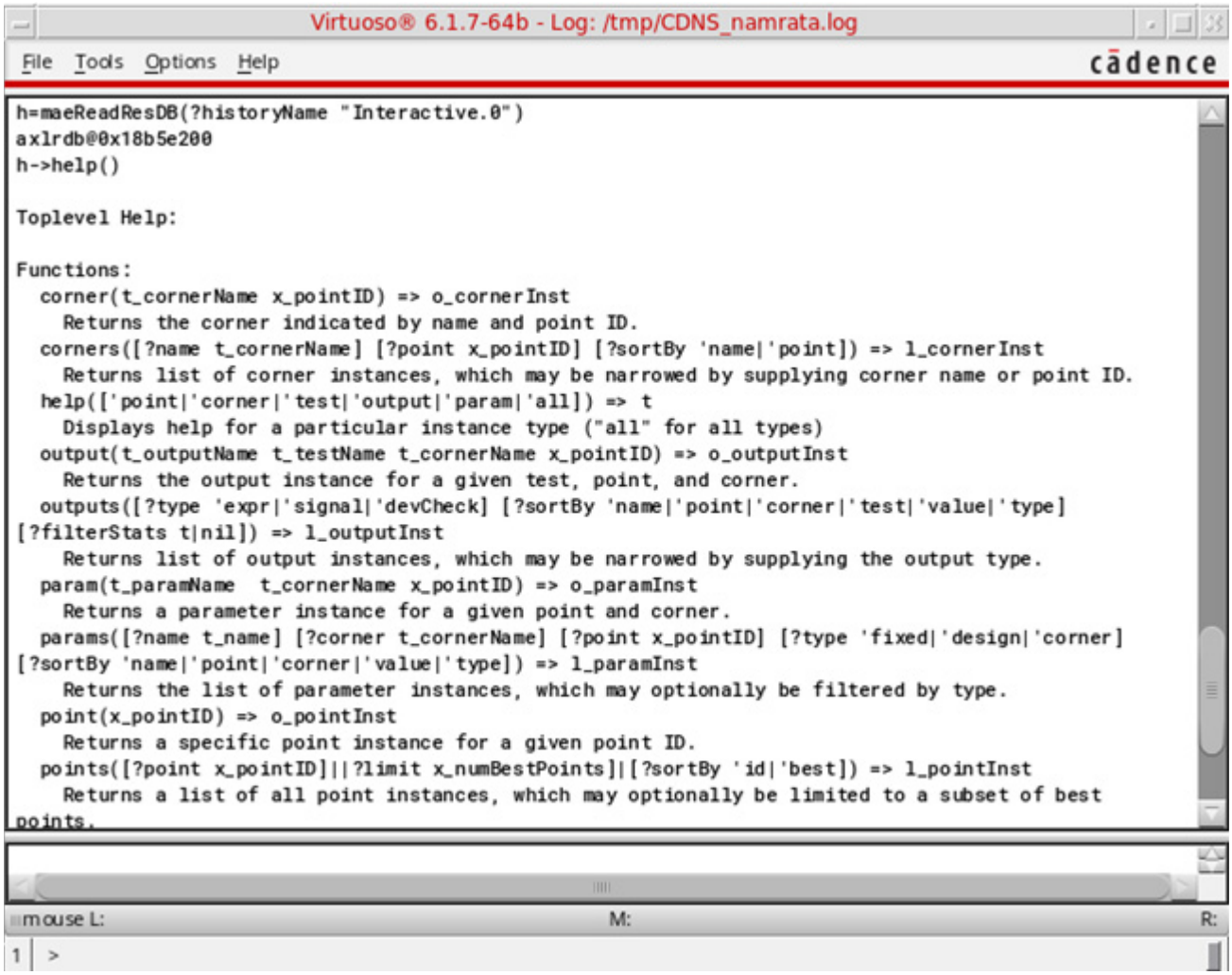
```
maeReadResDB( ?historyName "Interactive.1" )
```

Example 2:

The following example code opens the results database and displays built-in help:

```
rdb = maeReadResDB( ?historyName "Interactive.1" )
=> axlrdb@0x18b5e200
rdb -> help()
```

Help is displayed in the CIW, as shown below.



As shown in the help above, you can use the handle to access specific results from the results database.

```
; The following statement returns the point object for design point 1

pt = rdb->point(1)
=> axlrdbd@0x18b5e230

; The following code prints corner name, test name, output name and its value
; for each output of type expression


foreach(out pt->outputs(?type 'expr ?sortBy 'corner)

printf("corner=%s, test=%s, output=%s, value=%L\n" out->cornerName out->testName
out->name out->value)
)
corner=C0, test=test1, output=VAR("val1"), value=99.22984
corner=C0, test=test1, output=freq_res, value=5.112893e+08
```

```
corner=C0, test=test2, output=(((xmin(mag(IF("/I0/p1")) 1) / 1000000) - 200) * 4),
value=1299.23
corner=C0, test=test2, output=myCalib, value=99.22984
corner=C0, test=test2, output=calib_dummy, value=1
(axlrdbo@0x18476638 axlrdbo@0x18476620 axlrdbo@0x184765f0 axlrdbo@0x184765c
axlrdbo@0x18476590)
```

maeRestoreHistory

```
maeRestoreHistory(
     t_histName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Restores the given history as active setup. The active setup is replaced with the setup from the history being restored.

Note: If you restore a history saved from ADE Explorer into an ADE Assembler session, all the existing tests are removed and only a single test, pertaining to the history, is displayed.

Arguments

t_histName	Name of the history to be restored.
?session *t_sessionName*	Name of the Assembler session. Default: Current session.

Value Returned

t	Successful operation
nil	Unsuccessful operation

Example

The following example shows how to restore setup from a history:

```
maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeRestoreHistory("Interactive.1")
```

maeWriteDatasheet

```
maeWriteDatasheet(
     [ ?directory t_directory ]
     [ ?resultsSummary t_resultsSummary ]
     [ ?testsSummary t_testsSummary ]
     [ ?detailedResults g_detailedResults ]
     [ ?plots g_points ]
     [ ?designVarsSummary g_designVarsSummary ]
     [ ?paramsSummary g_paramsSummary ]
     [ ?cornersSummary g_cornersSummary ]
     [ ?setupSummary g_setupSummary ]
     [ ?launchBrowser g_launchBrowser ]
     [ ?name t_name ]
     [ ?session t_sessionName ]
     [ ?historyName t_historyName ]
     )
     => t / nil
```

Description

Writes the results for the given history in a datasheet.

Arguments

`?directory` *`t_directory`*

Name and path to the target directory where the datasheet is to be saved.

`?resultsSummary` *`g_resultsSummary`*

Boolean to specify whether or not to print a results summary sheet containing specification sheet pass/fail table.

Default Value: `t`

`?testsSummary` *`g_testsSummary`*

Boolean to specify whether or not to print a tests summary sheet containing details about the tests, sweeps, and corners.

Default Value: `t`

`?detailedResults` *`g_detailedResults`*

Name of the history to generate results for all the points.

Default Value: `t`

`?plots` *`g_plots`*

Boolean to specify whether or not to print the plots in the generated datasheet.

Default Value: `t`

`?designVarsSummary` *`g_designVarsSummary`*

Boolean to specify whether or not to save the design variable summary in the generated datasheet.

Default Value: `t`

`?paramsSummary` *`g_paramsSummary`*

Boolean to specify whether or not to save the parameters summary in the generated datasheet.

Default Value: `t`

`?cornersSummary` *`g_cornersSummary`*

Boolean to specify whether or not to save the corners summary in the generated datasheet.

Default Value: `t`

`?setupSummary` *`g_setupSummary`*

Boolean to specify whether or not to save the setup summary in the generated datasheet.

Default Value: `t`

`?launchBrowser` *`g_launchBrowser`*

Boolean to specify whether or not to launch a browser window to view the generated datasheet.

Default Value: `t`

`?name` *`t_name`*

Specifies the name of the top level file and directory created for the datasheet. For example, if `?name` is set to `myDatasheet`, the top level file and directory are named as `myDatasheet.html` and `myDatasheet/`.

`?session` *`t_sessionName`*

Name of the Assembler session.

Default: Current session.

`?historyName` *`t_historyName`*

Name of the history for which results are to be printed in a datasheet.

Certain run modes create a group history that contains multiple child histories. For such run modes, provide the name of the group history to create a consolidated datasheet for the group run, and provide the name of that child history to create a datasheet for a specific child history.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

The following example code runs a simulation for a maestro cellview and writes the results in a datasheet:

```
maeOpenSetup("solutions" "ampTest" "maestro")
=> "session0"
maeRunSimulation()
=> "Interactive.2"
maeWriteDatasheet(?historyName "Interactive.2")
=> t
```

The following example code shows how to create a datasheet for run `Run.1` which is a part of the run plan history `Plan.1`:

```
maeWriteDatasheet(?historyName "Plan.1.Run.1")
=> t
```

For the same example, you can create a datasheet for the complete group history that contains all child histories by using the following code:

```
maeWriteDatasheet(?historyName "Plan.1")
=> t
```

Example script

The following example shows how to view the simulation results:

```
; load the setup
maeOpenSetup("solutions" "ampTest" "maestro") > t

; check which tests are there in setup
maeGetSetup() -> ("solutions"amptest:1" "solutions:amptest:2")

; Enable a test "solution:amptest:1"
maeSetSetup(?tests '("solutions:amptest:1"))

; Set the value of global variable
"CAP" as 5p: maeSetVar("CAP" 5p)

; Disable corners C0 and C1
maeSetSetup(?corners '("C0"  "C1") ?enabled nil)

; Run Simulation
maeRunSimulation()
=> "Interactive.2"

; open the results
maeOpenResults(?history "Interactive.2")
=> t

; view the list of outputs in the results
maeGetResultOutputs()

; Read the value of a specific output
maeGetOutputValue("outputA" "testA")
=> 2.2

; Find the best design point
maeGetNBestDesignPoints(?count 2)
=> (8 10)

; close the results
maeCloseResults()
=> t
```

Functions for Checks and Asserts

Use the following functions to manage violations reported for checks and asserts:

Function	Description
maeCloseViolationDb	Closes an open checks and asserts database using the database ID returned by maeOpenViolationDb.
maeOpenViolationDb	Opens a connection to the checks and asserts database for the given maestro cellview and returns a unique ID for the connection. You can use this ID to read or modify the database and its attached waiver SQL database using other SKILL functions. You must close this connection using maeCloseViolationDb to release the connection and complete the transactions on the database.
maeWaiveViolation	Adds to the waivers SQL database a rule to waive a check and assert for the given object.
maeUnWaiveViolation	Removes from the waivers SQL database a waive rule matching the given criteria.

maeCloseViolationDb

```
maeCloseViolationDb(
     x_dbId
     )
     => t / nil
```

Description

Closes an open checks and asserts database using the database ID returned by maeOpenViolationDb.

Note: Ensure that the history is closed before you run this function.

Arguments

x_dbId	An integer identifying the connection for an open checks and asserts database.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

The following example opens a database, performs a task on it, and then closes it:

```
let( (dbId)
     dbId = maeOpenViolationDb( "fnxSession0" "demo_top:1" "Interactive.1")
     when( dbId != -1
          maeWaiveViolation( dbId ?objectName "I19.PMO" ?objectType "instance"
?checkerName "testLib/demo_top/Constr_3" ?analysisName "tran" ?comment "waiving")

          maeCloseViolationDb(dbId)
     )
```

maeOpenViolationDb

```
maeOpenViolationDb(
     t_sessionName
     t_testName
     t_historyName
     )
     => x_dbId / nil
```

Description

Opens a connection to the checks and asserts database for the given maestro cellview and returns a unique ID for the connection. You can use this ID to read or modify the database and its attached waiver SQL database using other SKILL functions. You must close this connection using maeCloseViolationDb to release the connection and complete the transactions on the database.

Note: Ensure that the history is closed before you run this function.

Arguments

x_sessionName	Name of the ADE Explorer or ADE Assembler session.
t_testName	Name of the test for which the results are to be opened.
t_historyName	Name of the history or child history. If a run saves child histories, specify the name of the child history to open results. You can also use this argument to open the results of a history saved for a run in the run plan.

Value Returned

x_dbId	ID of the open database connection
nil	Unsuccessful operation

Example

The following example opens a database, performs a task on it, and then closes it:

```
let( (dbId)
     dbId = maeOpenViolationDb( "fnxSession0" "demo_top:1" "Interactive.1")
```

```
     when( dbId != -1
          maeWaiveViolation( dbId ?objectName "I19.PMO" ?objectType "instance"
?checkerName "testLib/demo_top/Constr_3" ?analysisName "tran" ?comment "waiving")

          maeCloseViolationDb(dbId)
     )
```

maeWaiveViolation

```
maeWaiveViolation(
     x_dbId
     ?objectName t_objectName
     ?objectType t_objectType
     ?checkerName t_checkerName
     ?analysisName t_analysisName
     ?time t_time
     ?comment t_comment
     )
     => t / nil
```

Description

Adds to the waivers SQL database a rule to waive a check and assert for the given object.

Arguments

x_dbId — An integer identifying the connection for an open checks and asserts database.

?objectName *t_objectName*

Name of the net or instance object for which you want to waive a violation.

Note: Specify the instance or net name in a format that is used in the netlist, not as shown in the *Inst/Net* column of the Checks/Asserts results table. For example, if an instance name is `I15/I1/NM0`, you should write it as `I15.I1.NM0`.

?objectType *t_objectType*

Type of the object for which you are waiving a violation.

?checkerName *t_checkerName*

Name of the checker in hierarchical format.

?analysisName *t_analysisName*

Name of the analysis.

?time *t_time* — A double or a string value with units.

?comment *t_comment* — A string value indicating the reason for providing a waiver.

Note: You must provide a value for this argument.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

The following example opens a database, adds a waiver to the database, and then closes it:

```
let( (dbId)
     dbId = maeOpenViolationDb( "fnxSession0" "demo_top:1" "Interactive.1")
     when( dbId != -1
          maeWaiveViolation( dbId ?objectName "I19.PMO" ?objectType "instance"
?checkerName "testLib/demo_top/Constr_3" ?analysisName "tran" ?comment "waiving")

          maeCloseViolationDb(dbId)
     )
```

maeUnWaiveViolation

```
maeUnWaiveViolation(
     x_dbId
     ?objectName t_objectName
     ?objectType t_objectType
     ?checkerName t_checkerName
     ?analysisName t_analysisName
     ?time t_time
     )
     => t / nil
```

Description

Removes from the waivers SQL database a waive rule matching the given criteria.

Note: Ensure that the history is closed before you run this function.

Arguments

x_dbId
: An integer identifying the connection for an open checks and asserts database.

?objectName *t_objectName*
: Name of the net or instance object for which you want to waive a violation.

  Note: Specify the instance or net name in a format that is used in the netlist, not as shown in the *Inst/Net* column of the Checks/Asserts results table. For example, if an instance name is `I15/I1/NM0`, you should write it as `I15.I1.NM0`.

?objectType *t_objectType*
: Type of the object for which you are waiving a violation.

?checkerName *t_checkerName*
: Name of the checker in hierarchical format.

?analysisName *t_analysisName*
: Name of the analysis.

?time *t_time*
: A double or a string value with units.

Value Returned

`t`	Successful operation
`nil`	Unsuccessful operation

Example

The following example opens a database, removes a waiver database, and then closes it:

```
axlSession = axlGetWindowSession(hiGetCurrentWindow())

dbId = maeOpenViolationDb(axlSession "test_demoTop:1" "Interactive.3")

when( dbId
   ;;
   maeUnWaiveViolation(dbId ?objectName "I8.I0.MP1" ?objectType "instance"
?checkerName "testLib/demo_top/AssertionCheck" ?analysisName "tran" ?time
"101.1p")

   maeCloseViolationDb(dbId)
)
```

Functions to Work with Plotting Templates

SKILL functions to work with plotting templates in maestro cellviews

Function	Description
maeGetAllPlottingTemplates	Returns a list of all plotting templates saved in the given session.
maePlotWithPlottingTemplate	Plots the results for the given history using the specified plotting template.
maeSaveImagesUsingPlottingTemplate	Plots and saves the results for the given history using the specified plotting template.

Video

You can watch a video demonstration of these functions at SKILL Functions to Work with Plotting Templates.

maeGetAllPlottingTemplates

```
maeGetAllPlottingTemplates(
     [ ?session t_sessionName ]
     )
     => l_templates / nil
```

Description

Returns a list of all plotting templates saved in the given session.

Arguments

`?session` *`t_sessionName`*

Name of the maestro session.

Default: Current session.

Value Returned

`l_templates`	Returns a list of templates. Each list item contains the template name, history name, time stamp, and a description for the template.
`nil`	Returns `nil` if there is an error or no plotting template is available.

Example

The below example shows how you can get a list of templates and use those for plotting for results saved in a history.

```
maeGetAllPlottingTemplates()
=> (("p2" "Interactive.29" "May 14 12:19:12 2019" "")
  ("p1" "Interactive.29" "May 14 12:19:58 2019" "test")
)
```

```
maePlotWithPlottingTemplate(?history "Interactive.29" ?name "templateName")
=> (window:33)
```

maePlotWithPlottingTemplate

```
maePlotWithPlottingTemplate(
     [ ?session t_sessionName ]
     [ ?history t_historyName ]
     [ ?replaceMode t_replaceMode ]
     [ ?name t_templateName ]

     )
     => l_windowsPlotted / nil
```

Description

Plots the results for the given history using the specified plotting template.

Arguments

`?session` *`t_sessionName`*

Name of the maestro session.

Default: Current session.

`?history` *`t_historyName`*

Name of the history for which the results are to be plotted

Default: History saved for the last run.

`?replaceMode` *`t_replaceMode`*

Plotting mode.

Possible values:

- `t`: Specifies that the currently open window should be replaced with the plotted template.
- `nil`: Specifies that the template should open in a new window.

Default: `nil`

`?name` *`t_name`* Name of the template to be used for plotting.

Value Returned

`l_windowsPlotted`	Returns a list of windows plotted with the given template.
`nil`	Returns `nil` if there is an error.

Examples

The below example shows how you can get a list of templates and use those for plotting for results saved in a history.

```
maeGetAllPlottingTemplates()
=> (("p2" "Interactive.29" "May 14 12:19:12 2019" "")
   ("p1" "Interactive.29" "May 14 12:19:58 2019" "test")
)
```

maePlotWithPlottingTemplate(?history "Interactive.29" ?name "p1")
=> (window:33)

maeSaveImagesUsingPlottingTemplate

```
maeSaveImagesUsingPlottingTemplate(
     [ ?session t_sessionName ]
     [ ?history t_historyName ]
     [ ?replaceMode t_replaceMode ]
     [ ?name t_templateName ]
     [ ?dir t_dirPath ]
     [ ?saveEachSubwindowSeparately t_saveEachSubwindowSeparately ]
     )
     => t / nil
```

Description

Plots and saves the results for the given history using the specified plotting template.

Arguments

`?session` *`t_sessionName`*

Name of the maestro session.

Default: Current session.

`?history` *`t_historyName`*

Name of the history for which the results are to be plotted

Default: History saved for the last run.

`?replaceMode` *`t_replaceMode`*

Plotting mode.

Possible values:

- `t`: Specifies that the currently open window should be replaced with the plotted template.
- `nil`: Specifies that the template should open in a new window.

Default: `nil`

`?name` *`t_name`* Name of the template to be used for plotting.

`?dir` *`t_dirPath`* Path to the directory in which plots are to be saved.

`?saveEachSubwindowSeparately` *`t_saveEachSubwindowSeparately`*

Specifies whether to save each subwindow in a separate image.

Possible values: `t` or `nil`

Default: `nil`

Value Returned

`t`	Returns t if the windows are successfully plotted
`nil`	Returns `nil` if there is an error.

Example

The below example shows how you can get a list of templates and use those for plotting for results saved in a history.

```
maeGetAllPlottingTemplates()
=> (("p2" "Interactive.28" "May 14 12:19:12 2019" "")
   ("p1" "Interactive.29" "May 14 12:19:58 2019" "test")
)
```

maeSaveImagesUsingPlottingTemplate(?history "Interactive.29" ?replaceMode nil ?name "p1" ?dir "." ?saveEachSubwindowSeparately t)

=> t

`; This function plots the templates and saves the windows as images, for example,` **window:33.png**`, in the current project directory.`

Functions for Sensitivity Analysis Setup

You can use the following SKILL functions to manage the setup in the Sensitivity form for Sensitivity Analysis in ADE Assembler:

Function	Description
maeSensDeleteModel	Deletes the specified model from the setup for Sensitivity Analysis in the current or the specified maestro session.
maeSensDeleteModelGroup	Deletes all the model groups specified in the Model Groups setting on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensDeleteParameter	Deletes the specified parameter from the setup for Sensitivity Analysis in the current or the specified maestro session.
maeSensDeleteVar	Deletes the specified variable from the setup for Sensitivity Analysis in the current or the specified maestro session. You can also use this function to delete the value for Temperature.
maeSensEnableDesignVariation	Selects or clears the Enable Design and PVT Variation check box on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensEnableStatVariation	Selects or clears the Enable Variation of Statistical Parameters check box on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensGetModel	Returns a list of valid values and the nominal value for the specified model from the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensGetModelGroup	Returns a list of values specified in the Model Groups setting on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Function	Description
maeSensGetModels	Returns a list containing the names of all enabled models on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensGetParameter	Returns a list of valid values and the nominal value for the specified parameter from the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensGetParameters	Returns a list of all enabled parameters on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensGetVar	Returns a list of valid values and the nominal value for the specified global variable from the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensGetVars	Returns a list containing the names of all enabled global variables, including Temperature, on the Sensitivity form for the Sensitivity Analysis run mode in the current or the specified maestro session.
maeSensSetMethod	Sets the value in the Method field on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensSetModel	Adds a model on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensSetModelGroup	Adds a model group on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.
maeSensSetParameter	Adds a parameter on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session. If a parameter already exists with the given name, its value is updated.

Function	Description
maeSensSetVar	Adds a global variable to the run options for the Sensitivity Analysis run mode in the current or the specified maestro session. If a global variable already exists with the specified name, its value is updated. You can also use this function to change the value of `Temperature`.

maeSensDeleteModel

```
maeSensDeleteModel(
     t_modelName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Deletes the specified model from the setup for Sensitivity Analysis in the current or the specified maestro session.

Arguments

t_modelName	Name of the model to be deleted.
?session *t_sessionName*	Name of the maestro session. Default: Current session.

Value Returned

t	If the model was successfully deleted.
nil	If the model was not deleted. Review the errors in the log.

Example

The following example shows how to delete a model from the run options:

```
maeSensDeleteModel("gpdk045.scs")
=> t
```

maeSensDeleteModelGroup

```
maeSensDeleteModelGroup(
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Deletes all the model groups specified in the *Model Groups* setting on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

`?session` *`t_sessionName`*

Name of the maestro session.

Default: Current session.

Value Returned

`t`	If the model group was successfully deleted.
`nil`	If the model group was not deleted. Review the errors in the log.

Example

The following example shows how to delete all the model groups from the run options:

```
maeSensDeleteModelGroup()
=> t
```

maeSensDeleteParameter

```
maeSensDeleteParameter(
     t_parameterName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Deletes the specified parameter from the setup for Sensitivity Analysis in the current or the specified maestro session.

Arguments

t_parameterName	Name of the parameter. The parameter name must contain at least five non-empty string values, each separated by a `/` in the following format: *library-name/cell-name/view-name/instance-name/property-name* For hierarchical parameters, specify the hierarchical path in *instance-name*. For example, `mylib/mycell/view1/instance0/instance1/instance2/length`.
`?session` *t_sessionName*	Name of the maestro session. Default: Current session.

Value Returned

`t`	If the parameter was successfully deleted.
`nil`	If the parameter was not deleted. Review the errors in the log.

Example

The following example delete the specified parameter from the run options:

```
maeSensDeleteParameter("minimal/cm/schematic/NM0/fw")
=> t
```

maeSensDeleteVar

```
maeSensDeleteVar(
     t_varName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Deletes the specified variable from the setup for Sensitivity Analysis in the current or the specified maestro session. You can also use this function to delete the value for *Temperature*.

Arguments

t_varName	Name of the variable to be deleted. To delete the value for temperature, specify `"temperature"`.
`?session` *t_sessionName*	Name of the maestro session. Default: Current session.

Value Returned

`t`	Returns `t` if the variable is successfully deleted.
`nil`	Returns `nil` if there is an error.

Example

The following example delete the specified variable from the run options:

```
maeSensDeleteVar("vdd")
=> t
```

maeSensEnableDesignVariation

```
maeSensEnableDesignVariation(
     g_status
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Selects or clears the *Enable Design and PVT Variation* check box on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

g_status	Selection status for the check box. Specify `t` to select the check box. Otherwise, specify `nil`.
`?session` *t_sessionName*	Name of the maestro session. Default: Current session.

Value Returned

`t`	Returns `t` if the status is successfully changed.
`nil`	Returns `nil` if there is an error.

Example

The following example shows how to clear the *Enable Design and PVT Variation* check box on the Sensitivity form in the active setup:

```
maeSensEnableDesignVariation(nil)
=> t
```

maeSensEnableStatVariation

```
maeSensEnableStatVariation(
     g_status
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Selects or clears the *Enable Variation of Statistical Parameters* check box on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

g_status	Selection status for the check box. Specify `t` to select the check box. Otherwise, specify `nil`.
`?session` *t_sessionName*	Name of the maestro session. Default: Current session.

Value Returned

`t`	Returns `t` if the status is successfully changed.
`nil`	Returns `nil` if there is an error.

Example

The following example shows how to clear the *Enable Variation of Statistical Parameters* check box on the Sensitivity form in the active setup:

```
maeSensEnableStatVariation(nil)
=> t
```

maeSensGetModel

```
maeSensGetModel(
     t_modelName
     [ ?session t_sessionName ]
     )
     => l_modelDetails / nil
```

Description

Returns a list of valid values and the nominal value for the specified model from the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

t_modelName	Name of the model for which the values are to be returned.
?session *t_sessionName*	Name of the maestro session. Default: Current session.

Value Returned

l_modelDetails	Returns a list containing two items: ■ A list of valid values for the given model ■ The nominal value specified for the given model
nil	Returns nil if there is an error.

Example

The following example returns the details of the specified model:

```
maeSensGetModel("gpdk045.scs")
=> ("ff mc dd" "ff")
```

maeSensGetModelGroup

```
maeSensGetModelGroup(
     [ ?session t_sessionName ]
     )
     => l_modelGroupDetails / nil
```

Description

Returns a list of values specified in the *Model Groups* setting on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

`?session` *`t_sessionName`*

Name of the maestro session.

Default: Current session.

Value Returned

`l_modelGroupDetails`

Returns a list containing two items:

- A list of model group names specified in the *Value* column
- The name of the model group specified in the *Nominal* column

`nil`

Returns `nil` if there is an error.

Example

The following example shows how to get and print the model groups specified in the setup for Sensitivity Analysis:

```
;; get and print details of the model group

mgroup = (maeSensGetModelGroup)
=> ("mg1 mg2" "mg1")

(when mgroup
(printf "Sensitivity Model Group value = %s, nominal value = %s\n" (car mgroup)
(cadr mgroup))
)
=> Sensitivity Model Group value = mg1 mg2, nominal value = mg1
=> t
```

maeSensGetModels

```
maeSensGetModels(
     [ ?session t_sessionName ]
     )
     => l_modelNames / nil
```

Description

Returns a list containing the names of all enabled models on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

`?session` *`t_sessionName`*

Name of the maestro session.

Default: Current session.

Value Returned

`l_modelNames`	Returns a list of all enabled models.
`nil`	Returns `nil` if there is an error.

Example

The following example returns a list of names of enabled models on the Sensitivity form:

```
maeSensGetModels()
=> ("gpdk045.scs" "gpdk045_resistor.scs")
```

The following example deletes the existing models from the Sensitivity form, adds two new models, uses the `maeSensGetModels` function to get all enabled models, and then prints their details:

```
;models
models = (maeSensGetModels)  ;returns the model display names not paths
(foreach m models
     (maeSensDeleteModel m)
)
(maeSensSetModel "spectre/gpdk045.scs" "ss tt ff" ?nominalValue "tt")
;path is absolute or relative to include path
(maeSensSetModel "models.scs" "mc" ?nominalValue "mc")
;path is absolute or relative to include path
models = (maeSensGetModels)  ;returns the model display names not paths
```

```
=> ("cds_ff.scs" "cds_ff_res.scs")
(printf "Sensitivity models = %L\n" models)

(foreach m models
modelval = (maeSensGetModel m)
(printf "Model %s value = %s, nominal value = %s\n" m (car modelval) (cadr
modelval))

=> Model cds_ff.scs value = ff ss fs, nominal value = ff
Model cds_ff_res.scs value = ff_res1 ss_res, nominal value = ss_res
```

maeSensGetParameter

```
maeSensGetParameter(
     t_parameterName
     [ ?session t_sessionName ]
     )
     => l_paramDetails / nil
```

Description

Returns a list of valid values and the nominal value for the specified parameter from the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

t_parameterName

Name of the parameter. The parameter name must contain at least five non-empty string values, each separated by a `/` in the following format:

library-name/*cell-name*/*view-name*/*instance-name*/*property-name*

For hierarchical parameters, specify the hierarchical path in *instance-name*. For example, `mylib/mycell/view1/instance0/instance1/instance2/length`.

`?session` *t_sessionName*

Name of the maestro session.

Default: Current session.

Value Returned

l_paramDetails

Returns a list containing two items:

- A list of valid values for the given parameter
- The nominal value specified for the given parameter

`nil`

Returns `nil` if there is an error.

Example

The following example returns the values set for the given parameter:

```
maeSensGetParameter("minimal/cm/schematic/NM0/fw")
=> ("120n:10n:400n" "120n")
```

maeSensGetParameters

```
maeSensGetParameters(
     [ ?session t_sessionName ]
     )
     => l_parameterNames / nil
```

Description

Returns a list of all enabled parameters on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

`?session` *`t_sessionName`*

Name of the maestro session.

Default: Current session.

Value Returned

`l_parameterNames`	Returns a list containing the names of enabled parameters.
`nil`	Returns `nil` if there is an error.

Example

The following example returns a list of names of enabled parameters on the Sensitivity form:

```
maeSensGetParameters()

=> ("minimal/cm/schematic/NM0/fw" "Two_Stage_Opamp/AmpIn/schematic/M3/fingers"
"Two_Stage_Opamp/AmpIn/schematic/M3/w")
```

maeSensGetVar

```
maeSensGetVar(
     t_varName
     [ ?session t_sessionName ]
     )
     => l_varDetails / nil
```

Description

Returns a list of valid values and the nominal value for the specified global variable from the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

t_varName	Name of the global variable. Specify "`temperature`" to return the values set for the *Temperature* setting.
`?session` *t_sessionName*	Name of the maestro session. Default: Current session.

Value Returned

l_varDetails	Returns a list containing two items: ■ A list of valid values for the given variable ■ The nominal value specified for the given variable
`nil`	Returns `nil` if there is an error.

Example

The following example shows how to get the value set for *Temperature* and the variable `rload`:

```
maeSensGetVar("temperature")
=> ("-40 70 20" "20")
```

```
maeSensGetVar("rload")
=> ("1K 2K" "1K")
```

maeSensGetVars

```
maeSensGetVars(
     [ ?session t_sessionName ]
     )
     => l_varNames / nil
```

Description

Returns a list containing the names of all enabled global variables, including *Temperature*, on the Sensitivity form for the Sensitivity Analysis run mode in the current or the specified maestro session.

Arguments

`?session` *`t_sessionName`*

Name of the maestro session.

Default: Current session.

Value Returned

`l_varNames`	Returns a list containing the names of enabled global variables.
`nil`	Returns `nil` if there is an error.

Example

The following example returns the names of all global variables in the active setup:

```
maeSensGetVars()
=> ("temperature" "cload" "rload")
```

maeSensSetMethod

```
maeSensSetMethod(
     t_methodName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Sets the value in the *Method* field on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

t_method

Name of the method to be set.

Possible Values:

- `"Hammersley"`
- `"OFAT Sweep"`
- `"OFAT 3-level"`

`?session` *t_sessionName*

Name of the maestro session.

Default: Current session.

Value Returned

`t`	Returns t if the method is set successfully.
`nil`	Returns `nil` if there is an error.

Example

The following example shows how to set the method:

```
maeSensSetMethod("OFAT Sweep")
=> t
```

maeSensSetModel

```
maeSensSetModel(
     t_modelName
     g_modelValue
     [ ?nominalValue g_nominalValue ]
     [ ?session t_sessionName ]
     [ ?modelTest t_modelTest ]
     [ ?modelblock t_modelBlock ]
     )
     => t / nil
```

Description

Adds a model on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

t_modelName Name of the model.

g_modelValue Value range, which is the names of sections to be used from the specified model.

?nominalValue *g_nominalValue*

Nominal value, which is the section name to be used for the nominal run.

?session *t_sessionName*

Name of the maestro session.

Default: Current session.

?modelTest *t_modelTest*

Name of the test for which the model is to be used.

Default: `"All"`, which specifies that the model is associated with all the tests in the cellview.

?modelBlock *t_modelBlock*

Name of the block for which the model is to be used.

Default: `"Global"`, which specifies that the model is associated with all the design blocks.

Value Returned

`t`	Returns t if the model is set successfully.
`nil`	Returns `nil` if there is an error.

Example

The following example shows how to set the model `gpdk045.scs`. Sections `ff` and `mc` are used by all the tests. Section `mc` is used for the nominal run:

```
maeSensSetModel("gpdk045.scs" "ff mc" ?nominalValue "mc")
=> t
```

maeSensSetModelGroup

```
maeSensSetModelGroup(
     g_modelGroupValue
     [ ?nominalValue g_nominalValue ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Adds a model group on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session.

Arguments

g_modelValue
: Value range, which is the names of model groups.

?nominalValue *g_nominalValue*
: Nominal value, which is the name of model group to be used for the nominal run.

?session *t_sessionName*
: Name of the maestro session.

  Default: Current session.

Value Returned

`t`
: Returns t if the model group is successfully set.

`nil`
: Returns `nil` if there is an error.

Example

The following example shows how to set the model group:

```
maeSensSetModelGroup("modelGroupA modelGroupB" ?nominalValue "modelGroupA")
=> t
```

maeSensSetParameter

```
maeSensSetParameter(
     t_parameterName
     g_parameterValue
     [ ?nominalValue g_nominalValue ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Adds a parameter on the Sensitivity form for Sensitivity Analysis in the current or the specified maestro session. If a parameter already exists with the given name, its value is updated.

Arguments

t_parameterName

Name of the parameter. The parameter name must contain at least five non-empty string values, each separated by a `/` in the following format:

library-name/*cell-name*/*view-name*/*instance-name*/*property-name* to be modified.

For hierarchical parameters, specify the hierarchical path separated by `/` in *instance-name*. For example, `mylib/mycell/view1/instance0/instance1/instance2/length`.

g_parameterValue

Value range for the parameter.

?nominalValue *g_nominalValue*

Nominal value for the parameter.

?session *t_sessionName*

Name of the maestro session.

Default: Current session.

Value Returned

`t`

Returns t if the parameter is set successfully.

`nil`	Returns `nil` if there is an error.

Example

The following example shows how to set parameters:

```
maeSensSetParameter("minimal/cm/schematic/NM0/fw" "100n:10n:140n")
=> t

maeSensSetParameter("minimal/cm/schematic/NM0/fw" "100n:10n:140n" ?nominalValue
"120n")
=> t
```

maeSensSetVar

```
maeSensSetVar(
     t_varName
     g_varValue
     [ ?nominalValue g_nominalValue ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Adds a global variable to the run options for the Sensitivity Analysis run mode in the current or the specified maestro session. If a global variable already exists with the specified name, its value is updated. You can also use this function to change the value of *Temperature*.

Arguments

t_varName	Name of the global variable or `"temperature"`.
g_varValue	Value range for the variable or *Temperature*.
`?nominalValue` *g_nominalValue*	Nominal value.
`?session` *t_sessionName*	Name of the maestro session. Default: Current session.

Value Returned

`t`	Returns t if the variable is set successfully.
`nil`	Returns `nil` if there is an error.

Example

The following example shows how to set the values for global variables `rload` and `vdd`:

```
maeSensSetVar("rload" "50 70")
=> t

maeSensSetVar("vdd" "1 2 3" ?nominalValue "1")
=> t
```

Functions for XML File Management

SKILL functions to work with XML files saved for maestro cellviews

Function	Description
sevOpenXmlFile	Opens the specified XML state file. If the specified file does not exist, this function creates a new file.
sevConvertStateFormat	Translates all ADE XL states from normal to XML format in the current directory. This function can also be used to do translate a specific parameter set. You can use this function in the OCEAN script and perform the translation in a batch.
sevCloseXmlFile	Closes the specified XML state handle.
sevWriteTable	Writes the component table in the specified XML state file.
sevReadTable	Returns the component table from the specified XML state file.
sevWriteValue	Writes a XML component value in the specified XML state file.
sevReadValue	Reads the value of an XML component from the specified XML state file.

sevOpenXmlFile

```
sevOpenXmlFile(
     t_stateFile
     )
     => x_stateHandle / nil
```

Description

Opens the specified XML state file. If the specified file does not exist, this function creates a new file.

Arguments

x_stateFile	XML state file that also includes your home directory.

Value Returned

x_stateHandle	Returns the handler, which should be an integer greater than 0, for the specified XML state file.
`nil`	Returns `nil` if there is an error.

Example

The below example shows how you can use the sevOpenXmlFile function to open the XML state file.

```
directory = "/home/user"
// Specifies the user's directory
stateFile = strcat(directory "/xml.state")
stateHandle = sevOpenXmlFile(stateFile)
// The above function statements returns a handle for the XML state file
```

sevConvertStateFormat

```
sevConvertStateFormat(
     [ ?libs l_libs ]
     [ ?stateDir t_stateDir ]
     [ ?viewType t_viewType ]
     [ ?format t_format ]
     [ ?backup g_backup ]
     [ ?recover g_recover ]
     [ ?autoCheckin g_autoCheckin ]
     [ ?popUpWarnWin g_popUpWarnWin ]
     => t / nil
```

Description

Translates all ADE XL states from normal to XML format in the current directory. This function can also be used to do translate a specific parameter set. You can use this function in the OCEAN script and perform the translation in a batch.

Arguments

?libs l_libs	Library for which the state is to be converted. For example, `evaluation`, `list("evaluation" "tes")` or `all`. If you do not specify a library, all libraries under current directory are searched and converted. For read-only states, a warning message is printed. **Note:** For ADE XL, the search is based on data.sdb. For state view, search is based on ADE_state.info.
?stateDir t_stateDir	State directories that are not under tool default search directory. It works when *viewType* is set to `stateview`. It is used to convert state file for ADE L state.
?viewType *t_viewType*	Sets the state property Possible values: `adexl`, `stateview`, and `all`. If no view type is specified, `adexl` view is used. Default value: `adexl`
?format *t_format*	Specify the format in which the state is to be converted Possible values: `xml` and `files`. Default value: `xml`.
?backup *g_backup*	Sets the directory used to backup the original state. Possible values: `t` or `nil` Default value: `nil` When this argument is set to `t`, the function creates a backup library of the directory and files, such as `data.sdb`, `test_states`, that are present in the view to be changed.
?recover *g_recover*	Recovers the state automatically with the latest backup data. When both backup and recover arguments are set to `t`, backup is ignored. Default value: `nil`.
?autoCheckin *g_autoCheckin*	Checks in view automatically under the DM environment. Default value: `nil`.
?popUpWarnWin *g_popUpWarnWin*	Controls the display of the warning message when the state starts getting converted. It can be used in converting state by script. Possible values: `t` or `nil`. When set to `t`, the warning message is not displayed.

Value Returned

t	Returns t when the function runs successfully.
`nil`	Returns `nil` if there is an error.

Example

The below example shows how you can use the sevConvertStateFormat function with different argument settings:

```
sevConvertStateFormat()
```

Converts all `adexl` views under current directory from the `files` format to `xml` format.

```
sevConvertStateFormat(?libs "evaluation" ?format 'files ?backup
"/home/user/backup")
```

Converts the `adexl` view under `evaluation` library to `files` format and also creates a backup of original state under directory `/home/user/backup`.

```
sevConvertStateFormat(?libs "evaluation" ?format 'files ?backup
"/home/user/backup" ?recover t)
```

Recovers the `adexl` view under `evaluation` library from backup directory `/home/user/backup`

```
sevConvertStateFormat(?stateDir "/home/user/artist_state" ?viewType 'stateview )
```

Converts the ADE L state under directory `/home/user/artist_state` to the `xml` format.

```
sevConvertStateFormat(?libs "evaluation" ?autoCheckin t)
```

Converts the `adexl` view under `evaluation` library to `xml` format and check in the view automatically under DM environment.

sevCloseXmlFile

```
sevCloseXmlFile(
     t_stateFile
     )
```

Description

Closes the specified XML state handle.

Arguments

x_stateFile	State handle for the XML state file.

Value Returned

None.

Example

The below example shows how you can use the `sevCloseXmlFile` function to close the XML state file. This example includes a combination of open and close functions.

```
directory = "/home/user"
// Specifies the user's directory
stateFile = strcat(directory "/xml.state")
stateHandle = sevOpenXmlFile(stateFile)
// The above function statements returns a handle for the XML state file
...
sevCloseXmlFile(stateHandle)
// closes the given XML state handle.
```

sevWriteTable

```
sevWriteTable(
     x_stateHandle
     S_componentName
     o_table
     [ ?partition l_partitions ]
     )
     => t / nil
```

Description

Writes the component table in the specified XML state file.

Arguments

x_stateHandle	The handle of a state file.
t_componentName	Name of the XML component whose value you want to write in the table.
o_table	Table that contains information about the specified component.
?partitions *l_partitions*	List of partitions.

Value Returned

t	Returns t when function runs successfully.
nil	Returns nil if there is an error.

Example

The below example shows how you can use the sevWriteTable function to write the specified component table in a given XML state file.

```
directory = "/home/user"
stateFile = strcat(directory "/xml.state")
stateHandle = sevOpenXmlFile(stateFile)
// The above commands return the state file handle and open the XML state file.
table = makeTable('tableformat nil)
table['a] = list(nil)
table['b] = "a"
```

```
// The above commands creates the table and specify two arguments a and b of type
list and string respectively.
sevWriteTable(stateHandle "test" table)
// The above command writes the specified table for the component ‘test’.
```

When you view the contents of the XML state file, the following output appears:

```
<?xml version="1.0"?>
<statedb ICVersion="IC6.1.6-64b.ADE.634" version="5">xml
    <component Name="test" Type="skillTable">tableformat
        <field Name="b" Type="string">"a"</field>
        <field Name="a" Type="skillDpl"></field>
    </component>
</statedb>
```

sevReadTable

```
sevReadTable(
     x_stateHandle
     S_componentName
     o_table
     )
     => x_stateHandle / nil
```

Description

Returns the component table from the specified XML state file.

Arguments

x_stateHandle	The handler of a state file. Valid Values: Any integer value
t_componentName	Name of the XML component whose value you want to read from the table
o_table	Table that contains the information about the specified component.

Value Returned

`t`	Returns `t` when function runs successfully.
`nil`	Returns `nil` if there is an error.

Example

The below example shows how you can use the `sevReadTable` function to read the component table for a given XML state file.

```
stateFile = strcat(directory "/xml.state")
stateHandle = sevOpenXmlFile(stateFile)
//The above commands return the handle for the XML state file.
table = makeTable('tableformat nil)
//The above command creates a table.
sevReadTable(stateHandle "analyses" table)
// The above command reads the value of the component 'analyses' from the given
table for the specified XML state file handle.
```

sevWriteValue

```
sevWriteValue(
     x_stateHandle
     t_componentName
     t_attributeName
     t_compValue
     )
     => t / nil
```

Description

Writes a XML component value in the specified XML state file.

Arguments

x_stateHandle	The handler of a state file. Valid Values: Any integer value
t_componentName	Name of the XML component whose value you want to write.
t_attributeName	Name of the attribute of the specified component.
t_componentValue	Value you want to write for the specified component.

Value Returned

`t`	Returns `t` when function runs successfully.
`nil`	Returns `nil` if there is an error.

Example

The below example shows how you can use the sevWriteValue function to write the values of the arguments a and b of component test. A combination of write and read commands are used in this example.

```
directory = "/home/user"
stateFile = strcat(directory "/xml.state")
// The above commands return the handle of the XML state file.
stateHandle = sevOpenXmlFile(stateFile)
// The above commands opens the XML state file handle.
sevWriteValue(stateHandle "test" "a" list(nil))
sevWriteValue(stateHandle "test" "b" list('a))
```

```
// The above commands write the argument values ‘a’ and ‘b’for component ‘test’.
sevReadValue(stateHandle "test" "b")
// The above command reads the value for argument ‘b’ of component ‘test’.
sevCloseXmlFile(stateHandle)
// The above command closes the XML state file handle.
```

When you view the state file contents, the following output appears:

```
<?xml version="1.0"?>
<statedb ICVersion="IC6.1.6-64b.ADE.634" version="5">xml
    <test Name="a" Type="skillDpl"></test>
    <test Name="b" Type="list">(a)</test>
</statedb>
```

sevReadValue

```
sevReadValue(
     x_stateHandle
     t_componentName
     t_attributeName
     )
     => t_compValue / nil
```

Description

Reads the value of an XML component from the specified XML state file.

Arguments

x_stateHandle	The handler of a state file. Valid Values: Any integer value
t_componentName	Name of the XML component whose value you want to read.
t_attributeName	Name of the attribute of the specified component.

Value Returned

t_componentValue	Returns the value of the specified component.
`nil`	Returns `nil` if there is an error.

Example

The below example shows how you can use the sevReadValue command to read the values of a component test. A combination of write and read commands are used in this example.

```
directory = "/home/user"
stateFile = strcat(directory "/xml.state")
// The above commands return the handle of the XML state file.
stateHandle = sevOpenXmlFile(stateFile)
// The above commands opens the XML state file handle.
sevWriteValue(stateHandle "test" "a" list(nil))
sevWriteValue(stateHandle "test" "b" list('a))
// The above commands write the argument values ‘a’ and ‘b’for component ‘test’.
sevReadValue(stateHandle "test" "b")
```

```
// The above command reads the value for argument 'b' of component 'test'.
sevCloseXmlFile(stateHandle)
// The above command closes the XML state file handle.
```

Functions to Work with the Locally Scoped Models and Options (MTS Options)

Use the following functions to work with the MTS options:

Function	Description
`maeGetMTSMode`	Indicates whether the MTS options are enabled for the specified test.
`maeSetMTSMode`	Enables the MTS options for the specified test.
`maeGetMTSBlock`	Returns the requested MTS-related information about the specified library cell (block) or the instance.
`maeSetMTSBlock`	Enables local scoping for the specified library cell (block) or instance and sets the specified MTS options–models files to be scoped locally and the process parameters, such as `scale` and `temp` to be included in the simulation locally.

maeGetMTSMode

```
maeGetMTSMode(
     t_testName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Indicates whether the locally scoped models and options (MTS options) are enabled for the specified test.

Arguments

t_testName	Name of the test.
?session *t_sessionName*	Name of the session. **Note:** If you do not specify this argument, the currently active session is used.

Value Returned

t	Returns t if the MTS options are enabled for the specified test.
nil	Returns nil if the MTS options are disabled for the specified test or if there is an error.

Example

The following example indicates that the MTS options are enabled for the test_MTS test:

```
maeGetMTSMode("test_MTS")
=> t
```

maeSetMTSMode

```
maeSetMTSMode(
     t_testName
     [ ?session t_sessionName ]
     [ @rest g_mtsMode ]
     )
     => t / nil
```

Description

Enables the locally scoped models and options (MTS options) for the specified test.

Arguments

t_testName	Name of the test.
?session *t_sessionName*	Name of the session. **Note:** If you do not specify this argument, the currently active session is used.
@rest *g_mtsMode*	Boolean flag to specify whether to enable or disable the MTS options. Possible values: `t`: Enables the MTS options `nil`: Disables the MTS options Default value: `t`

Value Returned

`t`	Returns `t` if the MTS options are enabled for the specified test.
`nil`	Returns `nil` if there is an error.

Examples

The following example enables the MTS options for the `test_MTS` test:

```
maeSetMTSMode("test_MTS" t)
=> t
```

The following example enables the MTS options for the `test_MTS` test:

```
maeSetMTSMode("test_MTS")
=> t
```

The following example disables the MTS options for the `test_MTS` test:

```
maeSetMTSMode("test_MTS" nil)

=> t
```

maeGetMTSBlock

```
maeGetMTSBlock(
     t_testName
     [ ?lib t_lib ]
     [ ?cell t_cell ]
     [ ?inst t_instName ]
     [ ?type t_type ]
     [ ?session t_sessionName ]
     )
     => l_mtsBlockInformation / nil
```

Description

Returns the requested MTS-related information about the specified library cell (block) or the instance.

Arguments

t_testName	Name of the test.
?lib *t_lib*	Name of the library.
?cell *t_cell*	Name of the cell.
?inst *t_instName*	Name of the instance.
?type *t_type*	Type of information you want to print for the specified block or instance. Possible values: ■ `enable`: Returns `t` if local scoping is enabled for the specified block or instance. Returns `nil` if local scoping is disabled for the specified block or instance. ■ `modelFiles`: Returns the information related to the models files that are locally scoped for the specified block or instance. ■ `simOptions`: Returns the process parameters with their values that are included in the simulation locally for the specified block or instance. ■ `all`: Returns all the above information for the specified block or instance. Default value: `all`
?session *t_sessionName*	

Name of the session.

Note: If you do not specify this argument, the currently active session is used.

Value Returned

`t`	Returns `t` if the operation is successful.
`nil`	Returns `nil` if there is an error.

Examples

The following example returns all the MTS-related information for the library cell, `design_45 inv` in the `test_MTS` test.

```
maeGetMTSBlock("test_MTS" ?lib "design_45" ?cell "inv")
=>
(nil modelFiles (
    ("gpdk045/gpdk045_v_3_0/models/spectre/gpdk045.scs" "tt")
) simOptions (
    temp("40")
    (scale "5")
    (scalem "2")
)
enable t
)
```

The following example returns the values of the process parameters that are included in the simulation for the library cell, `design_45 inv` in the `test_MTS` test, locally:

```
maeGetMTSBlock("test_MTS" ?lib "design_45" ?cell "inv" ?type "simOptions")
=>
(temp("40")
    (scale "5")
    (scalem "2")
)
```

maeSetMTSBlock

```
maeSetMTSBlock(
     t_testName
     [ ?lib t_lib ]
     [ ?cell t_cell ]
     [ ?inst t_instName ]
     [ ?enable g_enable ]
     [ ?modelFiles l_modelFiles ]
     [ ?simOptions l_simOptions ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Enables local scoping for the specified library cell (block) or instance and sets the specified MTS options–models files to be scoped locally and the process parameters, such as `scale` and `temp` to be included in the simulation locally.

Arguments

t_testName	Name of the test.
?lib *t_lib*	Name of the library.
?cell *t_cell*	Name of the cell.
?inst *t_instName*	Name of the instance.
?enable *g_enable*	Boolean flag to specify whether to enable local scoping for the specified block or instance. Possible values: ■ `t`: Enables local scoping for the specified block or instance ■ `nil`: Disables local scoping for the specified block or instance Default value: `t`
?modelFiles *l_modelFiles*	List specifying the model files that will be locally scoped for the specified block or instance.
?simOptions *l_simOptions*	

List specifying the values of the process parameters that will be included in the simulation for the specified block or instance locally.

`?session` *`t_sessionName`*

Name of the session.

Note: If you do not specify this argument, the currently active session is used.

Value Returned

`t`	Returns `t` if the operation is successful.
`nil`	Returns `nil` if there is an error.

Examples

The following example enables local scoping for the library cell, `design_45 inv` in the `test_MTS` test. Additionally, it specifies that the `gpdk045.scs` model file is locally scoped for the specified library cell and the values of the process parameters, `temp` and `scale` are set to `30` and `5`, respectively. These parameters will be included in the simulation for the cell locally.

```
maeSetMTSBlock("test_MTS" ?lib "design_45" ?cell "inv" ?enable t ?modelFiles
list(list("gpdk045.scs" "tt")) ?simOptions list(list("temp" "30") list("scale"
"5")))
```

```
=> t
```

The following example enables local scoping for the instance, `I8` in the `test_MTS` test. Additionally, it specifies that the `gpdk045.scs` model file is locally scoped for the specified instance and the values of the process parameters, `temp` and `scale` are set to `30` and `5`, respectively. These parameters will be included in the simulation for the instance locally.

```
maeSetMTSBlock("test_MTS" ?inst "I8" ?enable t ?modelFiles list(list("gpdk045.scs"
"tt")) ?simOptions list(list("temp" "30") list("scale" "5")))
```

```
=> t
```

If you have another instance, `I0`, under the `I8` instance in your `test_MTS` test, the following example enables local scoping for the instance, `I0`:

```
maeSetMTSBlock("test_MTS" ?inst "I8/I0" ?enable t ?modelFiles
list(list("gpdk045.scs" "tt")) ?simOptions list(list("temp" "30") list("scale"
"5")))
```

```
=> t
```

The following example disables local scoping for the library cell, `design_45 inv` in the `test_MTS` test.

```
maeSetMTSBlock("test_MTS" ?lib "design_45" ?cell "inv" ?enable nil )
=> t
```

Function to Work with the Reliability Setup

Use the following function to work with the reliability setup:

Function	Description
calcValForRel	Retrieves the value of an output expression used in a reliability setup. You can use the value returned by this function in another output expression.
maeGetStressFile	Returns the path of the stress file to be reused from the specified reliability setup.
relxEnableFormTab	Enables or disables the specified tab in the Reliability Options form.
relxDisplayDiscField	Shows or hides the specified disclosure in the Reliability Options form.
relxEnableDiscField	Enables or disables the specified disclosure in the Reliability Options form.
relxHideAgeCalculationApproachField	Hides the Age Calculation Approach field from the Modeling tab in the Reliability Options form.
relxGetCustomTabName	Returns the name of the custom tab.
relxCustomizeDisplayOrEnableStatus	Defines the display and enable status of the disclosures, fields, and tabs.
relxCreateCustomizedTab	Adds a customized tab to the Reliability Options form.
relxAddSetupRelxOption	Adds new fields to the Reliability Options form.
asiFormatSpecialParameterForRel	Netlists the new options added to the customized tab.
relxInitOptionsInCdsenv	Creates environment variables for the customized options added to the Reliability Options form..

calcValForRel

```
calcValForRel(
     t_outputName
     [ ?relxName t_relxName ]
     [ ?ageValue t_ageValue ]
     [ ?ageUnit t_ageUnit ]
     [ ?she g_she ]
     [ ?cornerName t_cornerName ]
     [ ?historyName t_historyName ]
     [ ?run t_runName]
     )
     => g_output / nil
```

Description

Retrieves the value of an output expression used in a reliability setup. You can use the value returned by this function in another output expression.

Video

You can watch a video demonstration on how to use the `calcValForRel` function at Using the calcValForRel Function in Reliability Analysis. You can also read the related blog at Virtuoso Video Diary: Enhancements in Reliability Analysis.

Arguments

t_outputName
: Name of the output to be used. The output can return a scalar value or waveform.

?relxName *t_relxName*
: Name of the reliability setup to be used.

?ageValue *t_ageValue*
: Specifies one of the following values:

- `Fresh`: For the fresh stage
- `Stress`: For stress stage
- <*age_number*>: For aged stage (Aging simulation)

  Note: "*age_number*" specifies the age points at which the device degradation is calculated.

?ageUnit *t_ageUnit*

Specifies the unit for the age points.

Valid Values: `yr`, `year`, `years`, `d`, `day`, `days`, `h`, `hour`, `hours`, `min`, `minute`, `minutes`, `sec`, `second`, `seconds`.

Default Value: `year`

`?she` *`g_she`*

Boolean value to specify whether the self-heating effect is enabled.

Valid Values:

- `t`: Self-heating effect is enabled
- `nil`: Self-heating effect is disabled

Default Value: `nil`

`?cornerName` *`t_cornerName`*

Name of the corner. When specified, the value of the given output for this cornet is used.

Default Value: `nil`

`?historyName` *`t_historyName`*

Name of the history from which the results are to be retrieved. When specified, the scalar or waveform result of the output from the given history is used. Otherwise, the value of the output from the current simulation run is used.

The history must exist for the `calcValForRel` to return the expected result. To retrieve the value of an output of type '`signal`', the simulation waveform results must be saved.

Default Value: `nil`

`?run` *`t_runName`*

Name of the run in the run plan from which the value of the given output is to be returned. This argument is useful only when the setup has a run plan.

Use this argument to get the value of an output from the results of one run (in the run plan) and use it in a variable or expression for another run in the same run plan or history. It cannot be used to refer to the results of another history. When both `?run` and `?historyName` are specified, `?historyName` is ignored by `calcValForRel`.

Default Value: `nil`

Value Returned

g_output	Returns the value of the specified output used in the given reliability setup.
nil	Returns nil if there is an error.

Examples

Example 1

Retrieves the value of output yMax from "Stress" stage of the reliability setup "rel_0":

```
calcValForRel("yMax" ?relxName "rel_0" ?ageValue "stress")
```

Example 2

Retrieves the value of output yMax from C2 corner for the age point 10 year in the reliability setup "rel_0":

```
calcValForRel("yMax" ?relxName "rel_0" ?ageValue "10" ?cornerName "C2")
```

Example 3

Retrieves the value of output yMax from the reliability setup "rel_0" with the self-heating effect enabled.

```
calcValForRel("yMax" ?relxName "rel_0" ?she t)
```

Example 4

Retrieves the value of output yMax from the history "Interactive.3". The history contains the simulation results of fresh stage in the reliability setup "rel_0":

```
calcValForRel("yMax" ?relxName "rel_0" ?ageValue "fresh" ?historyName
"Interactive.3")
```

Example 5

Retrieves the value of output yMax for the age point 20 days of gradual aging (10, 20, 30) in the reliability setup "rel_0":

```
calcValForRel("yMax" ?relxName "rel_0" ?ageValue "20" ?ageUnit "day")
```

Example 6

Retrieves the value of output yMax from the reliability setup "rel_0" used in the run "Run.0":

```
calcValForRel("yMax" ?relxName "rel_0" ?run "Run.0")
```

maeGetStressFile

```
maeGetStressFile(
     t_relxSetupName
     [ ?cornerName t_cornerName ]
     [ ?historyName t_historyName ]
     [ ?run t_runName ]
     [ ?matchParams l_matchParams ]
     )
     => t_output / nil
```

Description

Returns the path of the stress file to be reused from the specified reliability setup.

Arguments

t_relxSetupName
: Name of the reliability setup from which the stress file is to be reused.

?cornerName *t_cornerName*
: Name of the corner from which the stress file is to be reused.

  If you have multiple corners in the test setup and you do not specify the corner name, the stress file generated for the first corner is reused.

  Default Value: `nil`

?historyName *t_historyName*
: Name of the history from which the stress file is to be reused.

  If you do not specify the history, the stress file from the history of the last simulation run is reused.

  Default Value: `nil`

?run *t_runName*
: Name of the run in the run plan for which the path of the stress file is to be reused. This argument is useful only when the setup has a run plan.

  Default Value: `nil`

?matchParams *l_matchParams*
: A list containing a list of name-value pairs of the sweep variables to identify the design point from which the stress file is to be reused.

Value Returned

t_output	Returns the path of the stress file to be reused.
nil	Returns nil if there is an error.

Examples

Example 1:

Returns the path of the stress file that is reused from the reliability setup Rel1:

```
maeGetStressFile("Rel1")
=>
"/servers/user/Reliability/simulation/library/cell/view/results/maestro/history/1
/Test/netlist/input.bs0"
```

Example 2:

Returns the path of the stress file that is reused from the C1 corner in the reliability setup Rel1:

```
maeGetStressFile("Rel1" ?cornerName "C1")
=>
"/servers/user/Reliability/simulation/library/cell/view/results/maestro/history/2
/Test/netlist/input.bs0"
```

Example 3:

Returns the path of the stress file that is reused from the history Interactive.2 for the reliability setup Rel1:

```
maeGetStressFile("Rel1" ?historyName "Interactive.2")
=>
"/servers/user/Reliability/simulation/library/cell/view/results/maestro/history/3
/Test/netlist/input.bs0"
```

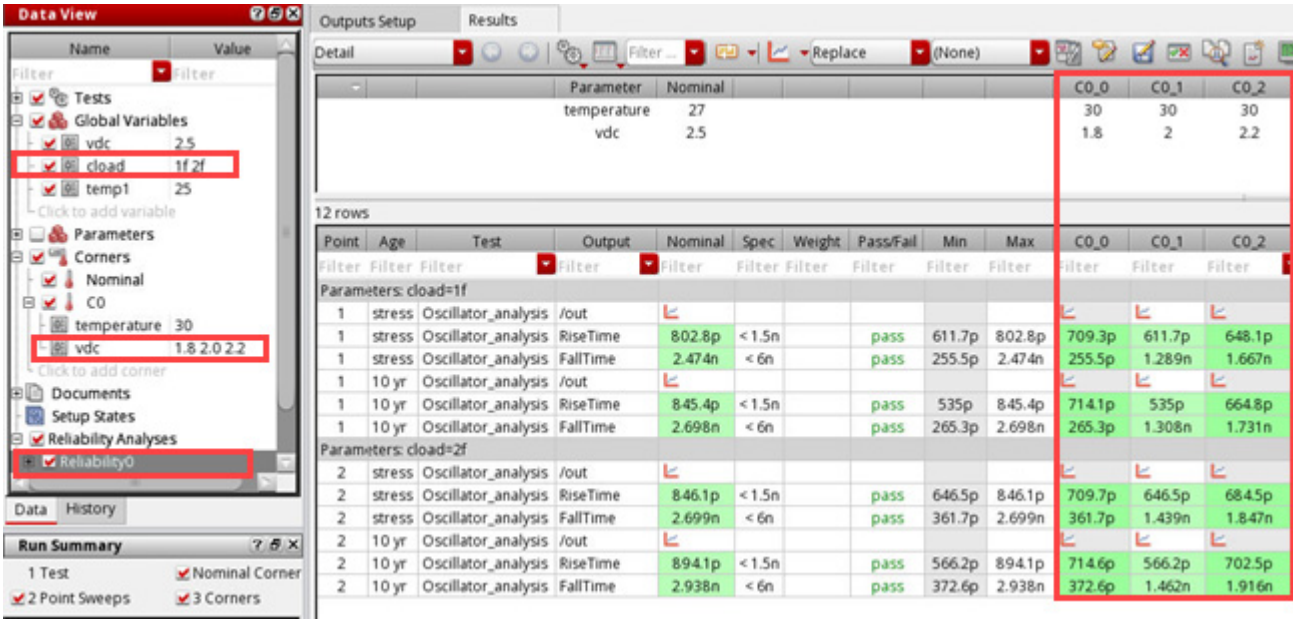
Example 4:

Returns the path of the stress file that is reused from the run Run.0 for the reliability setup Rel1:

```
maeGetStressFile("Rel1" ?run "Run.0")
=>
"/servers/user/Reliability/simulation/library/cell/view/results/maestro/history/4
/Test/netlist/input.bs0"
```

Example 5:

Consider the following setup in which the variable `cload` is a sweep variable. Note that the corner `C0` is also swept for different values of `vdc`.



When you run the reliability setup `Reliability0`, six stress files are generated for the following design points:

- `cload=1f`, `vdc=1.8` (`C0_0`)
- `cload=2f`, `vdc=1.8` (`C0_0`)
- `cload=1f`, `vdc=2.0` (`C0_1`)
- `cload=2f`, `vdc=2.0` (`C0_1`)
- `cload=1f`, `vdc=2.2` (`C0_2`)
- `cload=2f`, `vdc=2.2` (`C0_2`)

To identify the design point from which you want to reuse the stress file, use the `?matchParams` argument as follows:

```
maeGetStressFile("Reliability0" ?matchParams list(list("cload" "1f") list("vdc"
"1.8")))
maeGetStressFile("Reliability0" ?matchParams list(list("cload" "2f") list("vdc"
"1.8")))
maeGetStressFile("Reliability0" ?matchParams list(list("cload" "1f") list("vdc"
"2.0")))
```

```
maeGetStressFile("Reliability0" ?matchParams list(list("cload" "2f") list("vdc"
"2.0")))
maeGetStressFile("Reliability0" ?matchParams list(list("cload" "1f") list("vdc"
"2.2")))
maeGetStressFile("Reliability0" ?matchParams list(list("cload" "2f") list("vdc"
"2.2")))
```

relxEnableFormTab

```
relxEnableFormTab(
     r_formObject
     t_tabName
     g_enableBoolean
     )
     => t / nil
```

Description

Enables or disables the specified tab in the Reliability Options form.

Arguments

r_formObject	The form object: `relxOptionForm`.
t_tabName	Name of the tab to be enabled or disabled. Possible values are: ■ `Basic` ■ `Modeling` ■ `Degradation` ■ `Output`
g_enableBoolean	Specifies whether to enable or disable the specified tab. Possible values are: ■ `t`: Enables the specified tab. ■ `nil`: Disables the specified tab.

Value Returned

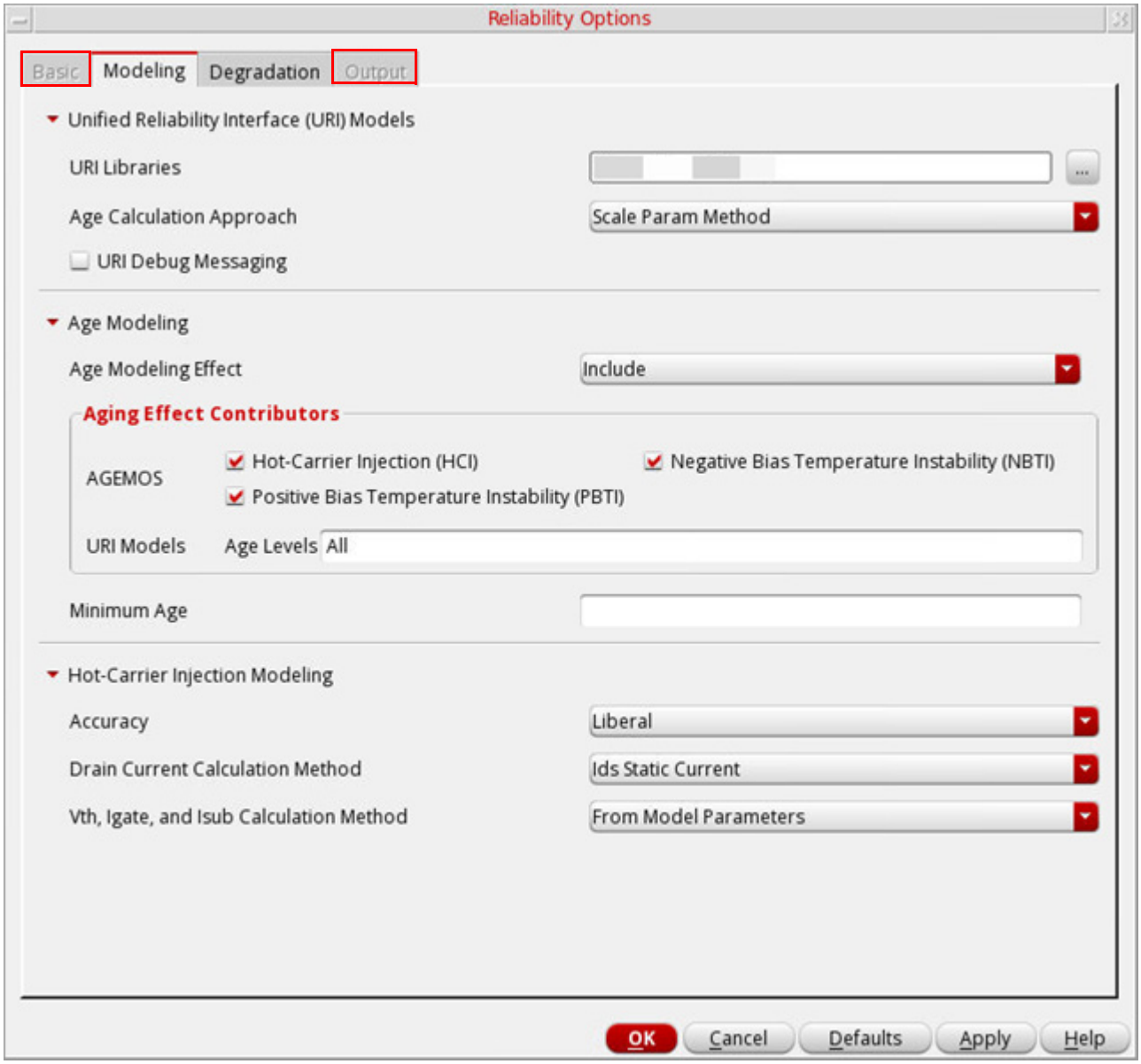
`t`	Returns `t` if the tab is enabled.
`nil`	Returns `nil` the tab is disabled.

Examples

To disable the *Basic* and *Output* tabs in the Reliability Options form, add the following SKILL code in the `.cdsinit` file:

```
(defun relxCustomizeDisplayOrEnableStatus (relxOptionForm)
    relxEnableFormTab(relxOptionForm "Basic" nil)
    relxEnableFormTab(relxOptionForm "Output" nil)
)
```

The *Basic* and *Output* tabs are disabled in the Reliability Options form:



relxDisplayDiscField

```
relxDisplayDiscField(
     r_formObject
     t_disclosureName
     g_displaybleBoolean
     )
     => t / nil
```

Description

Shows or hides the specified disclosure in the Reliability Options form.

Arguments

r_formObject	The form object: `relxOptionForm`.
t_disclosureName	Name of the disclosure to be shown or hidden. For example, `Age Modeling`
g_displayBoolean	Specifies whether to show or hide the specified disclosure. Possible values are: ■ `t`: Shows the disclosure. ■ `nil`: Hides the disclosure.

Value Returned

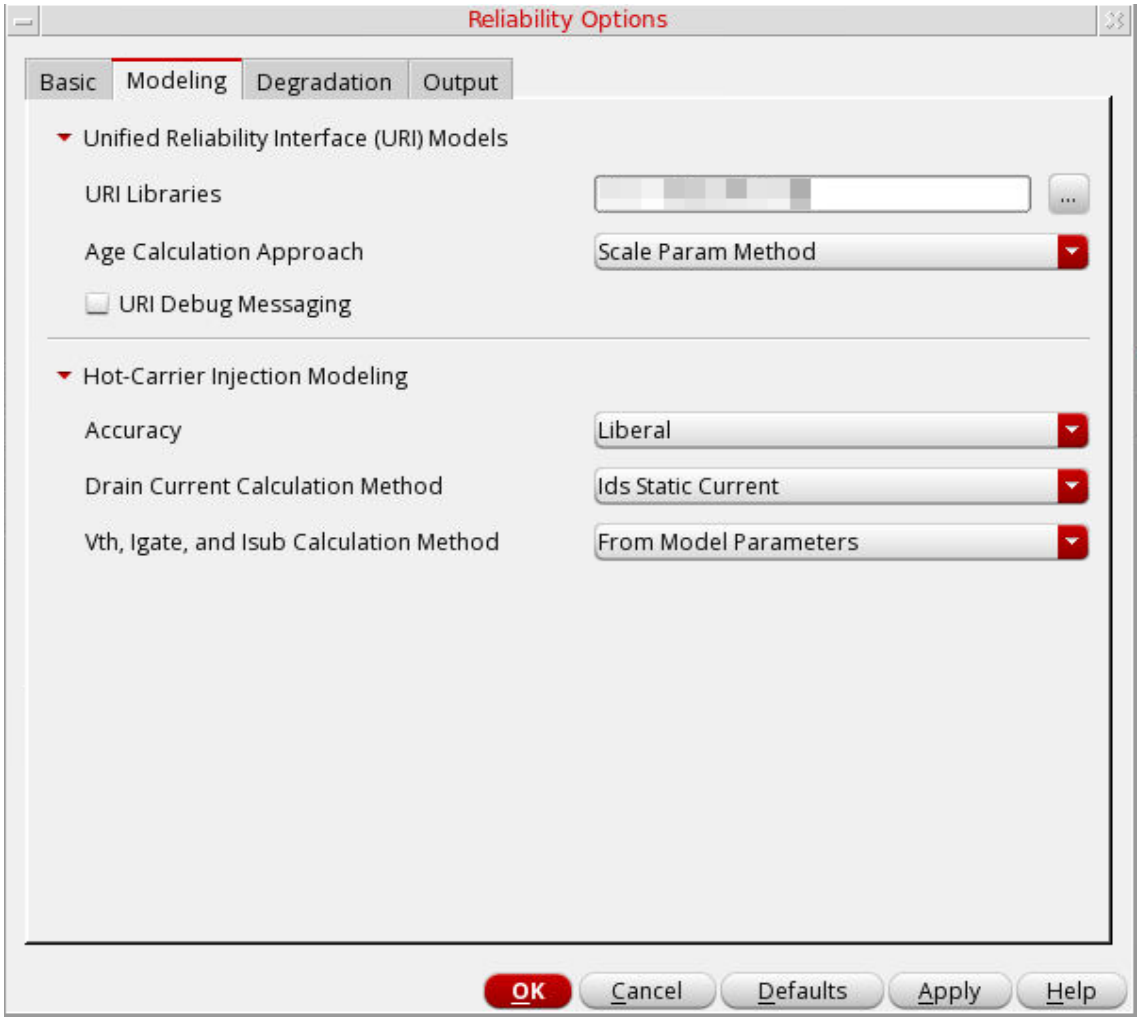
`t`	Returns `t` if the disclosure is shown.
`nil`	Returns `nil` the disclosure is hidden.

Examples

To hide the *Age Modeling* disclosure from the *Modeling* tab on the Reliability Options form, add the following SKILL code in the `.cdsinit` file:

```
(defun relxCustomizeDisplayOrEnableStatus (relxOptionForm)
    relxDisplayDiscField(relxOptionForm "Age Modeling" nil)
)
```

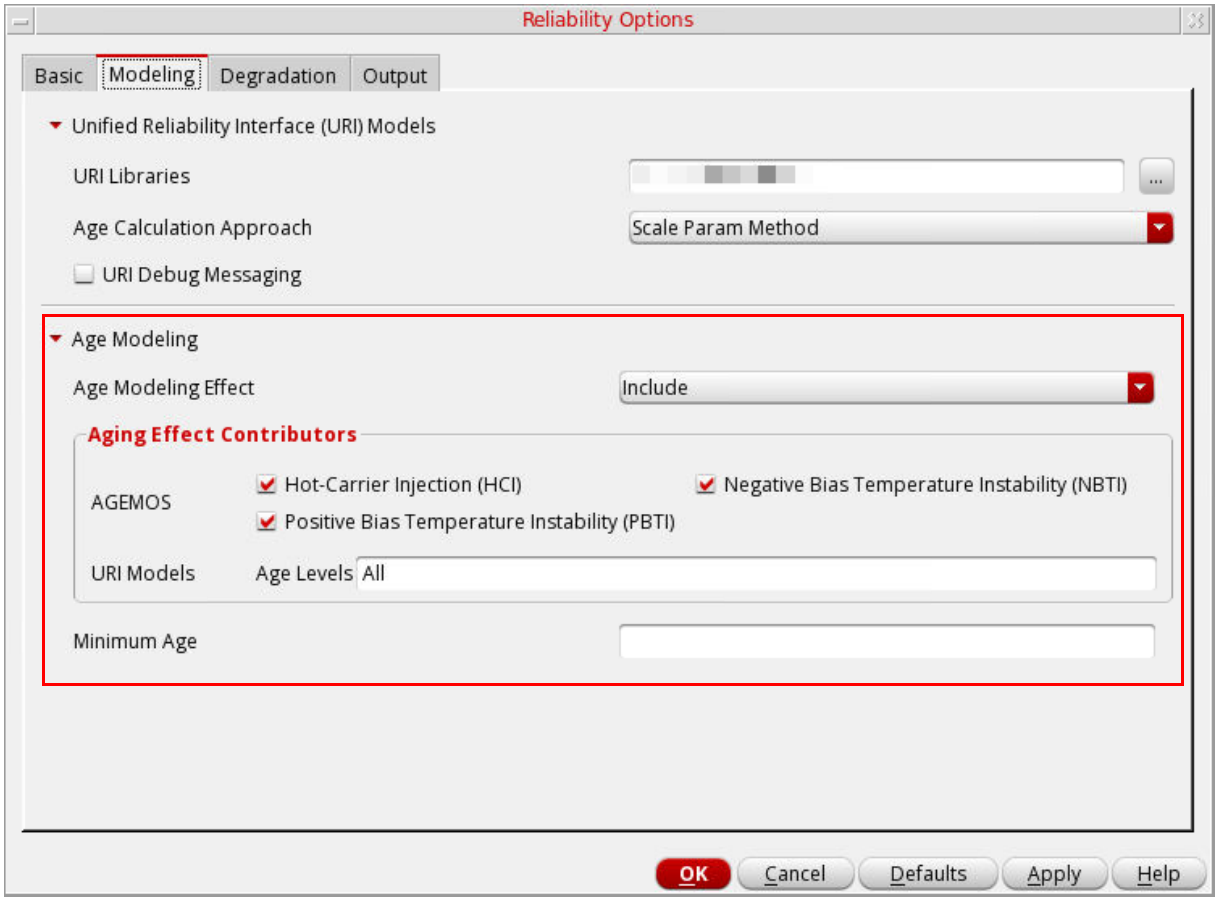
The *Age Modeling* disclosure is hidden from the *Modeling* tab on the Reliability Options form:



To hide the *Hot-Carrier Injection Modeling* disclosure and to show the *Age Modeling* disclosure in the *Modeling* tab in the Reliability Options form, add the following SKILL code in the `.cdsinit` file:

```
(defun relxCustomizeDisplayOrEnableStatus (relxOptionForm)
        relxDisplayDiscField(relxOptionForm "Hot-Carrier Injection Modeling" nil)
        relxDisplayDiscField(relxOptionForm "Age Modeling" t)
    )
```

The *Hot-Carrier Injection Modeling* disclosure is hidden and the *Age Modeling* disclosure is shown on the *Modeling* tab in the Reliability Options form:



relxEnableDiscField

```
relxEnableDiscField(
     r_formObject
     t_disclosureName
     g_enableBoolean
     )
     => t / nil
```

Description

Enables or disables the specified disclosure in the Reliability Options form.

Arguments

r_formObject	The form object: `relxOptionForm`.
t_disclosureName	Name of the disclosure to be enabled or disabled. For example, `Age Modeling`
g_enableBoolean	Specifies whether to enable or disable the specified disclosure. Possible values are: ■ `t`: Enables the disclosure. ■ `nil`: Disables the disclosure.

Value Returned

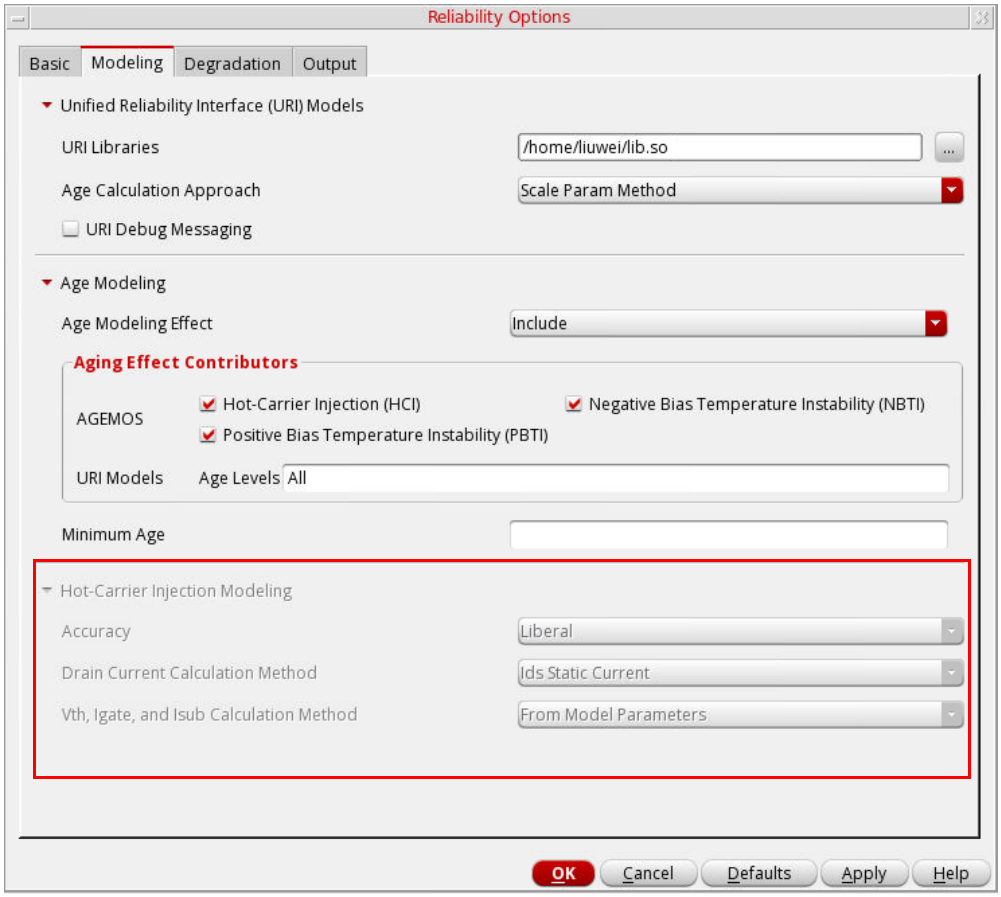
`t`	Returns `t` if the disclosure is enabled.
`nil`	Returns `nil` the disclosure is disabled.

Examples

To disable the *Hot-Carrier Injection Modeling* disclosure on the *Modeling* tab in the Reliability Options form, add the following SKILL code in the `.cdsinit` file:

```
(defun relxCustomizeDisplayOrEnableStatus (relxOptionForm)
    relxEnableDiscField(relxOptionForm "Hot-Carrier Injection Modeling" nil)
)
```

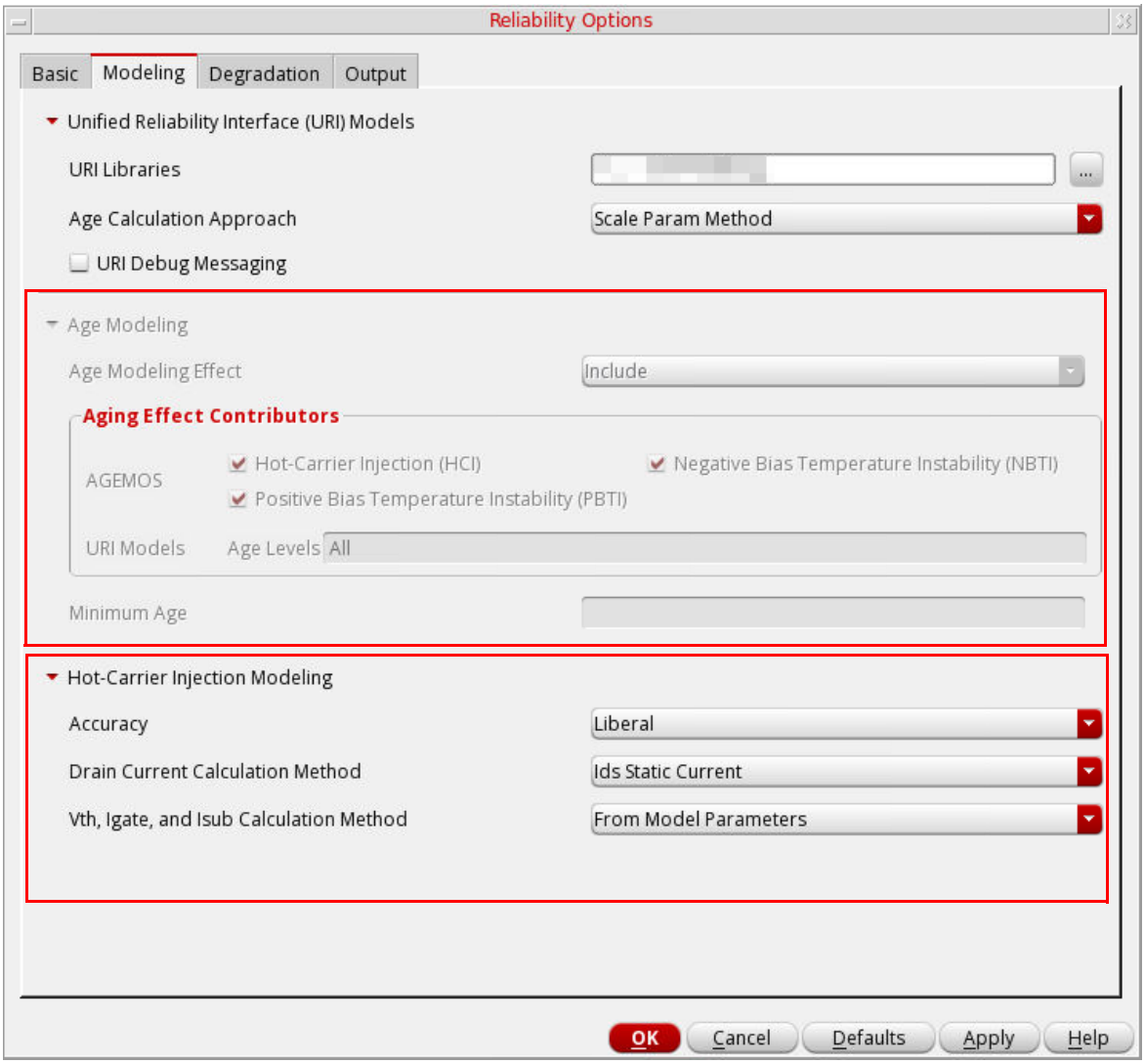
The *Hot-Carrier Injection Modeling* disclosure is disabled on the *Modeling* tab in the Reliability Options form:



To enable the *Hot-Carrier Injection Modeling* disclosure and to disable the *Age Modeling* disclosure on the *Modeling* tab in the Reliability Options form, add the following SKILL code in the `.cdsinit` file:

```
((defun relxCustomizeDisplayOrEnableStatus (relxOptionForm)
        relxEnableDiscField(relxOptionForm "Hot-Carrier Injection Modeling" t)
        relxEnableDiscField(relxOptionForm "Age Modeling" nil)
)
```

The *Hot-Carrier Injection Modeling* disclosure is enabled and the *Age Modeling* disclosure is disabled on the *Modeling* tab in the Reliability Options form:



relxHideAgeCalculationApproachField

```
relxHideAgeCalculationApproachField(
     r_formObject
     )
     => t
```

Description

Hides the *Age Calculation Approach* field from the *Modeling* tab in the Reliability Options form.

Arguments

r_formObject	The form object: `relxOptionForm`.

Value Returned

`t`	Returns `t` if the *Age Calculation Approach* field is hidden.

Examples

To hide the *Age Calculation Approach* field from the *Modeling* tab in the Reliability Options form, add the following SKILL code in the `.cdsinit` file:

```
(defun relxCustomizeDisplayOrEnableStatus (relxOptionForm)
        relxHideAgeCalculationApproachField(relxOptionForm)
    )
```

relxGetCustomTabName

```
relxGetCustomTabName(
     )
     => t_customTabName
```

Description

Returns the name of the custom tab.

Note: You can also use this function to rename the custom tab.

Arguments

None

Value Returned

t_customTabName	Returns the name of the custom tab.

Examples

The following example returns the name of the custom tab:

```
relxGetCustomTabName()
=> "Custom"
```

To rename the custom tab to `myCustomTab`, add the following SKILL code in the `.cdsinit` file:

```
(defun relxGetCustomTabName ()
        "myCustomTab"
    )
```

relxCustomizeDisplayOrEnableStatus

```
relxCustomizeDisplayOrEnableStatus(
     r_formObject
     )
     => t
```

Description

Defines the display and enable status of the disclosures, fields, and tabs.

Note: You need to define this function in the `.cdsinit` file. You need not call it.

Arguments

r_formObject	The form object: `relxOptionForm`.

Value Returned

`t`	Returns `t` by default.

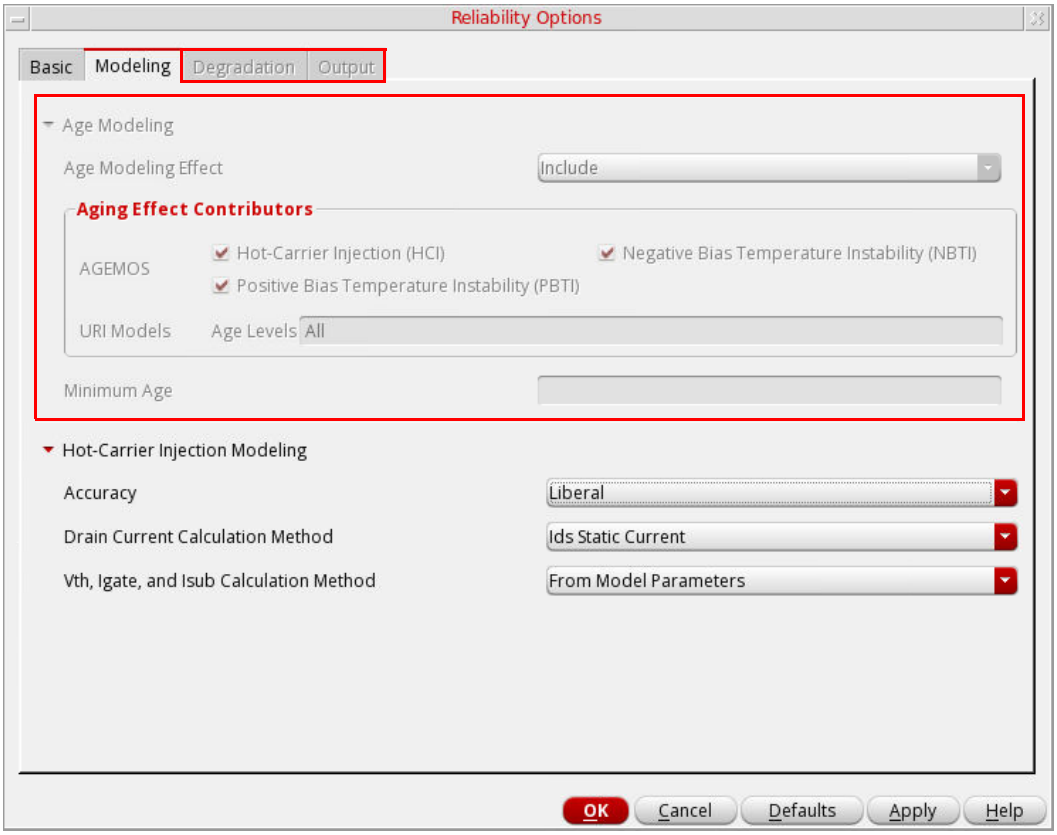
Example

Add the following SKILL code in the `.cdsinit` file to:

- Disable the *Degradation* and *Output* tabs
- Disable the *Age Modeling* section on the *Modeling* tab
- Hides the *Unified Reliability Interface (URI) Models* disclosure from the Modeling tab

```
(defun relxCustomizeDisplayOrEnableStatus (relxOptionForm)
    relxEnableFormTab(relxOptionForm "Degradation" nil)
    relxEnableFormTab(relxOptionForm "Output" nil)
    relxEnableDiscField(relxOptionForm "Age Modeling" nil)
    relxDisplayDiscField(relxOptionForm "Unified Reliability Interface (URI)
Models" nil)
```

)



relxCreateCustomizedTab

```
relxCreateCustomizedTab(
     s_formSymbol
     )
     => nil
```

Description

Adds a customized tab to the Reliability Options form.

Arguments

s_formSymbol	The form symbol.

Value Returned

`nil`	Returns `nil` by default.

Example

Add the following SKILL code in the `.cdsinit` file to:

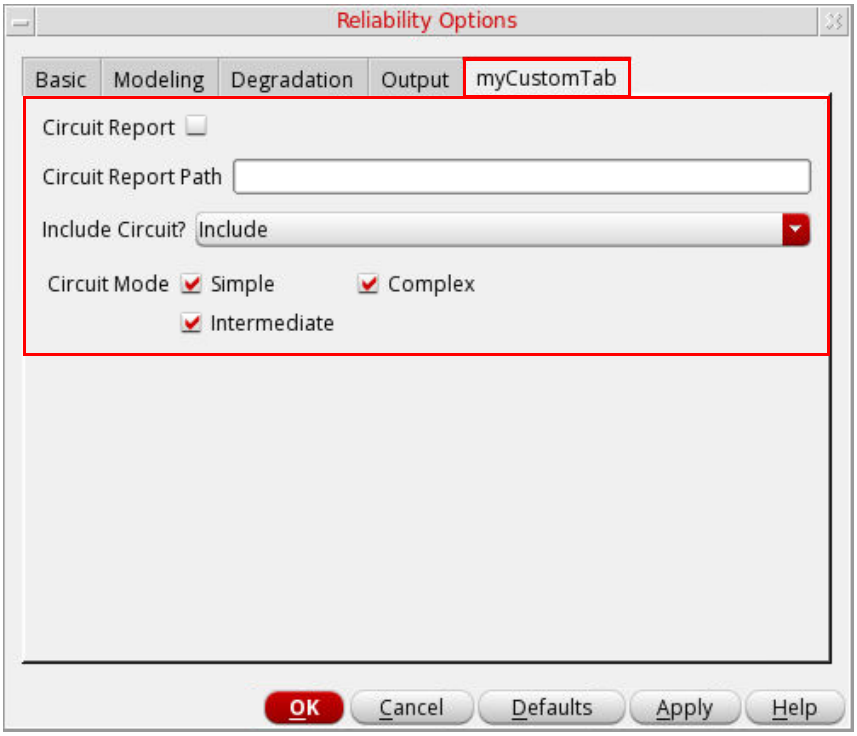
- Create a customized tab named `myCustomTab`
- Define the following fields:
  - *Circuit Report* of type `boolean`
  - *Circuit Report* Path of type `string`
  - *Circuit Type* of type `radio`
  - *Include Circuit* of type `cyclic`
  - *Circuit Mode* of type `toggle`

```
(defun relxGetCustomTabName ()
    "myCustomTab"
     )
(defun relxCreateCustomizedTab (formSymbol)
    let((cusTagVertLayout circuitReport circuitReportPath circuitType circuitMode
circuitInclude)
        circuitReport = relxAddSetupRelxOption( ?name 'circuitReport
```

```
                                        ?type 'boolean
                                        ?prompt "Circuit Report"
                                        ?defValue nil
                                            )
circuitReportPath = relxAddSetupRelxOption(?name 'circuitReportPath
                                            ?type 'string
                                            ?prompt "Circuit Report Path"
                                                )
circuitType = relxAddSetupRelxOption(?name 'circuitType
                                     ?type 'radio
                                     ?choices list("Static" "Dynamic")
                                     ?value "Static"
                                     ?prompt " Circuit Type"
                                    )
circuitMode = relxAddSetupRelxOption(?name 'circuitMode
                                     ?type 'toggle
                                     ?choices list( list( 't1 "Simple")
                                                list('t2 "Complex")
                                                list( 't3 "Intermediate")
                                                )
                                    ?value list( t t t)
                                    ?itemsPerRow 2
                                    ?prompt " Circuit Mode"
                                    )
circuitType = relxAddSetupRelxOption(?name 'circuitInclude
                                     ?type 'cyclic
                                     ?choices list("Include" "Exclude")
                                     ?value "Include"
                                     ?prompt "Include Circuit?"
                                    )
cusTagVertLayout = relxAddSetupRelxOption( ?name 'cusTagVertLayout
                           ?type 'verticalBoxLayout
                           ?items list(
                                        list(circuitReport)
                                        list(circuitReportPath)
                                        list(circuitType)
                                        list(circuitMode)
                                        list(circuitInclude)
                                        list('spacer_item 5)
                                        list('stretch_item 5)
                                        )
```

```
                                                        )
        cusTagVertLayout
                )
  )
```

The customized tab, *myCustomTab* is added to the Reliability Options form. The tab displays the fields defined bu using the `relxAddSeupRelxOption` SKILL function.



relxAddSetupRelxOption

```
relxAddSetupRelxOption (
     [ ?name s_name ]
     [ ?type s_type ]
     [ ?prompt t_prompt ]
     [ ?choices l_choices ]
     [ ?value g_value ]
     [ ?display g_display ]
     [ ?enabled g_enabled ]
     [ ?onFields l_onFields ]
     [ ?buttonLocation s_buttonLocation ]
     [ ?callback t_callback ]
     [ ?defValue g_defValue ]
     [ ?itemsPerRow x_itemsPerRow ]
     [ ?items l_items ]
     [ ?frame g_frame ]
     [ ?formApplyCB s_formApplyCB ]
     [ ?scrollable g_scrollable ]
     [ ?tooltip t_tooltip ]
     [ ?spacing x_spacing ]
     [ ?discLayout s_discLayout ]
     [ ?buttonIcon g_buttonIcon ]
     )
     => r_fieldHandle
```

Description

Adds new fields to the Reliability Options form.

Note: This function can be used as a unit function to customize the *Custom* tab.

Arguments

?name *s_name*	The symbol name of the field to be added.
?type *s_type*	Type of the field. Default value: `string`. Possible values: `radio`, `boolean`, `separator`, `label`, `cyclic`, `string`, `toggle`, and `button`.
?prompt *t_prompt*	Prompt for the field.
?choices *l_choices*	List of choices.
?value *g_value*	Value of the field.

`?display` *`g_display`*
: Controls the display of fields on the form. If this argument is set to `nil`, the field is not displayed in the form.

`?enabled` *`g_enabled`*
: Enables or disables the field. When set to `t`, the field is enabled, When set to `nil`, the field is disabled and becomes inactive in the form.

`?onFields` *`l_onFields`*
: It is applicable only for the `disclosureTriangle` type of fields.

`?buttonLocation` *`s_buttonLocation`*
: Location of the button.

  It is applicable only for the `boolean` type of fields.

`?callback` *`t_callback`*
: Specifies the SKILL functions (callbacks) that execute after you select *Apply*, *OK*, or *Cancel* in the form.

`?defValue` *`g_defValue`*
: Default value of the field.

`?itemsPerRow` *`x_itemsPerRow`*
: Number of items in a single row.

  It is applicable only for the `radio` and `toggle` type of fields.

`?items` *`l_items`*
: Defines the items in the layout.

  It is applicable only for `gridLayout`, `horizontalBoxLayout`, and `verticalBoxLayout` fields.

`?frame` *`g_frame`*
: Any of the following values:

  - `nil`: (Default) For an unframed layout.
  - `t`: For an untitled frame.
  - *`t_title`*: For a framed layout with a title.

  It is applicable only for `gridLayout`, `horizontalBoxLayout`, and `verticalBoxLayout` fields.

`?formApplyCB` *`s_formApplyCB`*
: Calls the `formApplyCB` function set when a button, such as *OK*, *Apply*, and so on, is clicked.

`?scrollable` *`g_scrollable`*

If set to `t`, adds a scroll bar in the layout if the content is large and cannot be accommodated. The default value is `nil`.

It is applicable only for `gridLayout`, `horizontalBoxLayout`, and `verticalBoxLayout` fields.

`?tooltip` *`t_tooltip`* Tooltip for the field.

`?spacing` *`x_spacing`* Specifies the integer value to apply for horizontal spacing.

`?discLayout` *`s_discLayout`*

Defines the layout of the disclosure triangle.

`?buttonIcon g_buttonIcon`

Defines the button icon.

Value Returned

`r_fieldHandle` Handle to the field.

Example

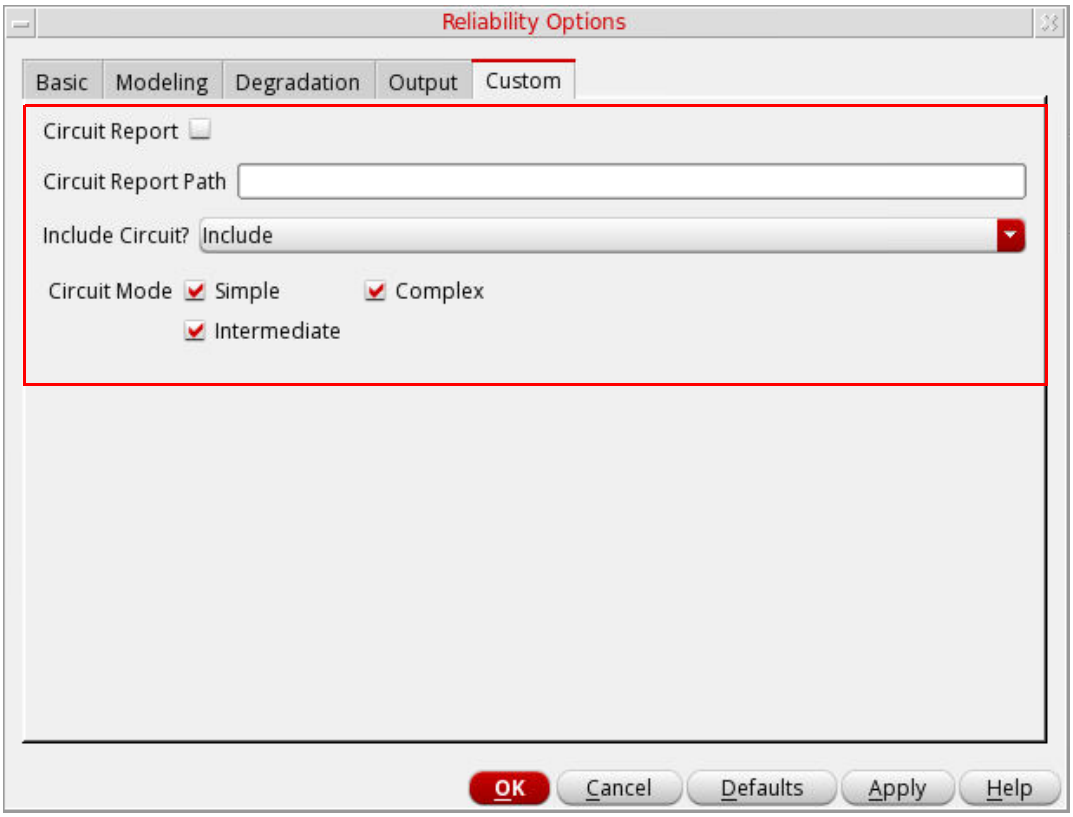
Add the following SKILL code in the `.cdsinit` file to add these fields to the *Custom* tab:

- *Circuit Report* of type `boolean`
- *Circuit Report* Path of type `string`
- *Circuit Type* of type `radio`
- *Include Circuit* of type `cyclic`
- *Circuit Mode* of type `toggle`

```
(defun relxCreateCustomizedTab (formSymbol)
    let((cusTagVertLayout circuitReport circuitReportPath circuitType circuitMode
circuitInclude)
        circuitReport = relxAddSetupRelxOption( ?name 'circuitReport
                                                ?type 'boolean
                                                ?prompt "Circuit Report"
                                                ?defValue nil
                                                    )
```

```
        circuitReportPath = relxAddSetupRelxOption(?name 'circuitReportPath
                                                   ?type 'string
                                                   ?prompt "Circuit Report Path"
                                                       )
        circuitType = relxAddSetupRelxOption(?name 'circuitType
                                             ?type 'radio
                                             ?choices list("Static" "Dynamic")
                                             ?value "Static"
                                             ?prompt " Circuit Type"
                                            )
        circuitMode = relxAddSetupRelxOption(?name 'circuitMode
                                             ?type 'toggle
                                             ?choices list( list( 't1 "Simple")
                                                        list('t2 "Complex")
                                                        list( 't3 "Intermediate")
                                                        )
                                            ?value list( t t t)
                                            ?itemsPerRow 2
                                            ?prompt " Circuit Mode"
                                            )
        circuitType = relxAddSetupRelxOption(?name 'circuitInclude
                                             ?type 'cyclic
                                             ?choices list("Include" "Exclude")
                                             ?value "Include"
                                             ?prompt "Include Circuit?"
                                            )
        cusTagVertLayout = relxAddSetupRelxOption( ?name 'cusTagVertLayout
                                  ?type 'verticalBoxLayout
                                  ?items list(
                                              list(circuitReport)
                                              list(circuitReportPath)
                                              list(circuitType)
                                              list(circuitMode)
                                              list(circuitInclude)
                                              list('spacer_item 5)
                                              list('stretch_item 5)
                                              )
                                                  )
        cusTagVertLayout
               )
  )
```

The new fields are added to the Custom tab in the Reliability Options form.



asiFormatSpecialParameterForRel

```
asiFormatSpecialParameterForRel(
     o_oasisSession
     p_fp
     )
     => t
```

Description

Netlists the new options added to the customized tab.

Note: You need to define this function in the `.cdsinit` file. You need not call it.

Arguments

o_oasisSession	The OASIS session object.
p_fp	The handle to the netlist file.

Value Returned

`t`	Returns `t` by default.

Example

Add the following SKILL code in the `.cdsinit` file to netlist the `circuitReport` and `circuitReportPath` options added to the *Custom* tab.

```
(defun  asiFormatSpecialParameterForRel (session)
        when(relxGetReliabilityOptionVal(session 'circuitReport)
        (artFprintf fp  "parameters circuit_report=1\n")
        (artFprintf fp  "parameters path_report=%s\n"
relxGetReliabilityOptionVal(session 'circuitReportPath))
    )
)
```

relxInitOptionsInCdsenv

```
relxInitOptionsInCdsenv(
     o_tool
     )
     => nil
```

Description

Creates environment variables for the customized options added to the Reliability Options form..

Note: You need to define this function in the `.cdsinit` file. You need not call it.

Arguments

o_tool	Simulation tool object.

Value Returned

`nil`	Returns `nil` by default.

Example

Add the following SKILL code in the `.cdsinit` file to add the customized variables `circuitReport` and `circuitReportPath` in the `.cdsenv` file: .

```
(defun relxInitOptionsInCdsenv (tool)
        relxAddReliabilityOption( tool
                                  ?name           'circuitReport
                                  ?type           'boolean
                                  ?value          nil
                                  ?page           "Custom"
                                  ?private        t
                                  )
        relxAddReliabilityOption( tool
                                  ?name           'circuitReportPath
                                  ?type           'string
                                  ?value          ""
                                  ?page           "Custom"
                                  ?private        t
                                  )

)
```

3

Functions for Fault Simulation

The SKILL functions described in this chapter are used for working with fault simulation in ADE Assembler:

Function	Description
maeAddFaultRule	Creates a new fault rule with the given set of properties.
maeAddFaultsToFaultGroup	Adds the given faults to one or more fault groups.
maeClearExistingFaultsForRevalidation	Clears the cache for all fault rules that exist in the active fault setup.
maeCreateOrRenameFaultGroup	Creates or renames a group with the specified name in the active fault setup.
maeDeleteFaultGroup	Deletes the specified fault group from the active fault setup.
maeDeleteFaultRule	Deletes the specified fault rule from the active fault setup.
maeEditFaultRule	Edits the given fault rule by changing the specified properties.
maeEnableFaults	Enables or disables the given fault rules and groups.
maeGetDUTForFaults	Returns a list containing the library, cell, and view name of the design under test in the active fault setup.
maeGetFaultGroups	Returns a list of fault groups that exist in the active fault setup.
maeGetFaultGroupToRun	Returns the name of the fault group selected for the fault simulation run.

Function	Description
maeGetFaultRule	Returns the properties set for the fault rule with the given name.
maeGetFaultRules	Runs Spectre `info` analysis to generate the faults for the specified fault group or fault rule. The generated faults are saved in a file. To view the generated faults, you can either open the file or use the Preview Faults commands in the Fault Setup assistant.
maeGetFaultRunModeOptions	Retrieves the run options for the fault simulation.
maeGetFaults	Runs Spectre `info` analysis to generate the faults for the specified fault group or fault rule. The generated faults are saved in a file. To view the generated faults, you can either open the file or use the Preview Faults commands in the Fault Setup assistant.
maeGetFaultSamplingOptions	Returns the sampling options set for fault simulation.
maeGetGlobalFaultOptions	Returns the global fault options set in the Virtuoso ADE Fault Setup Global Preferences form. These options are applicable to all faults unless overridden in individual faults or fault rules.
maeRunFaultSimulationWithFaultDroppingForActiveTests	Runs the fault dropping flow in batch mode for the enabled tests in the given session.
maeSetDUTForFaults	Sets the specified design under test for the fault setup.
maeSetFaultAnalysisType	Sets the analysis type for the fault simulation run mode.
maeSetFaultDFARunModeOptions	Sets the run mode options for the Direct Fault Analysis (DFA) to be run for fault simulation.
maeSetFaultGroupToRun	Sets the fault group to run for the active fault setup.
maeSetFaultSamplingOptions	Sets the sampling options for the fault setup.

Function	Description
maeSetFaultTFARunModeOptions	Sets the run mode options for Transient Fault Analysis to be run for fault simulation.
maeSetGlobalFaultOptions	Sets global preferences for fault simulations.
maeGetCurrentRunPlanName	Returns the name of the current run plan getting executed.
maeGetEnabledRuns	Returns a list of run IDs for the enabled runs in the run plan.
maeGetHistoryNameForCurrentRunInRunPlan	Returns the history name for the current run in the run plan.
maeGetNumberOfExecutedRuns	Returns the number of runs executed in the flow. It only counts a run that has been completed, partial runs are not included. This is exclusive for the fault dropping flow.
maeGetNumberOfUndetectedFaultsFromHistory	Returns the list of undetected faults for the history provided.
maeIsFinalRunCompleted	Checks if the current run is the last run executed in run plan. A run is said to be the last run if either no further runs exist in the run plan or no undetected faults are found in the current run.
maeIsFirstRunInRunPlan	Checks if the current run is the first run executed in run plan. Implemented exclusively for the fault dropping flow.
maeMergeFaultHistories	Merges fault histories into one.
maePrintFaultDroppingStatistics	Prints the formatted results obtained after execution of the fault dropping flow. It also writes the same to the log file in the current work directory. This is exclusive for the fault dropping flow.
maeSaveFaultsRunCount	Saves the faults count globally for further use. If a fault count already exists, it adds the provided value to the existing value. This is exclusive for the fault dropping flow.

Function	Description
maeSwitchActiveFaultGroupForCurrentRun	Sets the fault group to run for the current run in run plan. This is exclusive to the fault dropping flow.
maeSetFaultGroupToRun	Sets the fault group to run for the active fault setup.

maeAddFaultRule

```
maeAddFaultRule(
     [ ?session t_sessionName ]
     [ ?faultType t_faultType ]
     [ ?resistance t_resistance ]
     [ ?weightExpr t_weightExpr ]
     [ ?weightFactor t_weightFactor ]
     [ ?inst t_instance ]
     [ ?excludeInst t_excludeInst ]
     [ ?excludeSubckt t_excludeSubckt ]
     [ ?faultDevices t_faultDevices ]
     [ ?pinNames t_pinNames ]
     [ ?extraOptions t_extraOptions ]
     [ ?useNetlistSyntax g_useNetlistSyntax ]
     [ ?enableIEEE2427Mode g_enableIEEE2427Mode ]
     [ ?enableFaultCollapse g_enableFaultCollapse ]
     )
     => t / nil
```

Description

Creates a new fault rule with the given set of properties.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

`?faultType` *`t_faultType`*

Type of the fault to be inserted.

Possible values:

- `"Bridge"`
- `"Open (Terminal based)"`

Default value: `"Bridge"`

`?resistance` *`t_resistance`*

Resistance value for the fault rule. When not specified, the resistance value is taken from the global preferences for faults.

Possible values: A positive numeric value in string format.

`?weightExpr` *`t_weightExpr`*

An expression that defines the fault weighting function. When not specified, the weight expression is taken from the global preferences for faults.

Default value is taken from global fault options.

`?weightFactor` *`t_weightFactor`*

Weight factor for the fault rule. When not specified, the weight factor is taken from the global preferences for faults.

Possible values: A positive numeric value in string format.

`?inst` *`t_instance`*

Name of an instance below which you need to inject faults in design hierarchy.

Default: `"*"`

`?excludeInst` *`t_excludeInst`*

A space-separated list of instance paths to be excluded while injecting faults.

`?excludeSubckt` *`t_excludeSubckt`*

A space-separated list of subcircuits to be excluded while injecting faults.

`?faultDevices` *`t_faultDevices`*

A space-separated list of fault devices on which you want to inject faults.

Default value: `"*"`

`?pinNames` *`t_pinNames`*

A space-separated list of pin names on which you want to inject faults.

value: `"*"`

`?extraOptions` *`t_extraOptions`*

Additional info analysis options in the syntax supported by the simulator.

`?useNetlistSyntax` *`g_useNetlistSyntax`*

A boolean value that specifies whether to use the netlist format for instance names.

Default value: `nil`

`?enableIEEE2427Mode` *`g_enableIEEE2427Mode`*

A boolean value that enables the IEEE 2427 mode in Spectre info analysis.

Default value: `nil`

`?enableFaultCollapse` *`g_enableFaultCollapse`*

A boolean value that enables fault collapsing for the fault rule. This option is overridden by the fault collapsing options provided in the global preferences.

Value Returned

`t`	A fault rule is successfully created.
`nil`	In case of an error.

Example

The following example shows how to add or create a new fault rule with all default values:

```
maeAddFaultRule()
=> t
; a new fault rule is added to the maestro session
```

The following example shows how to add or create a new fault rule to insert bridges of resistance 150 ohms in the design hierarchy under instance I3:

```
maeAddFaultRule(?faultType "Bridge" ?weightExpr "w * 2" ?inst "I3" ?resistance
"150")
=> t
```

maeAddFaultsToFaultGroup

```
maeAddFaultsToFaultGroup(
     [ ?session t_sessionName ]
     [ ?faultsName l_faultsName ]
     [ ?faultGroupsName l_faultGroupsName ]
     )
     => t / nil
```

Description

Adds the given faults to one or more fault groups.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

`?faultsName` *`l_faultsName`*

A list containing the names of the fault rules to be added to the groups.

`?faultGroupsName` *`l_faultGroupsName`*

A list containing the names of the fault groups to which the given faults are to be added.

If no group is found with the given name, a new group is created.

Value Returned

`t`	Faults are successfully added to the groups.
`nil`	If the given fault rule is not found in the setup.

Example

The following example shows how to add the fault rules `RB1` and `rule2` to the fault group named `Group1`:

```
maeAddFaultsToFaultGroup(?faultsName '("RB1" "rule2") ?faultGroupsName
'("Group1"))
=> t
```

maeClearExistingFaultsForRevalidation

```
maeClearExistingFaultsForRevalidation(
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Clears the cache for all fault rules that exist in the active fault setup.

Arguments

`?session` *`t_sessionName`*	Name of a session. Default: Current session

Value Returned

`t`	The cache is cleared for all fault rules.
`nil`	In case of an error.

Example

The following example shows how to clear cache for all fault rules that exist in the active fault setup of the current session:

```
maeClearExistingFaultsForRevalidation()
=> t
```

maeCreateOrRenameFaultGroup

```
maeCreateOrRenameFaultGroup(
     t_newGroupName
     [ ?oldGroupName t_oldGroupName ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Creates or renames a group with the specified name in the active fault setup.

ArgumentsValue Returned

t_newGroupName

Name of the new group.

Note: This is a mandatory argument. Ensure that this value does not match with any of the existing group names.

?oldGroupName *t_oldGroupName*

Name of an existing group that you want to rename. Provide this argument only when you want to rename an existing group.

?session *t_sessionName*

Name of a session.

Default: Current session

t	Creates or renames a group in the active fault setup
nil	If another group with the same name already exists in the setup

Example

The following example shows how to rename an existing group named Group1 to Group2:

```
maeCreateOrRenameFaultGroup("Group2" ?oldGroupName "Group1")
=t
```

maeDeleteFaultGroup

```
maeDeleteFaultGroup(
     t_groupName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Deletes the specified fault group from the active fault setup.

Arguments

t_groupName	Name of the existing fault group to be deleted
?session *t_sessionName*	Name of a session Default: Current session

Value Returned

t	Fault group is deleted successfully.
nil	In case of an error.

Example

The following example shows how to delete an existing fault group named `Group1`:

```
maeDeleteFaultGroup("Group1")
=> t
```

maeDeleteFaultRule

```
maeDeleteFaultRule(
     t_faultRuleName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Deletes the specified fault rule from the active fault setup.

Arguments

t_faultRuleName	Name of the fault rule to be deleted.
?session *t_sessionName*	Name of a session. Default: Current session

Value Returned

t	When the fault rule is deleted successfully.
nil	In case of an error.

Example

The following example shows how to delete an existing fault rule named RB1:

```
maeDeleteFaultRule("RB1")
=> t
```

maeEditFaultRule

```
maeEditFaultRule(
     t_faultRuleName
     [ ?session t_sessionName ]
     [ ?resistance t_resistance ]
     [ ?weightExpr t_weightExpr ]
     [ ?weightFactor t_weightFactor ]
     [ ?inst t_instance ]
     [ ?excludeInst t_excludeInst ]
     [ ?excludeSubckt t_excludeSubckt ]
     [ ?faultDevices t_faultDevices ]
     [ ?pinNames t_pinNames ]
     [ ?extraOptions t_extraOptions ]
     [ ?useNetlistSyntax g_useNetlistSyntax ]
     [ ?enableIEEE2427Mode g_enableIEEE2427Mode ]
     [ ?enableFaultCollapse g_enableFaultCollapse ]
     )
     => t / nil
```

Description

Edits the given fault rule by changing the specified properties.

Arguments

t_faultRuleName Name of the fault rule to be edited.

?session t_sessionName

Name of a session.

Default: Current session

?resistance *t_resistance*

Resistance value for the fault rule. When not specified, the resistance value is taken from the global preferences for faults.

Possible values: A positive numeric value in string format.

?weightExpr *t_weightExpr*

An expression that defines the fault weighting function. When not specified, the weight expression is taken from the global preferences for faults.

Default value is taken from global fault options.

`?weightFactor` *`t_weightFactor`*

Weight factor for the fault rule. When not specified, the weight factor is taken from the global preferences for faults.

Possible values: A positive numeric value in string format.

`?inst` *`t_instance`*

Name of an instance below which you need to inject faults in design hierarchy.

Default: `"*"`

`?excludeInst` *`t_excludeInst`*

A space-separated list of instance paths to be excluded while injecting faults.

`?excludeSubckt` *`t_excludeSubckt`*

A space-separated list of subcircuits to be excluded while injecting faults.

`?faultDevices` *`t_faultDevices`*

A space-separated list of fault devices on which you want to inject faults.

Default value: `"*"`

`?pinNames` *`t_pinNames`*

A space-separated list of pin names on which you want to inject faults.

value: `"*"`

`?extraOptions` *`t_extraOptions`*

Additional info analysis options in the syntax supported by the simulator.

`?useNetlistSyntax` *`g_useNetlistSyntax`*

A boolean value that specifies whether to use the netlist format for instance names.

Default value: `nil`

`?enableIEEE2427Mode` *`g_enableIEEE2427Mode`*

A boolean value that enables the IEEE 2427 mode in Spectre info analysis.

Default value: `nil`

`?enableFaultCollapse` *`g_enableFaultCollapse`*

A boolean value that enables fault collapsing for the fault rule. This option is overridden by the fault collapsing options provided in the global preferences.

Value Returned

`t`	When the specified fault rule is edited successfully.
`nil`	In case of an error.

Example

The following example shows how to edit a fault rule:

```
maeEditFaultRule("RB5" ?weightExpr "w * l" ?inst "I3" ?resistance "250")
=> t
```

maeEnableFaults

```
maeEnableFaults(
     [ ?session t_sessionName ]
     [ ?groupNamesList l_groupNamesList ]
     [ ?ruleNamesList l_ruleNamesList ]
     [ ?enable g_enable ]
     [ ?enableAll g_enableAll ]
     [ ?disableAll g_disableAll ]
     )
     => list(libName cellName viewName) / nil
```

Description

Enables or disables the given fault rules and groups.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

`?groupNamesList` *`l_groupNamesList`*

A space-separated list of fault group names that need to be enabled or disabled. It is mandatory to provide at least one of the two arguments, `?groupNamesList` and `?ruleNamesList`.

`?ruleNamesList` *`l_ruleNamesList`*

A space-separated list of fault group names that need to be enabled or disabled. It is mandatory to provide at least one of the two arguments, `?groupNamesList` and `?ruleNamesList`.

`?enable` *`g_enable`*

The enabled or disabled status to be set for the given list of faults or groups.

Default value: `t`

`?enableAll` *`g_enableAll`*

A boolean value that specifies whether to enable all fault rules and groups in the active setup.

Default value: `nil`

You can set either `?enableAll` or `?disableAll` to `t` at the same time.

`?disableAll` *`t_disableAll`*

A boolean value that specifies whether to disable all fault rules and groups in the active setup.

Default value: `nil`

Note: You can set either `?enableAll` or `?disableAll` to `t` at the same time.

Value Returned

`t`	When the specified rules and groups are enabled or disabled, as specified.
`nil`	In case of an error.

Example

The following example shows how to enable two fault rules and two groups:

```
maeEnableFaults(?ruleNamesList '("RB1" "ROT1") ?groupNamesList '("Group1"
"Group2"))
=> t
```

The following example code disables two fault rules and a group:

```
maeEnableFaults(?ruleNamesList '("RB1" "RB2") ?groupNamesList '("grp1") ?enable
nil)
t
```

maeGetDUTForFaults

```
maeGetDUTForFaults(
     [ ?session t_sessionName ]
     )
     => l_LCVDetails/ nil
```

Description

Returns a list containing the library, cell, and view name of the design under test in the active fault setup.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

Value Returned

`l_LCVDetails`	A list containing the library, cell, and view name of the design under test for the fault simulation.
`nil`	In case of an error.

Example

The following example shows how to get the design under test from the fault setup:

```
maeGetDUTForFaults
=> ("AMP" "pll_lpf_amp_sim" "schematic")
```

maeGetFaultGroups

```
maeGetFaultGroups(
     [ ?session t_sessionName ]
     )
     => l_faultGroupNames/ nil
```

Description

Returns a list of fault groups that exist in the active fault setup.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

Value Returned

`l_faultGroupNames`	A list of fault groups that exist in the active fault setup.
`nil`	In case of an error.

Example

The following example shows how to get a list of fault groups from the active fault setup:

```
maeGetFaultGroups()
=> ("Group1" "Group2")
```

maeGetFaultGroupToRun

```
maeGetFaultGroupToRun(
     [ ?session t_sessionName ]
     )
     => t_faultGroupToRun / nil
```

Description

Returns the name of the fault group selected for the fault simulation run.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

Value Returned

`t_faultGroupToRun`	Name of the fault group.
`nil`	In case of an error.

Example

The following example shows how to retrieve the current `faultGroupToRun` parameter:

```
maeGetFaultGroupToRun()
=> "Rules/Ind. Faults"
```

maeGetFaultRule

```
maeGetFaultRule(
     [ faultRuleName t_faultRuleName]
     [ ?session t_sessionName ]
     )
     => l_faultRule / nil
```

Description

Returns the properties set for the fault rule with the given name.

Arguments

`?faultRuleName` *`t_faultRuleName`*

Name of the fault rule.

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

Value Returned

`l_`*`faultRule`	A disembodied property list.
`nil`	A fault rule with given name is not found.

Example

The following example shows how to return the properties of a fault rule named:

```
maeGetFaultRule("RB5")
=> (nil faultRuleName "RB5" faultType "Bridge" resistance "250" weightExpr "w * l"
weightFactor "1" inst "I3" excludeInst "" excludeSubckt "" faultDevices "NM0"
pinNames "*" extraOptions "" useNetlistSyntax nil enableIEEE2427Mode nil
enableFaultCollapse t
)
```

maeGetFaultRules

```
maeGetFaultRules(
     [ ?session t_sessionName ]
     )
     =>l_faultRuleNames / nil
```

Description

Returns a list of the names of fault rules in the fault setup.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

Value Returned

`l_faultRuleNames`	List of fault rule names.
`nil`	In case of an error.

Examples

```
maeGetFaultRules()
=> ("RB1" "RB2" "RB3" "RB4" "RB5" "RB6" "ROT1" )
```

maeGetFaultRunModeOptions

```
maeGetFaultRunModeOptions(
     [ ?session t_sessionName ]
     [ ?faultAnalysisType t_faultAnalysisType ]
     )
     => l_runModeOptions / nil
```

Description

Retrieves the run options for the fault simulation.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

`?faultAnalysisType` *`t_faultAnalysisType`*

Type of fault analysis for which the run options are to be retrieved. By default, returns data for the currently selected analysis type.

Possible Values:

- `"DFA"`
- `"TFA"`

Value Returned

`l_runModeOptions`	List of options set for the specified or the currently selected analysis type.
`nil`	In case of an error.

Example

The following example shows how to retrieve the run mode options:

```
maeGetFaultRunModeOptions(?faultAnalysisType "DFA")
=> (nil analysisType "DFA" runNominal t)
```

maeGetFaults

```
maeGetFaults(
     [ ?session t_sessionName ]
     [ ?activeGroupToRun g_activeGroupToRun ]
     [ ?faultRuleName t_faultRuleName ]
     [ ?groupName t_groupname ]
     )
     => t_faultsFileName / nil
```

Description

Runs Spectre `info` analysis to generate the faults for the specified fault group or fault rule. The generated faults are saved in a file. To view the generated faults, you can either open the file or use the *Preview Faults* commands in the Fault Setup assistant.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

`?activeGroupToRun` *`g_activeGroupToRun`*

A boolean value that specifies whether to get faults for the currently selected group.

Possible values: `t` and `nil`

Default value: `t`

Note: If *`g_activeGroupToRun`* is set to `nil`, either *`t_faultRuleName`* or *`t_groupName`* is required.

`?faultRuleName` *`t_faultRuleName`*

Name of an existing fault rule for which faults need to be generated.

`?groupName` *`t_groupName`*

Name of an existing group for which faults need to be generated.

Value Returned

t_faultsFileName	The path to the fault file containing the generated faults.
nil	If the specified fault group is not found.

Example

The following example shows how to get faults for the group RB1:

```
maeGetFaults(?faultRuleName "RB1")

=> INFO (ADE-3071): Simulation completed successfully.
reading simulation data...
...successful.

To view the faults generated, open
/usePath/simulation/libName/cellName/viewName/results/maestro/.tmpADEDir_usename/
DCGain/libName_cellName_viewName_spectre/faultInfo/psf/ruleFaults.scs file.
```

maeGetFaultSamplingOptions

```
maeGetFaultSamplingOptions(
     [ ?session t_sessionName ]
     )
     => l_samplingOptions / nil
```

Description

Returns the sampling options set for fault simulation.

Arguments

`?session` *`t_sessionName`*	Name of a session Default: Current session

Value Returned

`l_samplingOptions`	A disembodied property list of fault sampling options and their values.
`nil`	When the given session is not found.

Examples

The following example shows how this function returns fault sampling options:

```
maeGetFaultSamplingOptions()
(nil enableSampling t samplingMethod "randomweighted" sampleRatio "50" seedValue
"1" confidenceValue 97.0 )
```

maeGetGlobalFaultOptions

```
maeGetGlobalFaultOptions(
     [ ?session t_sessionName ]
     )=> l_globalOptions / nil
```

Description

Returns the global fault options set in the Virtuoso ADE Fault Setup Global Preferences form. These options are applicable to all faults unless overridden in individual faults or fault rules.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

Value Returned

`l_globalOptions`	A disembodied property list of global fault options.
`nil`	When the given session is not found.

Example

The following example shows how to return global fault options set in the currently active session:

```
maeGetGlobalFaultOptions()
=> (nil defaultBridgeResistance "100" defaultOpenResistance "10K"
defaultStuckAtResistance "100" defaultWeightExpr "1" defaultWeightFactor "1"
defaultIndividualFaultWeight "1" stuckAtGround "/vin" stuckAtSupply "/vss"
applyFaultCollapsing "All Fault rules")
```

maeRunFaultSimulationWithFaultDroppingForActiveTests

```
maeRunFaultSimulationWithFaultDroppingForActiveTests(
     [ ?session t_sessionName ]
     )
     => t_historyName / nil
```

Description

Runs the fault dropping flow in batch mode for the enabled tests in the given session.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

Value Returned

`t`	Name of the history in which the results of fault simulation are saved.
`nil`	When the simulation is not run successfully.

Examples

The following example code runs the fault simulation with fault dropping enabled:

```
maeRunFaultSimulationWithFaultDroppingForActiveTests()
=> "FaultSimulation.1"
```

maeSetDUTForFaults

```
maeSetDUTForFaults(
     t_libName
     t_cellName
     t_viewName
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Sets the specified design under test for the fault setup.

Arguments

t_libName	Name of a library.
t_cellName	Name of a cell.
t_viewName	Name of a view.
?session *t_sessionName*	Name of a session. Default: Current session

Value Returned

t	The design is successfully set for the fault setup.
nil	The specified design is not found.

Examples

The following example shows how to set the design for fault simulations:

```
maeSetDUTForFaults("Two_Stage_Opamp" "DualAmp" "schematic")
=> t
```

maeSetFaultAnalysisType

```
maeSetFaultAnalysisType(
     [ ?analysisType t_analysisType ]
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Sets the analysis type for the fault simulation run mode.

Arguments

`?analysisType` *`t_analysisType`*

Name of the analysis type.

Possible values:

- `"DFA"`
- `"TFA"`

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

Value Returned

`t`	The analysis type for fault simulation is updated successfully.
`nil`	The analysis type or the maestro session is invalid.

Examples

The following example explains how to set the analysis type for the fault simulation run mode:

```
maeSetFaultAnalysisType("DFA")
=> t
```

maeSetFaultDFARunModeOptions

```
maeSetFaultDFARunModeOptions(
     [ ?session t_sessionName ]
     [ ?extraOptions t_extraOptions ]
     [ ?runNominal g_runNominal ]
     )
     => t / nil
```

Description

Sets the run mode options for the Direct Fault Analysis (DFA) to be run for fault simulation.

Arguments

`?session` *`t_sessionName`*

Name of a session

Default: Current session

`?extraOptions` *`t_extraOptions`*

Specifies the options other than `?runNominal`. These options are directly passed to Spectre. You must ensure the correctness of the names and values of these options.

Default Value: `""`

`?runNominal` *`g_runNominal`*

A boolean value that specifies whether to run simulation for nominal or not. Possible values are `t` and `nil`

Default Value: `nil`

Value Returned

`t`	Run mode options are successfully set for Direct Fault Analysis.
`nil`	When invalid values are provided for the options.

Examples

The following example explains how to set run mode options for DFA in the fault setup:

```
maeSetFaultDFARunModeOptions(?extraOptions "faultduplicate=yes" ?runNominal t)
=> t
```

maeSetFaultSamplingOptions

```
maeSetFaultSamplingOptions(
     [ ?session t_sessionName ]
     [ ?enableSampling g_enableSampling ]
     [ ?samplingMethod t_samplingMethod ]
     [ ?sampleByNumOrRatio t_sampleByNumOrRatio ]
     [ ?sampleNum t_sampleNum ]
     [ ?sampleRatio t_sampleRatio ]
     [ ?seedValue t_seedValue ]
     [ ?confidenceValue t_confidenceValue ]
     )
     => t / nil
```

Description

Sets the sampling options for the fault setup.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

`?enableSampling` *`g_enableSampling`*

A boolean value that specifies whether to enable sampling or not. Possible values are `t` and `nil`.

Default value: `t`

`?samplingMethod` *`t_samplingMethod`*

Sampling method.

Possible values:

- `"random"`
- `"randomuniform"`
- `"weightsorted"`
- `"randomweighted"`

Default value: `"random"`

For details on these sampling methods, see Fault Selection and Sampling in *Spectre Classic Simulator, Spectre APS, Spectre X, and Spectre XPS User Guide*.

`?sampleByNumOrRatio` *`t_sampleByNumOrRatio`*

Specifies how you want to choose samples.

Possible values:

- `"Number"`: Uses a specific number of faults in each sample.
- `"Ratio"`: Uses a percentage or ratio of the possible samples.

Default value: `"Number"`

`?sampleNum` *`t_sampleNum`*

Number of samples to be created.

Default value: `"1"`

Note: Specify this argument only when `?sampleByNumOrRatio` is set to `"Number"` and `?sampleRatio` is not specified.

`?sampleRatio` *`t_sampleRatio`*

The ratio of the possible samples to be created. Specify a value between 0 and 100.

Default value: `"1"`

Note: Specify this argument only when `?sampleByNumOrRatio` is set to `"Ratio"` and `?sampleNum` is not specified.

`?seedValue` *`t_seedValue`*

A seed value greater than 0.

Default value: `"1"`

`?confidenceValue` *`t_confidenceValue`*

Value for confidence interval between 0 and 100.

Default value: `"97.8"`

Value Returned

`t`	Sampling options are updated successfully.
`nil`	In case of an error.

Examples

The following example explains how to set sampling options for the current fault setup:

```
maeSetFaultSamplingOptions( ?enableSampling t ?samplingMethod "random"
?sampleByNumOrRatio "Ratio" ?sampleRatio "2" ?seedValue "12")
=> t
```

maeSetFaultTFARunModeOptions

```
maeSetFaultTFARunModeOptions (
     [ ?session t_sessionName ]
     [ ?extraOptions t_extraOptions ]
     [ ?simulationMethod t_simulationMethod ]
     [ ?maxIterations t_maxIterations ]
     [ ?leadTime t_leadTime ]
     [ ?faultPointsMethod t_faultPointsMethod ]
     [ ?faultTimePoints g_faultTimePoints ]
     [ ?start_step_stop l_start_step_stop ]
     )
     => t / nil
```

Description

Sets the run mode options for Transient Fault Analysis to be run for fault simulation.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

`?extraOptions` *`t_extraOptions`*

Extra options for Transient Fault Analysis to be passed to Spectre directly.

Default value: `""`

`?simulationMethod` *`t_simulationMethod`*

The simulation method for Transient Fault Analysis.

Possible values:

- `"linear"` (default)
- `"onestep"`
- `"testpoint"`
- `"timezero"`
- `"maxiters"`
- `"leadtime"`

`?maxIterations` *`t_maxIterations`*

The maximum number of iterations greater than 0. Set this argument only when the simulation method is `"maxiters"`.

Default value: `"50"`

`?leadTime` *`t_leadTime`*

The lead time value greater than 0. Set this argument only when the simulation method is `"leadtime"`.

Default value: `"50u"`

`?faultTimePointsMethod` *`t_faultTimePointsMethod`*

The method to set fault points.

Possible values:

- `"Start/Step/Stop"`
- `"Fault Time Points"`

Default value: ""

`?faultTimePoints` *`g_faultTimePoints`*

A string or list of strings to be set when `?faultTimePointsMethod` is set to `"Fault Time Points"`.

Default value: ""

`?start_step_stop` *`l_start_step_stop`*

A list of three numeric values greater than 0 in string format to be used as start, step, and stop values when `?faultTimePointsMethod` is set to `"Start/Step/Stop"` .

Note: The total of start and step time should be less than the stop time.

Default value: A blank list.

Value Returned

`t`	Run mode options for Transient Fault Analysis are set successfully.
`nil`	Returns `nil` if there is an error.

Examples

The following example code sets up the run mode options for Transient Fault Analysis in the fault setup:

```
maeSetFaultTFARunModeOptions(?simulationMethod "maxiters" ?maxIterations "50"
?faultPointsMethod "Start/Step/Stop" ?start_step_stop '("3" "4" "11"))
=> t
```

maeSetGlobalFaultOptions

```
maeSetGlobalFaultOptions (
     [ ?session t_sessionName ]
     [ ?defaultBridgeResistance t_defaultBridgeResistance ]
     [ ?defaultOpenResistance t_defaultOpenResistance]
     [ ?defaultStuckAtResistance t_defaultStuckAtResistance ]
     [ ?stuckAtGround t_stuckAtGround ]
     [ ?stuckAtSupply t_stuckAtSupply ]
     [ ?defaultWeightExpr t_defaultWeightExpr ]
     [ ?defaultWeightFactor t_defaultWeightFactor ]
     [ ?defaultIndividualFaultWeight t_defaultIndividualFaultWeight ]
     [ ?applyFaultCollapsing t_applyFaultCollapsing ]
     )
     => t / nil
```

Description

Sets global preferences for fault simulations.

Arguments

`?session` *`t_sessionName`*

Name of a session.

Default: Current session

`?defaultBridgeResistance` *`t_defaultBridgeResistance`*

Default resistance value greater than 0 for bridge faults.

Default value: `"100"`

`?defaultOpenResistance` *`t_defaultOpenResistance`*

Default resistance value greater than 0 for open faults.

Default value: `"10K"`

`?defaultStuckAtResistance` *`t_defaultStuckAtResistance`*

Default resistance value greater than 0 for stuck-at faults.

Default value: `"100"`

`?stuckAtGround` *`t_stuckAtGround`*

Stuck at ground node for stuck-at faults.

Default value: ""

`?stuckAtSupply` *`t_stuckAtSupply`*

Stuck at supply node for stuck-at faults.

Default value: ""

`?defaultWeightExpr` *`t_defaultWeightExpr`*

Default weight expression for all fault rules.

Default value: `"1"`

`?defaultWeightFactor` *`t_defaultWeightFactor`*

Default weight factor greater than 0 for all fault rules.

Default value: `"1"`

`?defaultIndividualFaultWeight` *`t_default`*`IndividualFaultWeight`

Default weight greater than 0 for individual faults.

Default value: `"1"`

`?applyFaultCollapsing` *`t_applyFaultCollapsing`*

Fault collapsing policy for fault rules.

Possible values:

- `"All Fault rules"`
- `"Rule Specific"`
- `"None"`

Default value: `"All Fault rules"`

Value Returned

`t`	Global preferences for fault simulation are set successfully.
`nil`	An invalid value is specified for a global option or the given session is not found.

Example

The following example shows how to set the global fault options:

```
maeSetGlobalFaultOptions(?defaultBridgeResistance "100" ?defaultOpenResistance
"10K" ?defaultStuckAtResistance "100" ?stuckAtGround "/vin" ?stuckAtSupply "/vss"
```

```
?defaultWeightExpr "1" ?defaultWeightFactor "1" ?defaultIndividualFaultWeight "1"
?applyFaultCollapsing "All Fault rules")

=> t
```

maeGetCurrentRunPlanName

```
maeGetCurrentRunPlanName(
     )
     => t_runPlanName
```

Description

Returns the name of the current run plan getting executed.

Arguments

None.

Value Returned

t_runPlanName	Name of the current run plan is returned.

Example

The following example shows how to return the name of the current run plan:

```
Example: maeGetCurrentRunPlanName()
=> "Plan.0"
```

maeGetEnabledRuns

```
maeGetEnabledRuns(
     [ ?session t_sessionName ]
     )
     => l_enabledRuns / nil
```

Description

Returns a list of run IDs for the enabled runs in the run plan.

Arguments

`?session` *`t_sessionName`*

Name of the session.

Default: Current session.

Value Returned

`l_enabledRuns`	List of run IDs for enabled runs.
`nil`	No runs are enabled.

Example

The following example shows how to return a list of run IDs:

```
maeGetEnabledRuns()
=> (14930 14948)
```

maeGetHistoryNameForCurrentRunInRunPlan

```
maeGetHistoryNameForCurrentRunInRunPlan(
     )
     => t_historyName / nil
```

Description

Returns the history name for the current run in the run plan.

Arguments

None.

Value Returned

t_historyName	History name for the run being executed.
nil	No history available for the current run.

Example

The following example shows how to return the history name:

```
maeGetHistoryNameForCurrentRunInRunPlan()
=> "Plan.0.Run.1"
```

maeGetNumberOfExecutedRuns

```
maeGetNumberOfExecutedRuns(
     [ ?session t_sessionName ]
     [ ?runPlanName t_runPlanName ]
     )
     =>x_runsCompleted
```

Description

Returns the number of runs executed in the flow. It only counts a run that has been completed, partial runs are not included. This is exclusive for the fault dropping flow.

Arguments

`?session` *`t_sessionName`*	Name of the maestro session. Default: Current session.
`?runPlanName` *`t_runPlanName`*	Name of the run plan whose status needs to be extracted.

Value Returned

`x_runsCompleted`	Complete runs executed in the flow are returned.

Examples

The following example shows how to return the number of runs:

```
maeGetNumberOfExecutedRuns(?runPlanName "Plan.0")
=> 3
)
```

maeGetNumberOfUndetectedFaultsFromHistory

```
maeGetNumberOfUndetectedFaultsFromHistory(
     [ ?session t_sessionName ]
     [ ?historyName t_historyName ]
     )
     => l_UDFaults / nil
```

Description

Returns the list of undetected faults for the history provided.

Arguments

`?session` *`t_sessionName`*	Name of the maestro session. Default: Current session.
`?historyName` *`t_historyName`*	History name for which list of undetected faults is to be retrieved.

Value Returned

`l_UDFaults`	List of undetected faults for the provided history name.
`nil`	No undetected faults present.

Example

The following example shows how to return the list of undetected faults:

```
maeGetNumberOfUndetectedFaultsFromHistory(?historyName "FaultSimulation.1")
=> ("ROT1_open_1" "ROT1_open_2" "ROT1_open_3" "ROT1_open_4")
```

maeIsFinalRunCompleted

```
maeIsFinalRunCompleted(
     [ ?session t_sessionName ]
     )
     => t / nil
```

Description

Checks if the current run is the last run executed in run plan. A run is said to be the last run if either no further runs exist in the run plan or no undetected faults are found in the current run.

Implemented exclusively for the fault dropping flow.

Arguments

`?session` *`t_sessionName`*	Name of the maestro session. Default: Current session.

Value Returned

`t`	Current executed run is the final run.
`nil`	Current executed run is not the final run.

Example

The following example shows how to check if the current run is the last run executed:

```
maeIsFinalRunCompleted()
=> t
```

maeIsFirstRunInRunPlan

```
maeIsFirstRunInRunPlan(
     [ ?session t_sessionName ]
     [ ?historyName t_historyName ]
     )
     => t / nil
```

Description

Checks if the current run is the first run executed in run plan. Implemented exclusively for the fault dropping flow.

Arguments

`?session` *`t_sessionName`*	Name of the session. Default: Current session.
`?historyName` *`t_historyName`*	History name for the current run executed.

Value Returned

`t`	Current executed run is the first run.
`nil`	Current executed run is not the first run.

Examples

The following example shows how to check if the current run is the first run:

```
maeIsFirstRunInRunPlan(?historyName "Plan.0.Run.0")
=> t
```

maeMergeFaultHistories

```
maeMergeFaultHistories(
     [ ?session t_sessionName ]
     [ ?historiesToMerge l_historiesToMerge ]
     [ ?finalHistoryName t_finalHistoryName ]
     [ ?logFile t_logFile ]
     )
     => t/ nil
```

Description

Merges fault histories into one.

Arguments

`?session` *`t_sessionName`*	Name of the session Default: Current session
`?historiesToMerge` *`l_historiesToMerge`*	List of histories that are to be merged.
`?finalHistoryName` *`t_finalHistoryName`*	History name in which all the histories are to be merged.
`?logFile` *`t_logFile`*	A file to print the logs of merging. The default file name is `_faultDropping.logs`. **Note:** Prints are disabled for now.

Value Returned

`t`	Fault histories are merged.
`nil`	Merging failed with a relevant error message.

Example

The following example shows how to merge fault histories:

```
maeMergeFaultHistories(?historiesToMerge '("Plan.0.Run.0" "Plan.0.Run.1"
"Plan.0.Run.2" "Plan.0.Run.3") ?finalHistoryName "Plan.0.Run.3" ?logFile
"_faultDropping.logs")
```

```
=> t
```

maePrintFaultDroppingStatistics

```
maePrintFaultDroppingStatistics(
     [ ?session t_sessionName ]
     [ ?historiesName l_historiesName ]
     [ ?logFile t_logFile ]
     )
     => t
```

Description

Prints the formatted results obtained after execution of the fault dropping flow. It also writes the same to the log file in the current work directory. This is exclusive for the fault dropping flow.

Arguments

`?session` *`t_sessionName`*

Name of the session

Default: Current session

`?historiesName` *`l_historiesName`*

Name of the history containing final results post fault dropping flow is completed.

`?logFile` *`t_logFile`*

Name of the log file to be written in the current work directory. The default file name is `_faultDropping.logs`.

Value Returned

`t`	Formatted results are printed.

Examples

The following example explains how to print the formatted results:

```
maePrintFaultDroppingStatistics(?session _axlGetCurrentSession() ?historyName
"Plan.2.Run.7" ?logFile "_faultDropping.logs")
=>
```

```
Fault Dropping : Sep 11 12:37:15 2020 : --------------Fault Dropping Run
Summary----------------

Fault Dropping : Sep 11 12:37:15 2020 : Number of expected simulations to run was
12.

Fault Dropping : Sep 11 12:37:15 2020 : Number of simulations saved is 6.

Fault Dropping : Sep 11 12:37:15 2020 : Number of corners run in the setup is 0.

Fault Dropping : Sep 11 12:37:15 2020 : Cumulative coverage is 100.0.

Fault Dropping : Sep 11 12:37:15 2020 : Cumulative weighted coverage is 100.0.

Fault Dropping : Sep 11 12:37:15 2020 : Fault simulation has been run for all active
tests. Results available in Plan.2.Run.7.

Fault Dropping : Sep 11 12:37:15 2020 : -----------Fault Dropping Run Flow
Completed------------

t
```

maeSaveFaultsRunCount

```
maeSaveFaultsRunCount(
     [ ?session t_sessionName ]
     [ ?faultsCount x_faultsCount ]
     )
     => x_faultsCount
```

Description

Saves the faults count globally for further use. If a fault count already exists, it adds the provided value to the existing value. This is exclusive for the fault dropping flow.

Arguments

`?session` *`t_sessionName`*

Name of the session

Default: Current session

`?faultsCount` *`x_faultsCount`*

Number of faults to be saved.

Value Returned

`x_faultsCount` Final faults count saved.

Examples

The following example explains how to save the fault counts:

```
maeSaveFaultsRunCount(?faultsCount 100)
=> 108
```

maeSwitchActiveFaultGroupForCurrentRun

```
maeSwitchActiveFaultGroupForCurrentRun(
     t_historyName
     )
     => t / nil
```

Description

Sets the fault group to run for the current run in run plan. This is exclusive to the fault dropping flow.

Arguments

t_historyName	Fault group to run for the current run.

Value Returned

`t`	Fault group to run is set.
`nil`	Fault group to run couldn't be set with a relevant error message.

Examples

The following example explains how to set the fault group to run for the current run:

```
maeSwitchActiveFaultGroupForCurrentRun("Plan.0.Run.2")
=> t
```

maeSetFaultGroupToRun

```
maeSetFaultGroupToRun(
     [ ?session t_sessionName ]
     [ ?faultGroupToRun t_faultGroupToRun ]
     )
     => t / nil
```

Description

Sets the fault group to run for the active fault setup.

Arguments

`?session` *`t_sessionName`*

Name of a session

Default: Current session

`?faultGroupToRun t_faultGroupToRun`

Specify the group to set as the active group to run fault simulation.

Possible values: `"Rules/Ind. Faults"` or any other valid group name.

Default Value: `"Rules/Ind. Faults"`

Value Returned

`t`	The `faultGroupToRun` parameter is updated.
`nil`	Returns `nil` if invalid *`t_faultGroupToRun`* is specified.

Examples

The following example explains how to set the `faultGroupToRun` parameter for the active fault setup:

```
Example: maeSetFaultGroupToRun(?faultGroupToRun "Group1")
=> t
```