Altair HyperMesh 2021

Reference Guide: Solver Templates

Updated: 01/13/2021

Intellectual Property Rights Notice

Copyrights, Trademarks, Trade Secrets, Patents & Third Party Software Licenses

Copyrights in the below are held by Altair Engineering, Inc., except where otherwise explicitly stated. This Intellectual Property Rights Notice is exemplary, not exhaustive.

> **Note:** Pre-release versions of Altair software are provided 'as is', without warranty of any kind. Usage of pre-release versions is strictly limited to non-production purposes.

Altair HyperWorks™ - The Platform for Innovation™

Altair AcuConsole™ ©2006-2020

Altair AcuSolve™ ©1997-2020

Altair Activate® ©1989-2020 (formerly solidThinking Activate)

Altair Compose® ©2007-2020 (formerly solidThinking Compose)

Altair ConnectMe™ ©2014-2020

Altair EDEM ©2005-2020 DEM Solutions Ltd, ©2019-2020 Altair Engineering Inc.

Altair ElectroFlo™ ©1992-2020

Altair Embed® ©1989-2020 (formerly solidThinking Embed)

- **Altair Embed SE** ©1989-2020 (formerly solidThinking Embed SE)
- **Altair Embed/Digital Power Designer** ©2012-2020
- **Altair Embed Viewer** ©1996-2020

Altair ESAComp™ ©1992-2020

Altair Feko™ ©1999-2014 Altair Development S.A. (Pty) Ltd., ©2014-2020 Altair Engineering Inc.

Altair Flux™ ©1983-2020

Altair FluxMotor™ ©2017-2020

Altair HyperCrash™ ©2001-2020

Altair HyperGraph™ ©1995-2020

Altair HyperLife™ ©1990-2020

Altair HyperMesh™ ©1990-2020

Altair HyperStudy™ ©1999-2020

Altair HyperView™ ©1999-2020

Altair HyperXtrude™ ©1999-2020

Altair Inspire™ ©2009-2020 including Altair Inspire Motion, Altair Inspire Structures, and Altair Inspire Print3D

Altair Inspire Cast ©2011-2020 (formerly Click2Cast)

Altair Packaged Solution Offerings (PSOs)

Altair Model Mesher Director™ ©2010-2020

Altair NVH Director™ ©2010-2020

Altair Squeak and Rattle Director™ ©2012-2020

Altair Virtual Gauge Director™ ©2012-2020

Altair Weight Analytics™ ©2013-2020

Altair Weld Certification Director™ ©2014-2020

Altair Multi-Disciplinary Optimization Director™ ©2012-2020

Altair PBSWorks™ - Accelerating Innovation in the Cloud™

Altair PBS Professional® ©1994-2020

Altair Control™ ©2008-2020; (formerly **PBS Control**)

Altair Access™ ©2008-2020; (formerly **PBS Access**)

Altair Accelerator™ ©1995-2020; (formerly **NetworkComputer**)

Altair Accelerator™ Plus©1995-2020; (formerly **WorkloadXelerator**)

Altair FlowTracer™ ©1995-2020; (formerly **FlowTracer**)

Altair Allocator™ ©1995-2020; (formerly **LicenseAllocator**)

Altair Monitor™ ©1995-2020; (formerly **LicenseMonitor**)

Altair Hero™ ©1995-2020; (formerly **HERO**)

Altair Software Asset Optimization (SAO) ©2007-2020

Note:

Compute Manager™ ©2012-2017 is now part of **Altair Access**

Display Manager™ ©2013-2017 is now part of **Altair Access**

PBS Application Services™ ©2008-2017 is now part of **Altair Access**

PBS Analytics™ ©2008-2017 is now part of **Altair Control**

PBS Desktop™ ©2008-2012 is now part of **Altair Access**, specifically **Altair Access desktop**, which also has **Altair Access web** and **Altair Access mobile**

e-Compute™ ©2000-2010 was replaced by **"Compute Manager"** which is now **Altair Access**

Altair KnowledgeWorks™

Altair Knowledge Studio® ©1994-2020 Angoss Software Corporation, ©2020 Altair Engineering Inc.

Altair Knowledge Studio for Apache Spark ©1994-2020 Angoss Software Corporation, ©2020 Altair Engineering Inc.

Altair Knowledge Seeker™ ©1994-2020 Angoss Software Corporation, ©2020 Altair Engineering Inc.

Altair Knowledge Hub™ ©2017-2020 Datawatch Corporation, ©2020 Altair Engineering Inc.

Third party software licenses

AcuConsole contains material licensed from Intelligent Light (www.ilight.com) and used by permission.

Software Security Measures:

Altair Engineering Inc. and its subsidiaries and affiliates reserve the right to embed software security mechanisms in the Software for the purpose of detecting the installation and/or use of illegal copies of the Software. The Software may collect and transmit non-proprietary data about those illegal copies. Data collected will not include any customer data created by or used in connection with the Software and will not be provided to any third party, except as may be required by law or legal process or to enforce our rights with respect to the use of any illegal copies of the Software. By using the Software, each user consents to such detection and collection of data, as well as its transmission and use if an illegal copy of the Software is detected. No steps may be taken to avoid or detect the purpose of any such security mechanisms.

Technical Support

Altair provides comprehensive software support via web FAQs, tutorials, training classes, telephone, and e-mail.

Altair One Customer Portal

Altair One (https://altairone.com/) is Altair's customer portal giving you access to product downloads, a Knowledge Base, and customer support. We strongly recommend that all users create an Altair One account and use it as their primary means of requesting technical support.

Once your customer portal account is set up, you can directly get to your support page via this link: www.altair.com/customer-support/

Altair Training Classes

Altair's in-person, online, and self-paced trainings provide hands-on introduction to our products, focusing on overall functionality. Trainings are conducted at our corporate and regional offices or at your facility.

For more information please visit: https://learn.altair.com/

If you are interested in training at your facility, please contact your account manager for more details. If you do not know who your account manager is, please contact your local support office and they will connect you with your account manager.

Telephone and E-mail

If you are unable to contact Altair support via the customer portal, you may reach out to technical support via phone or e-mail. Use the following table as a reference to locate the support office for your region.

When contacting Altair support, please specify the product and version number you are using along with a detailed description of the problem. It is beneficial for the support engineer to know what type of workstation, operating system, RAM, and graphics board you have, so please include that in your communication.

Location	Telephone	E-mail
Australia	+61 649 413 7981	anzsupport@altair.com
Brazil	+55 113 884 0414	br_support@altair.com
Canada	+1 416 447 6463	support@altairengineering.ca
China	+86 400 619 6186	support@altair.com.cn
France	+33 141 33 0992	francesupport@altair.com
Germany	+49 703 162 0822	hwsupport@altair.de
Greece	+30 231 047 3311	eesupport@altair.com

Location	Telephone	E-mail
India	+91 806 629 4500 +1 800 425 0234 (toll free)	support@india.altair.com
Israel		israelsupport@altair.com
Italy	+39 800 905 595	support@altairengineering.it
Japan	+81 36 225 5830	support@altairjp.co.jp
Malaysia	+60 32 742 7890	aseansupport@altair.com
Mexico	+52 555 658 6808	mx-support@altair.com
New Zealand	+64 9 413 7981	anzsupport@altair.com
South Africa	+27 21 831 1500	support@altair.co.za
South Korea	+82 704 050 9200	support@altair.co.kr
Spain	+34 910 810 080	support-spain@altair.com
Sweden	+46 46 460 2828	support@altair.se
United Kingdom	+44 192 646 8600	support@uk.altair.com
United States	+1 248 614 2425	hwsupport@altair.com

If your company is being serviced by an Altair partner, you can find that information on our web site at https://www.altair.com/PartnerSearch/.

See www.altair.com for complete information on Altair, our team, and our products.

Contents

Solver Templates

1

Solver templates are ASCII data files containing HyperMesh Template Language Commands and HyperMesh Template Language Functions.

This chapter covers the following:

Export templates define the form of an ASCII output file and format the data in a HyperMesh database for finite element codes. The template commands are organized in blocks which are output in the order they are defined. Export templates also define card images for data in the HyperMesh database. The HyperMesh Card Image panel is then used to review and edit the data. HyperMesh Card Preview Commands are available for this purpose.

Export templates exist for each solver supported by HyperMesh and are located in sub-folders under the `<altair_home>/templates/feoutput` directory. They are loaded when the user profile is changed or from the Global panel.

Summary templates use the same commands and functions. However, instead of defining the format of data and output files, they define how to summarize data in the database. Examples include a component summary, mass calculations and load summaries. Summary templates are utilized in the Summary panel.

1.1 Card Previewer

The HyperMesh Card Image panel is used to define solver specific data and review card information. The format of the cards and the menus in the panel are defined in the export template, using HyperMesh Card Preview Commands.

Export templates exist for each solver supported by HyperMesh and are located in subfolders under the `<altair_home>/templates/feoutput` directory. They are loaded when the user profile is changed or from the Global panel. A card image for each entity type is defined within the export template. Each entity type can have its own associated card image with appropriate options.

A sample card image for an OptiStruct `MAT1` material collector is shown below:

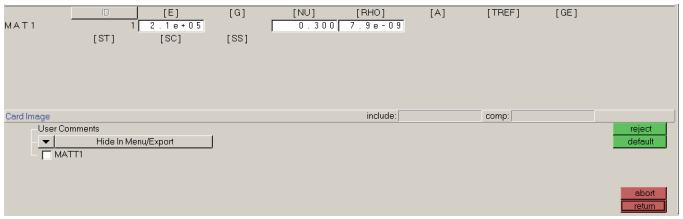

Figure 1:

For each item in the above card image, the export template contains HyperMesh Card Preview Commands to define how the data is defined and reviewed in HyperMesh. A portion of the OptiStruct export template for the above card image is shown here:

```
*beginmenu()
    *menustring("MAT1     ")
    *menufield(ID,integer,id,8)
    *menufield(E,real,$E,8)
        *menudefaultvalue("         ")
        *menuinitialvalue(210000.0)
        *menurestrictedvalue(>,0.0)
    *menufield(G,real,$G,8)
        *menudefaultvalue("         ")
        *menuinitialvalue(80769.2)
        *menurestrictedvalue(>,0.0)
    *menufield(NU,real,$Nu,8)
        *menudefaultvalue("         ")
        *menurestrictedvalue(>,-1.0)
        *menurestrictedvalue(<,0.5)
        *menuinitialvalue(0.3)
    *menufield(RHO,real,$Rho,8)
        *menudefaultvalue("         ")
        *menuinitialvalue(7.85e-09)
    *menufield(A,real,$MAT1_A,8)
        *menudefaultvalue("         ")
        *menufield(TREF,real,$MAT1_TREF,8)
        *menudefaultvalue("         ")
    *menufield(GE,real,$MAT1_GE,8)
        *menudefaultvalue("         ")
    *menulineend()

    *menustring("         ")
    *menufield(ST,real,$MAT1_ST,8)
        *menudefaultvalue("         ")
```

```
    *menufield(SC,real,$MAT1_SC,8)
        *menudefaultvalue("        ")
    *menufield(SS,real,$MAT1_SS,8)
        *menudefaultvalue("        ")
    *menulineend()
*endmenu()
```

1.2 Creating Solver Templates

1.2.1 Organization

A template file is organized into blocks and when outputting a database, HyperMesh processes a template on a per-block basis. Blocks are output in the order that they are defined in the template file.

You define the beginning of a block with a scan command (`*card`, `*component`, `*elements`, and so on) and then close the block with the `*output()` command.

As an entity is requested for output, HyperMesh reads the information from the database. Within each block there are five independent levels that you can utilize when exporting information to the ASCII file (coded numerically):

1. before() - At this level, you can define a series of commands that HyperMesh processes before moving to the next level. The before() level is executed once for each data type requested. Use this level to add comments and set up parameters on the following levels. Some of the data in the database is available at this level.
2. beforecollector() - At this level, HyperMesh scans the database for the requested data type. Each time HyperMesh finds the requested data type collector, it executes the beforecollector() level. Use this level to add comments about the data and create groups. Collector data is available at this level.
3. format() - At this level, HyperMesh processes the commands defined for each entity contained in a collector. The data associated with the printed entity is available, and the format required for each entity is defined at this level.
4. aftercollector() - After the data entities within a collector have been processed, HyperMesh goes to the aftercollector() level. The aftercollector() level is processed after each collector is scanned. Only the collector data is available at this level.
5. after() - The after() level is similar to the before() level. HyperMesh executes the after() level once after HyperMesh scans the database.

The before() and after() blocks are executed even if none of the entities specified exist in the database. Database information (except for global data) is only available on levels two through four.

Nodes

There ares several steps to output nodes.

1. HyperMesh executes the before() level.
2. HyperMesh executes the beforecollector() level using the only node collector in the database.
3. The format() level is executed once for each node in the database.
4. After processing the nodes, HyperMesh executes the aftercollector() level and the after() level.

When outputting nodes, the before() and beforecollector() levels perform the same operation. The after() and aftercollector() levels also perform the same operation. This is because there is no data associated with the collector.

Named Entities

There ares several steps to output named entities.

1. HyperMesh executes the before() level followed by the beforecollector() level.
2. Since the HyperMesh database does not have a collector collector, the beforecollector() level is a duplicate of the before() level and is immediately executed once.
3. HyperMesh executes the format() level for each of the named entities in the database. All of the data is available.
4. The aftercollector() and after() levels are processed similarly to the before() and beforecollector() levels in step 1.

Collected Entities

There ares several steps to output collected entities.

1. HyperMesh executes the before() level before scanning the database.
2. HyperMesh examines each collector for an entity that matches the requested type. When HyperMesh finds a collector, the beforecollector() level is processed. Collector data is available at this level.
3. HyperMesh executes the format() level once for each entity in the collector that matches the requested type.
4. The aftercollector() and after() levels are processed similarly to the before() and beforecollector() levels in step 1.

1.2.2 Data Entry and Access

Template files allow you to access information from the HyperMesh database with two methods, data names and attributes.

Template files use data names to access information from the HyperMesh database. A data name is a string that represents a piece of data. At output, HyperMesh replaces the data name string with the value that the data name represents. For example, a node has id, x, y, z, and system as possible data names in the template files. If you enter the command `*field(integer,id,8)` into a template file, HyperMesh outputs the node ID in an integer format using eight spaces.

A data name can also represent a pointer to another entity in the database. In the element data name list, the data name `node1` is a pointer. `node1` points to a node in the database. Using the command `*field(integer,node1,8)`, HyperMesh issues the error message "field statement references a pointer". HyperMesh can’t output the correct value because node1 points to a node entity that has many different possible values. To print the node ID, reference the pointer as `*field(integer,node1.id,8)`. A period separates the data name pointer `node1` and the data name `id`.

Template files additionally use attributes to describe solver-specific data. This data is created using the HyperMesh card image system and may be attached to entities within the HyperMesh database. Attributes are defined in the HyperMesh template system by giving them unique IDs for each solver. This is done by giving each solver a number through the use of the template command `*codename()`. Each attribute can be accessed either through its unique ID or the name that the user gives the attribute. Attributes are defined using the `*defineattribute()` command. An example for the OptiStruct MAT1 material is:

```
*defineattribute(E,1,real,none)
*defineattribute(G,2,real,none)
*defineattribute(Nu,3,real,none)
*defineattribute(Rho,4,real,none)
```

The template files also allow for advanced features to be defined from within the template like counters, tables and variables. This allows information to be passed from one function to another.

Comments can be added to the template using //. Add as many comments as needed to the template file in order to describe what is being coded and to allow other developers to follow your logic.

1.2.3 Mathematical Expressions

Template files can also contain mathematical expressions.

For example, to translate a model during output, apply a formula to the x-coordinate of a node:

```
*nodes()
    *format()
        *string("node,")
        *field(integer,id,10)
        *string(",")
        *field(real,[x+100.0],10)
        *string(",")
        *field(real,y,10)
        *string(",")
        *field(real,z,10)
    *end()
*output()
```

100.0 is added to the x-coordinate of the nodes and is sent to the formatter, where it is written to the file. The square brackets, [], around the formula tell the parser that the enclosed text is a formula and not to interpret the asterisk, *, as a new command.

The following operators are available for mathematical expressions:

Addition	+
Subtraction	-
Multiplication	*
Division	/

Modulus	%

1.2.4 Equalities, Inequalities and Logical Expressions

Templates contain functions to test values, perform conditional statement execution, or control the template flow.

These statements include `*if` and `*loopif`. The tests in these commands use the following syntax to determine if the statement is true or false.

These statements can be linked to logical expressions:

Equal to:	==
Not equal to:	!=
Less than:	<
Greater than:	>
Less than or equal to:	<=
Greater than or equal to:	>=
Logical and:	&&
Logical or:	||

1.2.5 Functions

The HyperMesh Template Functions are available to query information in the database, perform mathematical functions, and query user-defined tables. A template function begins with the symbol @ and is followed by the variable arguments in parentheses. Template functions are always placed between square brackets [].

The following example finds the length of all weld elements and outputs that information to a file:

```
*elements(3,0,"","")
    *format()
        *field(integer,id,10)
        *string(" ")
        *field(real,[@magnitude(node1.x - node2.x,node1.y - node2.y,node1.z -
 node2.z)],10)
    *end()
*output()
```

1.2.6 Previewing Cards

Card images are described by a template file block. You can display the card image using the card previewer.

To display the card image using the card previewer, specify the appropriate card previewer commands within a `*beginmenu()/*endmenu()` block. You can use these commands to display entity data and edit solver-specific attributes.

Since the card previewer is used to view and edit attributes on multiple entities, some data may not have a common value to display for the selected entities . When this occurs, a large X is drawn through the data display area. If the data is an editable field (attribute), select the field and enter a value that is common to all of the selected entities. If the entities do not share a common value, HyperMesh ignores any logic commands (`*menuif()`, `*menuoption()`, `*menuoptionenum()`, and so on) dependent on entities sharing a common value.

Error Messages

The following list describes the card previewer error messages and the corresponding solutions.

Message	Invalid variable <variable> in `*setvariable()` command.
Meaning	The **<variable>** specified in the `*menusetvariable()` command was not between variable1 and variable20.
Solution	Change the invalid variable to a valid variable name.
Message	Attribute id **<id>** on entity does not match type in template.
Meaning	The attribute's type did not match the type specified for the attribute in the template.
Cause	The `*defineattribute()` command in the template may have been modified, a different template with the same `*codename()` was used to edit the entity, or an invalid entity was created by an input translator.
Solution	You must clear the attributes for the solver off of the entity to edit the card.
Message	Invalid entity type **<type>** specified in `*menuentitytype()`.

Meaning	A `*menuentitytype()` command with invalid type was present in the template file.
Solution	Change the `*menentitytype()` command's parameter to be a valid entity name.

Message	Too many **<type>** collectors used.
Meaning	A maximum of 48 `*menuentitytype()` commands for the same entity type can be used in a card image. You have exceeded this limit.
Solution	Reduce the number of `*menuentitytype()` commands for **<type>** to be less than 48.

Message	Could not find entity **<id>** associated with attribute **<attribute name>**
Meaning	An entity attribute holds an id without an entity.
Cause	Entities may have been deleted or renumbered, or the `*menuentitytype()` command specifying the entity collected by this attribute was changed in the template file.
Solution	None. The value is set to 0.

Message	No attribute attached to menu item **<name>**.
Meaning	An internal error has occurred while parsing an expression.
Solution	Contact HyperMesh support. The data and template file are required to further investigate the problem.

Message	No attribute attached to collector item **<id>**.

Meaning	An internal error has occurred when a collector was selected.
Solution	Contact HyperMesh support. The data and template file are required to further investigate the problem.

Message	Could not find attribute named **<name>** in `*menuoption()`, skipping.
Meaning	A `*menuoption()` or `*menuoptionenum()` command referenced **<name>** that does not have an `*defineattribute()` command.
Solution	Change the `*menuoption()` or `*menuoptionenum()` command to reference a valid attribute name.

Message	Could not find enumeration **<enumeration>** for `*menuoptionenum()`.
Meaning	`*menuoptionenum()` command referenced an **<enumeration>** that does not have an existing `*enumeration()` command.
Solution	Change the `*menuoptionenum()` command to reference a valid enumeration.

Message	Enumeration attribute **<attribute>** contains value **<value>** beyond limit of **<maximum>**.
Meaning	**<attribute>** holds a **<value>** > **<maximum>**.
Cause	Both *menuoptionenum() and `*menufield()` commands referencing **<attribute>** may exist in the template file. If this is the case, the user can type in **<value>** > **<maximum>**. An input translator may have also created **<attribute>** with the invalid **<value>**.
Solution	If this occurs in a HyperMesh-developed template, contact HyperMesh support with the error. User

	generated templates can ensure that the value is within a certain range by using `*menuif()` and `*menuattributeset()` commands. Here is an example of how to limit an attribute's value to between 1 and 5.
	`*menuif([$ATTRIBUTE_NAME < 1])`
	`*menuattributeset($ATTRIBUTE_NAME,1)`
	`*menuendif()`
	`*menuif([$ATTRIBUTE_NAME > 5])`
	`*menuattributeset($ATTRIBUTE_NAME,5)`
	`*menuendif()`

Message	Could not find attribute named **<attribute>**, skipping.
Meaning	A `*menufield()` command referenced the attribute named **<attribute>** for which no `*defineattribute()` command exists.
Solution	Change the `*menufield()` command to reference a valid attribute name.

Message	Default value specified for always on attribute <attribute>.
Meaning	A `*menudefaultvalue()` command modifies **<attribute>** that does not have a valid on/off value. The `*menudefaultvalue()` command will be ignored.
Cause	The `*menudefaultvalue()` command in the template may have been added after the template was used on the current database, a different template with the same `*codename()` was used to edit the entity, or **<attribute>** was created with an invalid status by an input translator.

Message	Failed to created attribute **<attribute>**.
Meaning	An internal error has occurred in the card editor.

Solution	Contact HyperMesh support. The data and template file are required to further investigate the problem.

Message	Initial value expression for **<attribute>** could not be evaluated.
Meaning	The expression specified in `*menuinitialvalue()` returned an error code. The **<attribute>** is created with the value 0 or a zero length string, depending on string.
Cause	This error should only occur if you using the card editor on multiple entities.
Solution	If you need the `*menuinitialvalue()` command to be executed, abort the editing on the current set of entities and edit them one at a time.

Message	Initial value for **<attribute>** only valid for integer, real, or string.
Meaning	A `*menuinitialvalue()` command references an attribute that was not of type integer, real, or string.
Solution	Remove the `*menuinitialvalue()` command that references **<attribute>**.

Message	Attribute **<attribute>** has different entity type (**<entity>**) than template.
Meaning	A `*menuentitytype()` command conflicts with the entity type stored on **<attribute>**.
Cause	The `*menuentitytype()` command modifying **<attribute>** in the template may have been changed after the template was used on the current database, two or more `*menufield()` commands referencing **<attribute>** with differing `*menuentitytype()` commands exists in the template file, a different template with the

	same `*codename()` was used to edit the entity, or **<attribute>** was created with an invalid entity type by an input translator.
Solution	If a user-generated input translator is being used, make sure that attribute's entity type created by the translator matches that of the template file. Also check for and remove multiple `*menufield()`/`*menuentitytype()` commands referencing the same attribute in the block.

Message	Failed to created entity attribute **<attribute>**.
Meaning	An internal error has occurred in the card editor.
Solution	Contact HyperMesh support. The data and template file are required to further investigate the problem.

Message	Enumerated and legal input specified for **<attribute>**, legal ignored.
Meaning	Both `*menulegalvalue()` and `*menuenum()` commands modify the same `*menufield()` command. The `*menulegalvalue()` commands are ignored.
Solution	Remove either the `*menuenum()` command or all of the `*menulegalvalue()` modifiers from the `*menufield()` in question.

Message	Enumerated and restricted input specified for **<attribute>**, restricted ignored.
Meaning	Both `*menurestrictedvalue()` and `*menuenum()` commands modify the same `*menufield()` command. The `*menurestrictedvalue()` commands are ignored.

Solution	Remove either the `*menuenum()` command or all `*menurestrictedvalue()` modifiers from the `*menufield()` in question.

Message	Restricted and legal input specified for **<attribute>**, restricted ignored.
Meaning	Both `*menurestrictedvalue()` and `*menulegalvalue()` commands modify the same `*menufield()` command. The `*menurestrictedvalue()` commands are ignored.
Solution	Remove either all `*menulegalvalue()` commands or all `*menurestrictedvalue()` modifiers from the `*menufield()` in question.

Message	`menulegalvalue` unsupported for storage type of **<attribute>**.
Meaning	A `*menulegalvalue()` command modifies a `*menufield()` command that references an attribute that is not of type real, integer, or string.
Solution	Remove the `*menulegalvalue()` modifier from the `*menufield()` command.

Message	Illegal unsigned integer value returned for **<data>**.
Meaning	An expression or data member value could not be displayed as an unsigned integer. The string "ERROR" is displayed in red for this field.
Solution	Change the display type of this `*menufield()` command to real or exponential.

Message	Illegal integer value returned for **<data>**.

Meaning	An expression or data member value could not be displayed as an integer. The string "ERROR" is displayed in red for this field.
Solution	Change the display type of this `*menufield()` command to real or exponential.

Message	Illegal hexadecimal value returned for **<data>**.
Meaning	An expression or data member value could not be displayed as a hexadecimal number. The string "ERROR" is displayed in red for this field.
Solution	Change the display type of this `*menufield()` command to real or exponential.

Message	No string data returned for **<data>**.
Meaning	An expression or data member value could not be displayed as a string. The string "ERROR" is displayed in red for this field.
Solution	Change the display type of this `*menufield()` command to something other than string.

Message	beginrepeat - Expression **<expression>** could not be evaluated, skipping repeat block.
Meaning	**<expression>** could not be evaluated for multiple entities.
Solution	Reduce set of entities to use card editor until **<expression>** is resolvable.

Message	Illegal repeat value return for **<expression>**.
Meaning	A `*beginrepeat()` or `*beginrepeat2d()` command containing **<expression>** returned a

	value too large or too small. The repeat block is skipped.
Solution	It is possible to get this error with valid data. Usually it is caused by a bad `*menupointerset()` command.

Message	Illegal repeat 2d value return for **<expression>**.
Meaning	A `*beginrepeat()` or `*beginrepeat2d()` command containing **<expression>** returned a value too large or too small. The repeat block is skipped.
Solution	It is possible to get this error with valid data. Usually it is caused by a bad `*menupointerset()` command.

Message	Invalid pointer name **<pointer>** in `*menupointerset()` command, ignoring.
Meaning	The **<pointer>** specified in the `*menupointerset()` command was not between pointer1 and pointer20.
Solution	Change the invalid pointer to a valid pointer name.

Message	Invalid counter name **<counter>** in `*menucounterset()` command, ignoring.
Meaning	The **<counter>** specified in the `*menucounterset()` command was not between pointer1 and pointer20.
Solution	Change the invalid pointer to a valid counter name.

Message	Error evaluating data for **<counter>**, setting to 0.

Meaning	The expression in a `*menucounterset()` command could not be evaluated for this set of entities.
Solution	Reduce the set of entities you are editing until the expression in `*menucounterset()` command can be evaluated.

Message	Invalid variable name **<variable>** in `*menuvariableset()` command, ignoring.
Meaning	The **<variable>** specified in the `*menuvariableset()` command was not between variable1 and variable20.
Solution	Change the invalid variable to a valid variable name.

Message	Error evaluating data for **<variable>**, setting to 0.
Meaning	The expression in a `*menuvariableset()` command could not be evaluated for this set of entities.
Solution	Abort editing the current set of entities, reduce the set of entities you are editing until the expression in `*menuvariableset()` command is evaluated.

Message	Unable to find attribute **<attribute>** in `*menuattributeset()` command.
Meaning	A `*menuattributeset()` command references an illegal attribute name.
Solution	Change the attribute referenced by the `*menuattributeset()` command.

Message	Could not find selected item **<id>**.

Meaning	An internal error has occurred in the card editor.
Solution	Contact HyperMesh support. The current model and template file are required to further investigate the problem.

Message	Could not find tag attribute **<attribute>**.
Meaning	A `*cardmenuitem()` command references an attribute that has no `*defineattribute()` command.
Solution	Change **<attribute>** so that it references a valid attribute, or add a `*defineattribute()` command for **<attribute>**.

Message	**menuenum** references undefined enumeration **<name>**.
Meaning	An enumeration called **<name>** was not previously defined in the template file.
Solution	Change **<name>** to match a valid enumeration name.

Message	Output command reached before `*beginmenu()` for this entity.
Meaning	The `*output()` command for this entity block, signifying the end of the block, was reached before a `*beginmenu()` command. Either the `*beginmenu()` does not exist in this entity block, or it was placed after the `*output()` command.
Solution	Add or move the `*beginmenu()/*endmenu()` section before the `*format()` command for this block.

Message	2d array attribute <**attribute**> found outside of `*repeat2d block()`.
Meaning	Attributes of type arrayofreal2d and arrayofinteger2d can only be referenced in a `*beginrepeat2d()/*endrepeat2d()` section.
Solution	Move the reference to <**attribute**> into a `*beginrepeat2d()/ *endrepeat2d()` section.

Message	2d array attribute <**attribute**> not resized properly (out of bounds).
Meaning	An internal error has occurred in the card editor causing the array attribute to have the wrong size.
Solution	Contact HyperMesh support. The data and template file are required to further investigate the problem.

Message	Unknown attribute type <**type number**> found in attribute table.
Meaning	An internal error has occurred in the card editor causing the array attribute to have the wrong size.
Solution	Contact HyperMesh support. The data and template file are required to further investigate the problem.

Message	`*endmenu()` command not reached for this entity.
Meaning	The end of the template file was reached before the `*endmenu()` command.
Solution	Add a `*endmenu()` command to match the last `*beginmenu()` command in template file.

Message	`*menuif()` missing `menuelse()/menuendif()` command, aborting.
Meaning	The end of the template file was reached before a `*menuelse()` or `*menuendif()` command for a `*menuif()` statement.
Solution	Add a `*menuendif()` command to match the `*menuif()` command. Since the `*menuif()` commands can be nested, it may not be obvious which `*menuif()` is missing a `*menuelse()` or `*menuendif()` command.

Message	`*menuendif()` command not found to terminate `*menuelse()` block.
Meaning	The end of the template file was reached before a `*menuendif()` command was found to terminate a `*menuif()` and `*menuelse()` statement.
Solution	Add a `*menuendif()` command to match the `*menuif()` command. Since the `*menuif()` commands can be nested, it may not be obvious which `*menuif()` is missing a `*menuelse()` or `*menuendif()` command.

Message	Menu does not own pointer returned by strrgy.
Meaning	An internal error has occurred in the card editor.
Solution	Contact HyperMesh support. The data and template file are required to further investigate the problem.

Message	Option attribute **<attribute>** not of type integer.
Meaning	**<attribute>** referenced in a `*menuoption()` or `*menuoptionenum()` command was not an integer attribute.

Solution	Changed the `*menuoption()` or `*menuoptionenum()` command so that it references an integer attribute.

Message	Negative optioncount reached in case search block, aborting.
Meaning	Too many `*menuoptionend()` commands exist in this block. There should be one `*menuoptionend()` command per `*menuoption()` or `*menuoptionenum()` command.
Solution	Locate and remove the extra `*menuoptionend()` commands.

Message	Invalid attribute **<attribute>** referenced in `*beginrepeat()` command.
Meaning	The name **<attribute>** does not match a name in a `*defineattribute()` command.
Solution	Change **<attribute>** to match the name of a valid attribute.

Message	`*beginrepeat()` attribute **<attribute>** is not type integer.
Meaning	Attributes specified in the `*beginrepeat()` command must be of type integer.
Solution	Change the `*beginrepeat()` command to reference an integer attribute.

Message	No `*endrepeat()` for zero length array.
Meaning	The end of the template file was reached before a `*endrepeat()` command was found to match a `*beginrepeat()`.

Solution	Add a `*endrepeat()` command to match the `*beginrepeat()` command.

Message	`*endrepeat()` found without matching `*beginrepeat()`.
Meaning	An extra `*endrepeat()` command exists in this block. There should be one `*endrepeat()` command per `*beginrepeat()` command.
Solution	Locate and remove the extra `*endrepeat()` command.

Message	`*beginrepeat2d()` found outside of `*beingrepeat()` block.
Meaning	All `*beginrepeat2d()` commands must occur within a `*beginrepeat()/*endrepeat()` block.
Solution	Move the `*beginrepeat2d()/*endrepeat2d()` block so that it occurs in a `*beginrepeat()/*endrepeat()` block.

Message	Invalid attribute **<attribute>** referenced in `*beginrepeat2d()` command.
Meaning	The name **<attribute>** does not match a name in a `*defineattribute()` command.
Solution	Change **<attribute>** to match the name of a valid attribute.

Message	`*beginrepeat2d()` attribute **<attribute>** is not type integer.
Meaning	Attributes specified in the `*beginrepeat2d()` command must be of type integer.

Solution	Change the `*beginrepeat2d()` command to reference an integer attribute.

Message	No `*endrepeat()` for zero length array.
Meaning	The end of the template file was reached before a `*endrepeat()` command was found to match a `*beginrepeat()`.
Solution	Add an `*endrepeat()` command to match the `*beginrepeat()` command.

Message	`*endrepeat2d()` found without matching `*beginrepeat2d()`.
Meaning	An extra `*endrepeat2d()` command exists in this block. There should be one `*endrepeat2d()` command per `*beginrepeat2d()` command.
Solution	Locate and remove the extra `*endrepeat2d()` command.

1.2.7 Node Output Example

This example demonstrates how to use template files to generate output files for nodes.

Assume that a particular analysis code requires the nodes in the data deck to appear in the following format:

```
1       8       16       24       32       40
NODE  DATA
NODE         <id>      <x>      <y>      <z>*
.
.
.
END NODES
```

The following template generates the necessary output:

```
*text()
  *string("1       8       16       24       32       40")
  *end()
*output()

*nodes()
```

```
  *before()
    *string("NODE DATA")
    *end()
  *format()
    *string("NODE     ")
    *field(integer,id,8)
    *field(real,x,8)
    *field(real,y,8)
    *field(real,z,8)
    *string("*")
    *end()
  *after()
    *string("END NODES")
    *end()
*output()
```

The `*text()` command indicates the beginning of a simple text block that does not require database access.

The `*nodes()` command indicates the beginning of a node block. The following commands format all node entities in the HyperMesh database. The next command, `*before()`, instructs HyperMesh to execute the following commands on the before() level. `*string()` and `*end()` instruct HyperMesh to output the string contained in double quotes and perform a carriage return, respectively. The next command informs HyperMesh to execute the following commands on the format() level, or with each of the entities (nodes in this case). `*string()` instructs HyperMesh to place the item in double quotes in the output file. The extra spaces after the word NODE allow you to define a width, since the `*string()` command does not allow width definition. The next command is the first data request from the database.

The `*field()` command is how template files communicate with the HyperMesh database. The `*field()` command instructs HyperMesh to scan the database and retrieve the next piece of information for the output file. `*field()` takes three parameters: the output type, the name of the data accessed, and the width of the generated field. The output type can be integer, real, exponential, string, or hexadecimal. In this case, the data type is an integer. The next parameter is the data name. ID indicates that HyperMesh places the value of the node ID into this field. The last parameter is the width of the field. In this example, all fields are formatted to eight characters. The next three `*field()` commands place x, y, and z into the output file formatted as reals. `*string()` places the trailing asterisk on the end of the line. The `*end()` command (carriage return) is the last command placed into the format() level.

The `*after()` command indicates the last process level. This command instructs HyperMesh to execute the following commands on the after() level. The `*string()` command places the string contained in double quotes in the output file and the `*end()` command terminates the current line.

The `*output()` command processes the node entity output requests made in the preceding block. If the `*output()` command is missing, HyperMesh does not print anything to the file.

In addition to this example, the default HyperMesh templates serve as examples.

1.2.8 Element Output Example

This example demonstrates how to use template files to generate output files for elements.

Assume that a particular analysis code requires the elements in the data deck to appear in the following format:

```
1       10        20        30        40       50
BEGIN ELEMENT DATA
  ELEMENTS GROUP = <name>
      <id>   <node1>   <node2>   <node3>  <node4>
  .
  .
  .
  END
.
.
.
END ELEMENTS
```

This example assumes that the analysis code requires grouped element output, where elements with the same property are placed into the same output element group, regardless of component organization.

The following template generates the necessary output:

```
*text()
  *string("1       10        20        30        40       50") *end()
*output()

*elements(104,1,"quads","property")
  *before()
    *string("BEGIN ELEMENT DATA") *end()
  *beforecollector()
    *string("  ELEMENTS GROUP = ") *field(string,collector.name,10) *end()
  *format()
    *field(integer,id,10) *field(integer,node1.id,10)
    *field(integer,node2.id,10)*field(integer,node3.id,10)
    *field(integer,node4.id,10) *end()
  *aftercollector()
    *string("  END") *end()
  *after()
    *string("END ELEMENTS") *end()
*output()
```

Note the placement of the *() commands. HyperMesh does not place restrictions on where the commands appear in the file, so you can format the template as necessary. Comments can also be added for documentation purposes. HyperMesh ignores file content until it finds an asterisk, *. A command is defined as the characters between the asterisk and the closing parenthesis,).

The `*elements()` command instructs HyperMesh that the following commands define an element output block or process. The parameters to the `*elements()` command are the configuration and type of element the output block applies to: a user-defined name for the elements, and a user-defined name for the property the elements require. Element configurations are listed in the following sections for each of the HyperMesh elements. Element types are a user-defined number associated with each HyperMesh element (defaults to 1). Changing the type allows multiple definitions for a HyperMesh element configuration. A quad is configuration 104 and type 1. When these parameters are supplied,

HyperMesh limits the output of this process to quad elements. The next series of commands, `*before()`, `*string()`, and `*end()` behave as described in the node example.

The `*beforecollector()` command instructs HyperMesh to process the following commands on the beforecollector() level. When the beforecollector() level is selected, HyperMesh processes the commands each time it finds a collector holding the required type of information in the database. In this case, `*string()`, `*field()`, and `*end()` are the commands HyperMesh processes when a component is found. On the `*field()` command, the data name is **`collector.name`**.

`*format()` describes the process for each element of configuration 104 and type 1. The format commands use the data name **`node1.id`**, **`node2.id`**, and so on. The period, ., is required because **`node1`** is a pointer to a node. **`node1.id`** is the ID of the node.

In addition to this example, the default HyperMesh templates serve as examples.

1.2.9 Assembly Output Example

This example demonstrates how to use template files to generate output files for assemblies.

```
*assemblies()
  *format()
    *string("ASSEMBLY: ")
    *field(integer,id,0)
    *string(" ")
    *field(quotedstring,name,0)
    *end()

    *counterset(counter1,0)
    *loopif([counter1 != numberofcomponents])
      *pointerset(pointer1,components,counter1)
      *string("   COMPONENT: ")
      *field(integer,pointer1.pointervalue,0)
      *string(" ")
      *field(quotedstring,pointer1.component.name,0)
      *end()
      *counterinc(counter1)
    *endloop()

    *counterset(counter1,0)
    *loopif([counter1 != numberofassemblies])
      *pointerset(pointer1,assemblies,counter1)
      *string("   SUBASSEMBLY: ")
      *field(integer,pointer1.pointervalue,0)
      *string(" ")
      *field(quotedstring,pointer1.assembly.name,0)
      *end()
      *counterinc(counter1)
    *endloop()

    *counterset(counter1,0)
    *loopif([counter1 != numberofmultibodies])
      *pointerset(pointer1,multibodies,counter1)
      *string("   MULTIBODY: ")
      *field(integer,pointer1.pointervalue,0)
      *string(" ")
      *field(quotedstring,pointer1.multibody.name,0)
      *end()
```

```
      *counterinc(counter1)
    *endloop()
*output()
```

1.3 Commands and Functions

1.3.1 Card Previewer Commands

*begincardmenu()

Indicates the beginning of the Control Cards list.

Syntax

`*begincardmenu` *()*

Type

HyperMesh Card Previewer Command

Example

Only `*cardmenuitem()` commands are valid between the `*begincardmenu()` and `*endcardmenu()` commands.

*beginmenu()

Indicates the beginning of the description used for the card previewer.

Syntax

`*beginmenu` *()*

Type

HyperMesh Card Previewer Command

Example

Must be accompanied by the `*endmenu()` command.

*beginrepeat()

Repeats execution of a block of code a specified number of times.

Syntax

`*beginrepeat` *(expression)*

Type

HyperMesh Card Previewer Command

Inputs

expression
A relational expression or an attribute.

Example

```
*beginrepeat($ARRAY_LENGTH)
 *repeatwrap(80)
 *repeatcounter(1)
 *menufield(Array,integer,$ARRAY_ATTRIBUTE,10)
*endrepeat()
```

The `*repeatwrap()` command can be used to automatically begin new lines. If used, it must be the first command following the `*beginrepeat()`.

The `*repeatcounter()` command can be used to store the current repeat value in a counter. If no `*repeatcounter()` command is specified, the repeat value is not stored in any counter.

All `*menufield()` commands that reference attributes can only reference the following types in a `*beginrepeat()` block: arrayofinteger, arrayofreal, arrayofstring.

The `*beginrepeat()` command must be accompanied by a `*endrepeat()` command.

*beginrepeat2d()

Repeats execution of a block of code a specified number of times.

Syntax

`*beginrepeat2d` *(expression)*

Type

HyperMesh Card Previewer Command

Inputs

expression
Either a relational expression or an attribute.

Example

```
*beginrepeat($ARRAY_LENGTH)
*repeatwrap(80)
*repeatcounter(1)
*menufield(Array,integer,$ARRAY_ATTRIBUTE,10)
 *beginrepeat2d($ARRAY2D_LENGTH)
 *repeatcounter(2)
  *menufield(Array2d,integer,$ARRAY2D_ATTRIBUTE,10)
 *endrepeat2d()
```

```
*endrepeat()
```

The `*beginrepeat2d()` command must always occur with a `*beginrepeat()` /`*endrepeat()` block.

The `*repeatwrap()` command can be used to automatically begin new lines. If used, it must be the first command following the `*beginrepeat2d()`. If a `*repeatwrap()` command is specified following a `*beginrepeat()` command, that same value is used for the `*beginrepeat2d()` block.

The `*repeatcounter()` command can be used to store the current repeat value in a counter. If no `*repeatcounter()` command is specified, the repeat value is not stored in any counter.

All `*menufield()` commands that reference attributes can only reference the following types in a `*beginrepeat2d()` block: arrayofinteger2d or arrayofreal2d.

The `*beginrepeat2d()` command must be accompanied by a `*endrepeat2d()` command.

*cardmenuitem()

Specifies the name and tag attribute for a Control Card.

Syntax

`*cardmenuitem` *(button text, attribute name)*

Type

HyperMesh Card Previewer Command

Inputs

button text
: The text that is displayed to you in the Control Cards menu. Should be less than 60 characters in length.

attribute name
: The tag attribute for a Control Card entity. This must have an accompanying *card(attribute name) block in the template file.

Example

The `*cardmenuitem()` commands specify the list that appears in the Control Cards menu. If the list is too large to display, Prev/Next and First/Last buttons are added to the menu so you can page through all options.

The Control Card buttons are displayed with different colors, depending on their status and their existence in the database. Items with gray text do not exist in the database. Items with red text exist in the database, but are inactive and are not written to a file when exporting data. Items with green text exist in the database and are active; and are output when exporting data. Only one instance of each type of Control Card can exist in the database.

*endcardmenu()

Indicates the end of the Control Cards list.

Syntax

`*endcardmenu` *()*

Type

HyperMesh Card Previewer Command

*endmenu()

Indicates the ending of the description used for the card previewer.

Syntax

`*endmenu` *()*

Type

HyperMesh Card Previewer Command

Example

Must be accompanied by the `*beginmenu()` command.

*endrepeat()

Terminates the code block to be executed by `*beginrepeat()`.

Syntax

`*endrepeat` *()*

Type

HyperMesh Card Previewer Command

*endrepeat2d()

Terminates the code block to be executed by `*beginrepeat2d()`.

Syntax

`*endrepeat2d` *()*

Type

HyperMesh Card Previewer Command

*enumeration()

Creates an enumeration.

Syntax

`*enumeration` *(enum name, str1, str2,...)*

Type

HyperMesh Card Previewer Command

Inputs

enum name
The name of the enumeration.

str1, str2
The members of the enumeration.

*globaldefaults()

Used to specify that each real, integer, and string attribute is modified as per the `*menudefaultvalue()`.

Syntax

`*globaldefaults` *()*

Type

HyperMesh Card Previewer Command

Example

Any default value not overridden by a `*menudefaultvalue()` fills the field with the number of blanks equal to the width parameter in the `*menufield()` command when the field has the status off. `*menudefaultvalue()` can still be used to specify a different default value for individual fields when needed.

*menuattributecreate()

Initializes an attribute to a specified value upon creation of an entity.

Syntax

`*menuattributecreate` *(attribute, expression)*

Type

HyperMesh Card Previewer Command

Inputs

attribute
The name of the attribute to hold the value.

expression
An expression defining the value.

Example

`*menuattributecreate()` can only be used for integer, real, entity, or string attributes.

*menuattributeset()

Sets the value of attribute.

Syntax

`*menuattributeset` *(attribute, expression)*

Type

HyperMesh Card Previewer Command

Inputs

attribute
The name of the attribute to hold the value.

expression
An expression defining the value.

Example

`*menuattributeset()` can only be used for integer, real, entity, or string attributes.

*menucase()

The `*menucase()` command specifies a block of card image template commands that are executed when an attribute has the value matching the `*menucase()` command.

Syntax

`*menucase` *(value)*

Type

HyperMesh Card Previewer Command

Inputs

value
Must be an integer value.

Example

`*menucase()` commands must occur in consecutive, incremental order.

The values 0 and 1 are the only legal values for a `*menuoption()` command.

The values 1 to N are legal for a `*menuoptionenum()` command where N is the number of values in the enumeration referenced by `*menuoptionenum()`.

*menucounterset()

Sets the value of the local counter.

Syntax

`*menucounterset` *(counter, value)*

Type

HyperMesh Card Previewer Command

Inputs

counter
: Value from counter1 to counter20 indicating which of the 20 possible counters should be set to the *value* parameter.

value
: Value of the counter.

Example

Unlike the `*counterset()` command that sets a global value for the entire output section of the template, the `*menucounterset()` command's value is only for the local scope of an entity as specified by the `*beginmenu()` and `*endmenu()` commands.

*menudefaultvalue()

Modifies an attribute field so that it has an on/off status.

Syntax

`*menudefaultvalue` *(string)*

Type

HyperMesh Card Previewer Command

Inputs

string
: The text that is displayed when the status of the attribute is off.

Example

```
*menufield("MISC",integer,$INT_ATTRIBUTE,10)
*menudefaultvalue("XXXXXXXXXX")
*menufield("MISC 2",real,$REAL_ATTRIBUTE,8)
*menudefaultvalue("no value")
```

string should be equal in length to the *width* in the `*menufield()` command.

This command must follow a `*menufield()` command that references an integer, real, or string attribute. If multiple entities are being displayed in the card previewer and the status of any attribute on different entities conflict, the title is displayed in red and the value for this attribute is displayed as a large X. The title must be selected to make all entities have the same status for this attribute, which is off. The title for the field can be selected to change the on/off status for the attribute.

If the attribute is off, the title is displayed in yellow and the string specified in the `*menudefaultvalue()` command is displayed. If the attribute is on, the title is displayed in cyan and an input field is displayed. When the attribute is on, you can select the input field and type in a value for this `*menufield()`.

An attribute's status can be accessed in the output section of the template file using the `@defaultstatus()` command.

*menuelse()

Used to define the false block of a `*menuif()` statement.

Syntax

`*menuelse` *()*

Type

HyperMesh Card Previewer Command

*menuendif()

Used to define the end of an `*menuif()` block.

Syntax

`*menuendif` *()*

Type

HyperMesh Card Previewer Command

*menuentitysubtype()

Modifies an attribute field so that the contents of the selection must be of the specified entity type in the HyperMesh database.

Syntax

`*menuentitysubtype` *(entity_type)*

Type

HyperMesh Card Previewer Command

Inputs

entity_type

Example

entity_type must be a valid HyperMesh entity type.

This command can only be used following the sequence of one `*menufield` and one `*menuentitytype` command. It will only be effective if the argument to the preceding `*menuentitytype` is SETS, TITLES, OUTPUTBLOCKS, or TAGS; otherwise it will be ignored. The command may be used more than once. The following fragment is valid:

```
*menufield($description,integer,$name,$width)
  *menuentitytype(sets)
  *menuentitysubtype(elems)
  *menuentitysubtype(nodes)
  *menuentitysubtype(comps)
```

`*menuentitysubtype` indicates that only sets, titles, and so on that contain entities of the given type may be selected for this field in the card previewer. In the preceding example, you may select sets that contain elements, nodes or components only. If `*menuentitysubtype` does not follow `*menuentitytype`, it is assumed that all types are allowed. Multiple instances of `*menuentitysubtype` allow multiple subtypes.

*menuentitytype()

Modifies an attribute field so that the selection must be an ID of an entity of a specified type in the HyperMesh database.

Syntax

`*menuentitytype` *(entity_type)*

Type

HyperMesh Card Previewer Command

Inputs

entity_type

Example

entity_type must be a valid HyperMesh entity type.

The `*menuentitytype()` command cannot modify a `*menufield()` along with a `*menuenum()`, a `*menurestrictedvalue()` , a `*menulegalvalue()` , or a `*menudefaultvalue()` command.

This command follows a `*menufield()` command that references an entity attribute. This changes the appearance of the field in the card previewer to a collector of the specified type.

*menuenum()

Modifies an attribute field so that its value is displayed as and restricted to an enumeration's values.

Syntax

`*menuenum` *(enumeration)*

Type

HyperMesh Card Previewer Command

Inputs

enumeration

The name of a previously defined enumeration. Instead of an input field, the field appears as a button. When selected, the enumeration's values are displayed in a pop-up. After selected, the position of the value in the enumeration is stored in the attribute (1 to N).

Example

The `*menuenum()` command can only be used to modify integer attributes.

The `*menuenum()` command cannot modify a `*menufield()` along with a `*menuentitytype()` , a `*menurestrictedvalue()` , a `*menulegalvalue()` , or a `*menudefaultvalue()` command.

*menufield()

Places a formatted value from the database into the card image.

Syntax

`*menufield` *(description, type, data name, width)*

Type

HyperMesh Card Previewer Command

Inputs

description

A string that is displayed along with the value. The length of the string should not be longer than *width*.

type
Either integer, unsigned, real, exponential, string, or hexadecimal.

data name
The name of the data to be accessed.

width
The width of the formatted field. In the case of real, HyperMesh uses scientific notation in order to make the value displayed fit in the specified number of characters.

*menuif()

Used to conditionally execute branches in the card image.

Syntax

`*menuif` *(expression)*

Type

HyperMesh Card Previewer Command

Inputs

expression
A relational expression.

Example

Requires an `*menuendif()` command.

The expression is required to be enclosed in square brackets.

The following operators are available:

==, =	equal
!=	not equal
<=	less than or equal
<	less than
>=	greater than or equal
>	greater than

If multiple entities are being displayed and an `*menuif()` returns conflicting results for the *expression*, the entire `*menuif()`/ `*menuelse()` / `*menuendif()` block is skipped.

*menuinitialarrayvalue()

Sets the initial values of an attribute.

Syntax

`*menuinitialarrayvalue` *(value)*

Type

HyperMesh Card Previewer Command

Inputs

value
Can be an expression for a real or integer attribute, or a string for string attributes.

Example

This command works only for attributes declared as arrayofints, arrayofreals, or arrayofstrings. Initially, all values in the array are set to the specified value. After the initial values are set, whenever the array size is increased, all new items in the array are set to the new value.

*menuinitialvalue()

Sets the initial value of an attribute if the attribute does not currently exist on the entity.

Syntax

`*menuinitialvalue` *(value)*

Type

HyperMesh Card Previewer Command

Inputs

value
Can be an expression for real or integer attributes, or a string for string attributes.

Example

May be combined with `*menulegalvalue()`, `*menuenum()` or `*menurestrictedvalue()` modifiers. If the attribute already exists on the entity, its value is not changed by the command. This command must follow a `*menufield()` command that references an integer, real, or string attribute.

*menulegalvalue()

Modifies an attribute field so that a selection must be made from a list.

Syntax

`*menulegalvalue` *(string)*

Type

HyperMesh Card Previewer Command

Inputs

string

The string that is added to the list you can select from for this `*menufield()`. The length of *string* in characters should be less than or equal to the *width* specified in the `*menufield()` command.

Example

Below is an example of the `*menulegalvalue()` command modifying each of the three attribute types:

```
*menufield("INT FIELD",integer,$INT_ATTRIBUTE,10)
*menulegalvalue(1)
*menulegalvalue(2)
*menulegalvalue(4)
*menulegalvalue(8)
*menufield("REAL FIELD",real,$REAL_ATTRIBUTE,10)
*menulegalvalue(1.1)
*menulegalvalue(1.2)
*menulegalvalue(3.0)
*menufield("STRING FIELD",string,$STRING_ATTRIBUTE,10)
*menulegalvalue("String 1")
*menulegalvalue("String 2")
*menulegalvalue("Last String")
```

If the `*menufield()` attribute is of type integer, the *string* specified in the `*menulegalvalue()` command is converted to an integer and stored in the attribute.

If the `*menufield()` attribute is of type real, the *string* specified in the `*menulegalvalue()` command is converted to a real number and stored in the attribute.

If the `*menufield()` attribute is of type string, the *string* specified in the `*menulegalvalue()` command is copied to the attribute.

The `*menulegalvalue()` cannot modify a `*menufield()` along with a `*menuenum()`command or a `*menurestrictedvalue()` command.

Multiple `*menulegalvalue()` commands may be applied to the same `*menufield()` command.

*menulineend()

Ends the current line in the card image.

Syntax

`*menulineend`

()

Type

HyperMesh Card Previewer Command

Description

Card images always require at least 1 `*menulineend()` command. The next command that displays a field displays it in column 1 on the next line.

*menuoption()

Specifies that the value of an attribute can only be 0 or 1.

Syntax

`*menuoption` *(attribute)*

Type

HyperMesh Card Previewer Command

Inputs

attribute

The name of the integer attribute that holds the value. The value stored in *attribute* is either 0 (off) or 1 (on).

Example

```
*menuoption($Option_Attribute)
*menucase(0)
*menustring("Option is off")
*menucase(1)
*menustring("Option is on")
*menuoptionend()
```

You must have a `*menucase()` for both values, 0 and 1. All of the commands between the `*menucase()` statement that are equal to the value stored in *attribute* and the next `*menucase()` or `*menuoptionend()` statement are executed.

The attribute's value is displayed in the options (bottom) portion of the menu as a diamond toggle.

*menuoptionend()

Terminates a `*menuoption()` or `*menuoptionenum()` command.

Syntax

`*menuoptionend` *()*

Type

HyperMesh Card Previewer Command

*menuoptionenum()

Specifies that the value of an attribute is defined by a previously defined `*enumeration()` command.

Syntax

`*menuoptionenum` *(attribute, enumeration)*

Type

HyperMesh Card Previewer Command

Inputs

attribute
: The name of the integer attribute that holds the value.

enumeration
: The name of a previously defined enumeration. The value stored in *attribute* is converted to an integer based on the selection's place in the enumeration and has a value from one to N (where N is the number of items in the enumeration).

Example

```
*enumeration(Numbers,One,Two,Three)
*menuoptionenum($Numbers_Attribute,Numbers)
*menucase(1)
*menustring("One (1)")
*menucase(2)
*menustring("Two (2)")
*menucase(3)
*menustring("Three (3)")
*menuoptionend()
```

For each value specified in the `*enumeration()` statement that *enumeration* references, you must have a `*menucase()` statement. All of the commands between the `*menucase()` statement that are equal to the value stored in *attribute* and the next `*menucase()` or `*menuoptionend()` statement are executed.

The attribute's value is displayed in the options (bottom) portion of the menu as a selector. You can choose from the values in the enumeration.

*menupointerset()

Sets the initial value of a pointer.

Syntax

`*menupointerset` *(pointer number, pointer, value)*

Type

HyperMesh Card Previewer Command

Inputs

pointer number
: A value from pointer1 to pointer20 indicating which of the 20 possible pointers should be set to the *value* parameter.

pointer
: The pointer to the data object to be accessed. Only certain data types may user pointers. These are described in the template commands where they are valid.

value
: The value of the counter.

Example

For example, to display all the dependent node IDs on a rigid link element, the following commands could be used while in a rigid link card description:

```
*beginrepeat(dependentnodesmax)
  *repeatcounter(1)
  *menupointerset(pointer1,dependentnodes,[counter1 - 1])
  *menufield(dnod,integer,pointer1.pointervalue,8)
*endrepeat()
```

Unlike the `*pointerset()` command that sets a global value for the entire output section of the template, the `*menupointerset()` command's value is only for the local scope of an entity as specified by the `*beginmenu()` and `*endmenu()` commands.

*menurestrictedvalue()

Modifies an attribute field so that it alerts you when a value is outside of a specified range.

Syntax

`*menurestrictedvalue` *(restriction, value)*

Type

HyperMesh Card Previewer Command

Inputs

restriction
Must be one of the following: >, >=, <=, or <.

value
The value that is used for comparison with the value that you entered.

Example

The following is an example on how to restrict a real attribute to values > 0.0 and <= 1.0:

```
*menufield("MISC",real,$REAL_ATTRIBUTE,10)
*menurestrictedvalue(>,0.0)
*menurestrictedvalue(<=,1.0)
```

`*menurestrictedvalue()` cannot modify a `*menufield()` along with a `*menuenum()` or a `*menulegalvalue()` command.

This command must follow a `*menufield()` command that references an integer or real attribute. If you specify a value outside of the range specified by the `*menurestrictedvalue()` value, an error message is displayed and the value is displayed in red. Multiple `*menurestrictedvalue()` commands may be applied to the same `*menufield()` command, but if restricted to the same boundary, only the last one is used.

*menustring()

Displays a string in the card image.

Syntax

`*menustring` *(string)*

Type

HyperMesh Card Previewer Command

Inputs

string
A string of characters. If the string contains a space, an asterisk, or a comma, the string must be enclosed by double quotes.

*menuvariableset()

Sets a variable to a specific value.

Syntax

`*menuvariableset` *(variable, value)*

Type

HyperMesh Card Previewer Command

Inputs

variable

A value from variable1 to variable20 indicating the parameter.

value

The value of the variable.

Example

Variables can be used to hold real or integer values. For example, to add the value of variable5 to the current value of variable1, the following command could be used:

```
*menuvariableset(variable1,[variable1+variable5])
```

Unlike the `*variableset()` command that sets a global value for the entire output section of the template, the `*menuvariableset()` command's value is only for the local scope of an entity as specified by the `*beginmenu()` and `*endmenu()` commands.

*nomenu()

Specifies that the card image definition is in a following block.

Syntax

`*nomenu`*()*

Type

HyperMesh Card Previewer Command

Example

The `*nomenu()` command must be the first command to follow an entity block header.

*repeatcounter()

Specifies a counter to store the current repeat value for a `*beginrepeat()` or `*beginrepeat2d()` command.

Syntax

`*repeatcounter` *(counter number)*

Type

HyperMesh Card Previewer Command

Inputs

counter number
An integer value between 1 and 20 indicating which of the 20 possible counters should be hold the current repeat value.

*repeatwrap()

Sets the right margin so that no field goes beyond in a `*beginrepeat()`/`*endrepeat()` block.

Syntax

`*repeatwrap` *(column)*

Type

HyperMesh Card Previewer Command

Inputs

column
The right hand margin beyond which no field is displayed. If a field's length would place it beyond *column*, it is placed in column 1 of the next line.

Example

If used, the `*repeatwrap()` command must be the first command to follow a `*beginrepeat()` or `*beginrepeat2d()` command.

Undocumented Card Previewer Commands

The list of undocumented card previewer commands.

`*globalmenuminimumstringlength()`

`*menudefault()`

`*menuentitypointerset()`

`*menuhelp()`

*globalmenuminimumstringlength()

*menudefault()

*menuentitypointerset()

*menuhelp()

1.3.2 Solver Template Commands

*accelerometers()

Starts an accelerometers block.

Syntax

`*accelerometers` *(config)*

Type

HyperMesh Template Command

Description

Starts an accelerometers block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all accelerometers with the format:

*accelerometers(id,"name")

```
*accelerometers()
  *format()
    *string("*accelerometers(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
```

```
    *string(")")
    *end()
*output()
```

Version History

14.0.130

*addblock()

Adds a block to a link.

Syntax

`*addblock` *(name)*

Type

HyperMesh Template Command

Inputs

name

*addcomment()

Sets the characters of strings which are considered comment lines.

Syntax

`*addcomment` *("comment_string")*

Type

HyperMesh Template Command

Description

Sets the characters of lines which are considered comment lines. These strings are then used for export/do not export of comments.

Inputs

comment_string
: The complete string or the starting characters of the string.

Example

To add as comment lines that start with "**H" (**HWCOLOR COMP, **HMNAME, and so on) except "**HM_UNSUPPORTED_CARDS_START":

```
*addcomment("**H")
*ignorecomment("**HM_UNSUPPORTED_CARDS_START")
```

Version History

11.0.101

*after()

Indicates that the commands following are processed on the `*after()` level.

Syntax

`*after` *()*

Type

HyperMesh Template Command

*aftercollector()

Indicates that the commands following are processed on the `*aftercollector()` level.

Syntax

`*aftercollector` *()*

Type

HyperMesh Template Command

*alefsiprojections()

Starts an alefsiprojections block.

Syntax

`*alefsiprojections` *(config)*

Type

HyperMesh Template Command

Description

Starts an alefsiprojections block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all alefsiprojections with the format:

*alefsiprojections(id,"name")

```
*alefsiprojections()
  *format()
    *string("*alefsiprojections(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*alereferencesystemcurves()

Starts an alereferencesystemcurves block.

Syntax

`*alereferencesystemcurves` *(config)*

Type

HyperMesh Template Command

Description

Starts an alereferencesystemcurves block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all alereferencesystemcurves with the format:

*alereferencesystemcurves(id,"name")

```
*alereferencesystemcurves()
  *format()
    *string("*alereferencesystemcurves(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
```

```
    *end()
*output()
```

Version History

2019

*alereferencesystemgroups()

Starts an alereferencesystemgroups block.

Syntax

`*alereferencesystemgroups` *(config)*

Type

HyperMesh Template Command

Description

Starts an alereferencesystemgroups block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all alereferencesystemgroups with the format:

*alereferencesystemgroups(id,"name")

```
*alereferencesystemgroups()
  *format()
    *string("*alereferencesystemgroups(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*alereferencesystemnodes()

Starts an alereferencesystemnodes block.

Syntax

`*alereferencesystemnodes` *(config)*

Type

HyperMesh Template Command

Description

Starts an alereferencesystemnodes block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all alereferencesystemnodes with the format:

*alereferencesystemnodes(id,"name")

```
*alereferencesystemnodes()
  *format()
    *string("*alereferencesystemnodes(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*alereferencessystemswitches()

Starts an alereferencessystemswitches block.

Syntax

`*alereferencessystemswitches` *(config)*

Type

HyperMesh Template Command

Description

Starts an alereferencessystemswitches block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all alereferencessystemswitches with the format:

*alereferencessystemswitches(id,"name")

```
*alereferencessystemswitches()
  *format()
    *string("*alereferencessystemswitches(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*alesmoothings()

Starts an alesmoothings block.

Syntax

`*alesmoothings` *(config)*

Type

HyperMesh Template Command

Description

Starts an alesmoothings block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all alesmoothings with the format:

*alesmoothings(id,"name")

```
*alesmoothings()
  *format()
    *string("*alesmoothings(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*aletanktests()

Starts an aletanktests block.

Syntax

`*aletanktests` *(config)*

Type

HyperMesh Template Command

Description

Starts an aletanktests block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all aletanktests with the format:

*aletanktests(id,"name")

```
*aletanktests()
  *format()
    *string("*aletanktests(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
```

```
    *end()
*output()
```

Version History

2019

*assemblies()

Starts an assembly output block. All the assemblies in the HyperMesh database are output according to the user-defined format in this block.

Syntax

`*assemblies` *(assembly name)*

Type

HyperMesh Template Command

Inputs

assembly name
Used as a key to distinguish between different types of assemblies.

Example

To output the components in an assembly, the `*pointerset()` command must be used to retrieve the component IDs:

```
*counterset(counter1,0)
*loopif([counter1 != numberofcomponents])
*pointerset(pointer1,components,counter1)
*field(integer,pointer1.pointervalue,0)
*counterinc(counter1)
*endloop()
```

Requires an `*output()` command at the end of the block.

*attachmentcontrols()

Starts an attachmentcontrols block.

Syntax

`*attachmentcontrols` *(config)*

Type

HyperMesh Template Command

Description

Starts an attachmentcontrols block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all attachmentcontrols with the format:

*attachmentcontrols(id,"name")

```
*attachmentcontrols()
  *format()
    *string("*attachmentcontrols(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*attachments()

Starts an attachments block.

Syntax

`*attachments` *(config)*

Type

HyperMesh Template Command

Description

Starts an attachments block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all attachments with the format:

*attachments(id,"name")

```
*attachments()
  *format()
    *string("*attachments(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*bags()

Starts a bag output block.

Syntax

*bags

()

Type

HyperMesh Template Command

Description

This command starts a bag output block. All bag entities in the HyperMesh database are output according to the user-defined format in this block.

Example

The following code outputs the ID and coordinates of every node that exists in all bags of configuration 1.

```
*bags()
 *format()
 *setcurrentbagentitytype(nodes)
 *if([config == 1])
   *if([entitylistmax > 0])
     *string("$")
     *end()
     *string("$  GRID Data For ")
     *field(string,name,0)
     *string(": ")
     *end()
     *string("$")
     *end()
   *endif()
```

```
    *counterset(counter1,0)
    *loopif([counter1 < entitylistmax])
      *pointerset(pointer1,entitylist,counter1)
      *string("GRID     ")
      *fieldleft(integer,pointer1.pointervalue,8)
      *variableset(variable1,[@getentityvalue(nodes,pointer1.pointervalue,x)])
      *field(real,variable1,8)
      *variableset(variable1,[@getentityvalue(nodes,pointer1.pointervalue,y)])
      *field(real,variable1,8)
      *variableset(variable1,[@getentityvalue(nodes,pointer1.pointervalue,z)])
      *field(real,variable1,8)
      *end()
      *counterset(counter1,[counter1+1])
    *endloop()
    *end()
    *string("$--------------------------------------------------")
    *end()
    *string("$--------------------------------------------------")
    *end()
   *endif()
*output()
```

*beamsectcols()

Starts a beamsectcols block.

Syntax

`*beamsectcols` *()*

Type

HyperMesh Template Command

Description

Starts a beamsectcols block.

This command must be accompanied by a `*output()` command at the end of the block.

Example

To write out all beamsectcols with the format:

*beamsectcols(id,"name",color)

```
*beamsectcols()
*format()
*string("*beamsectcols(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,color,0)
*string(")")
*end()
*output()
```

*beamsects()

Starts a beamsects block.

Syntax

`*beamsects` *()*

Type

HyperMesh Template Command

Description

Starts a beamsects block.

This command must be accompanied by a `*output()` command at the end of the block.

Example

To write out all beamsects with the format:

*beamsects(id,setid,"name",config)

```
*beamsects()
*format()
*string("*beamsects(")
*field(integer,id,0)
*string(",")
*field(integer,setid,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,config,0)
*string(")")
*end()
*output()
```

*before()

Indicates that the commands following are processed on the `*before()` level.

Syntax

`*before` *()*

Type

HyperMesh Template Command

*beforecollector()

Indicates that the commands following are processed on the `*before()` collector level.

Syntax

`*beforecollector` *()*

Type

HyperMesh Template Command

*beginlink()

Starts a link.

Syntax

`*beginlink` *(type, name)*

Type

HyperMesh Template Command

Inputs

type

name

Example

Links can be used to tie output blocks together.

*blocks()

Starts a finite difference block output block. All the blocks in the HyperMesh database are output according to the user-defined format in this block.

Syntax

`*blocks` *()*

Type

HyperMesh Template Command

Inputs

name

Blocks with this card image name will be output.
Used as a key to distinguish different types of blocks.

Example

To output the locations of the i, j, or k divisions of a block, the `*pointerset()` command must be used:

```
*counterset(counter1,0)
*loopif([counter1 != divi])
  *pointerset(pointer1,idivisions,counter1)
  *field(real,pointer1.pointervalue,8)
  *counterinc(counter1)
*endloop()
To output the wall data for a block, the following commands may be used:
*counterset(counter1,0)
*loopif([counter1 != wallsmax])
  *pointerset(pointer1,blockwall,counter1)
  *field(integer,pointer1.wallid,8)
  *field(string,pointer1.wallname,0)
  *field(integer,pointer1.wallcolor,8)
  *counterinc(counter1)
*endloop()
```

Requires an `*output()` command at the end of the block.

*bodies()

Starts a bodies block.

Syntax

`*bodies` *(config)*

Type

HyperMesh Template Command

Description

Starts a bodies block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out standard mechanisms and their associated objects:

```
*mechanisms(1)
  *format()
    *string("*MECHANISM")
    *end()
    *fieldright(integer,id,10)
    *fieldleft(string,name,80)
    *end()
```

```
    *bodies()
      *format()
        *string("*BODY")
        *end()
        *fieldright(integer,id,10)
        *fieldleft(string,name,80)
        *end()
      *output()

      *joints()
        *format()
          *string("*JOINT")
          *end()
          *fieldright(integer,id,10)
          *fieldleft(string,name,80)
          *end()
      *output()

      *constraints()
        *format()
          *string("*CONSTRAINT")
          *end()
          *fieldright(integer,id,10)
          *fieldleft(string,name,80)
          *end()
      *output()

      *positions(1)
        *format()
          *string("*POSITION")
          *end()
          *fieldright(integer,id,10)
          *fieldleft(string,name,80)
          *end()
      *output()
*output()
```

Version History

14.0.120

*boxes()

Starts a boxes block.

Syntax

`*boxes` *(config)*

Type

HyperMesh Template Command

Description

Starts a boxes block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all boxes with the format:

*boxes(id,"name")

```
*boxes()
  *format()
    *string("*boxes(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

14.0

*cards()

Starts a Control Card output block.

Syntax

`*cards` *(card name)*

Type

HyperMesh Template Command

Inputs

card name
The name of the card to output.

Example

To output the *CTRL_TITLE* card the following commands may be used:

```
*cards("CTRL_TITLE")
 *format()
  *string("TITLE / ")
  *string(" ")
  *fieldleft(string,$TITLE_VAL,48)
*end()
*output()
```

Requires a `*output()` at the end of the block.

*codename()

Sets a unique solver number to be used for identifying attributes.

Syntax

`*codename` *(solver, identifier)*

Type

HyperMesh Template Command

Inputs

solver
: The name of the solver.

identifier
: A unique number identifying the solver.

Example

The `*codename()`command must occur before the first `*defineattribute()` command. All attributes created with this template are marked with this solver identifier.

Solver identifiers 0-63 are reserved for HyperMesh officially supported templates. Solver identifiers 64-127 are available for user-defined templates. If two templates are to share a set of attributes, they should have both the same solver identifier and the exact set of `*defineattribute()` commands.

*collections()

Starts a collections block.

Syntax

`*collections` *(config)*

Type

HyperMesh Template Command

Description

Starts a collections block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
: An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all collections with the format:

*collections(id,"name")

```
*collections()
  *format()
    *string("*collections(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Errors

Incorrect usage results in an import error.

*collisions()

Starts a collisions block.

Syntax

`*collisions` *(config)*

Type

HyperMesh Template Command

Description

Starts a collisions block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all collisions with the format:

*collisions(id,"name")

```
*collisions()
  *format()
    *string("*collisions(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
```

```
    *end()
*output()
```

Version History

2017.1

*comments()

Starts a comment block.

Syntax

`*comments` *(config)*

Type

HyperMesh Template Command

Description

Starts a comment block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all comments with the format:

*comments(id,"name")

```
*comments()
  *format()
    *string("*comments(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*components()

Starts a component block.

Syntax

`*components` *(card_image_name, mat_card_image_name)*

Type

HyperMesh Template Command

Description

Starts a component block. Components with a card image matching the specified card image name are considered for the block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

card_image_name
: A 32-character string enclosed in double quotes that defines the card image name of the components in the block. If specified as double quotes "", all components are considered for the block.
If a `*elements()` block has specified a *prop_card_image_name* value matching this string, the component collector to which those elements point is marked.

mat_card_image_name
: A 32-character string enclosed in double quotes that defines the card image name of the material that the components in the block require. If not needed, use empty double quotes "".
The name is also used to link to the `*materials()` commands.

Example

To write out all components with the format:

*component(id,"name",material ID,color,property ID)

```
*components("","")
*format()
*string("*component(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,materialid,0)
*string(",")
*field(integer,color,0)
*string(",")
*field(integer,propertyid,0)
*string(")")
   *end()
*output()
```

*compressreal()

Changes how real numbers are written to the output deck.

Syntax

`*compressreal` *(flag)*

Type

HyperMesh Template Command

Description

This command changes how real numbers are written to the output deck. The default value is 1. If used along with `*realprecision`, trailing zeros will be converted to blanks after the digits specified by `*realprecision`. For example, `*realprecision(4)` forces four digits to be written, even if they are zeros.

Inputs

flag

Trailing zeros are written as blanks.
Zero is not written to the left of the decimal.
In addition to performing compression, this command will add extra significant digits where applicable, while also rounding off precisely at the last digit. The compression is performed in accordance with the following rules and conditions:
This flag is applicable only for output of real numbers using any of the *field commands that have "real" as their first argument, and also have a non-zero width argument. Thus, the command will have no effect on the output produced using *field(exponential,value,8), or *field(real,value,0), for instance.
For values that are either too large or too small such that use of the 'E' is necessary to represent the number within the given width, the following conditions are applied:
By default, 'E' is omitted, and an extra significant digit is added, if applicable. However, 'E' can be output, by passing a custom export string to the command `*feouputwithdata`. Please refer to the documentation of that command for more details.
Any leading zeros on the exponential part are always omitted. Instead, additional significant digit(s) are output, where applicable.
For values that do not require the use of 'E', the following conditions apply:
For a number whose absolute value is less than 1.0, the zero to the left of the decimal point is omitted if doing so results in adding an extra significant digit.
For a whole number whose absolute value is greater than or equal to 1.0, the zero to the right of the decimal point is omitted if doing so results in a more accurate representation of the number within the given width.
Trailing zeros that do not contribute to the accuracy of a number are always replaced with blanks, leaving only one zero, at the most.
This flag also provides a means of rounding off real numbers whose absolute value is within a specified tolerance, to zero. This option is off by default, but can be turned on by passing a custom export string to the command `*feoutputwithdata`. Please refer to the documentation of that command for more details.

Example

Using `*compressreal(1)` causes HyperMesh to write numbers such as 0.0000 as 0.0.

Using `*compressreal(2)` causes HyperMesh to write numbers such as 0.123456 as .123456.

The table below has examples of output obtained using `*compressreal(3)`. The examples assume default options, that is, without the use of the custom strings mentioned above in `*feoutputwithdata`.

Actual value	Output in field of width 8	Output in field of width 16
0.125	0.125	0.125
0.1254768	.1254768	0.1254768
0.12547687	.1254769	0.12547687
0.125476842	.1254768	0.125476842
1.02356E+015	1.024+15	1.02356+15
-1.02356E+015	-1.02+15	-1.02356+15

*configurations()

Starts a configurations block.

Syntax

`*configurations` *(config)*

Type

HyperMesh Template Command

Description

Starts a configurations block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all configurations with the format:

*configurations(id,"name")

```
*configurations()
  *format()
    *string("*configurations(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

*connectorgroups()

Starts a connectorgroups block.

Syntax

`*connectorgroups` *(config)*

Type

HyperMesh Template Command

Description

Starts a connectorgroups block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all connectorgroups with the format:

*connectorgroups(id,"name")

```
*connectorgroups()
  *format()
    *string("*connectorgroups(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*connectorsets()

Starts a connectorsets block.

Syntax

`*connectorsets` *(config)*

Type

HyperMesh Template Command

Description

Starts a connectorsets block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all connectorsets with the format:

*connectorsets(id,"name")

```
*connectorsets()
  *format()
    *string("*connectorsets(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*constrainedextranodes()

Starts a constrainedextranodes block.

Syntax

`*constrainedextranodes` *(config)*

Type

HyperMesh Template Command

Description

Starts a constrainedextranodes block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all constrainedextranodes with the format:

*constrainedextranodes(id,"name")

```
*constrainedextranodes()
  *format()
    *string("*constrainedextranodes(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

14.0

*constrainedrigidbodies()

Starts a constrainedrigidbodies block.

Syntax

`*constrainedrigidbodies` *(config)*

Type

HyperMesh Template Command

Description

Starts a constrainedrigidbodies block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all constrainedrigidbodies with the format:

*constrainedrigidbodies(id,"name")

```
*constrainedrigidbodies()
*format()
*string("*constrainedrigidbodies(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(")")
*end()
*output()
```

Version History

14.0.120

*constraints()

Starts a constraints block.

Syntax

`*constraints` *(config)*

Type

HyperMesh Template Command

Description

Starts a constraints block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out standard mechanisms and their associated objects:

```
*mechanisms(1)
  *format()
    *string("*MECHANISM")
    *end()
    *fieldright(integer,id,10)
    *fieldleft(string,name,80)
    *end()

    *bodies()
```

```
      *format()
        *string("*BODY")
        *end()
        *fieldright(integer,id,10)
        *fieldleft(string,name,80)
        *end()
    *output()

    *joints()
      *format()
        *string("*JOINT")
        *end()
        *fieldright(integer,id,10)
        *fieldleft(string,name,80)
        *end()
    *output()

    *constraints()
      *format()
        *string("*CONSTRAINT")
        *end()
        *fieldright(integer,id,10)
        *fieldleft(string,name,80)
        *end()
    *output()

    *positions(1)
      *format()
        *string("*POSITION")
        *end()
        *fieldright(integer,id,10)
        *fieldleft(string,name,80)
        *end()
    *output()
*output()
```

Errors

Incorrect usage results in an import error.

*contactsurfs()

Starts a contactsurfs block.

Syntax

`*contactsurfs` *(card_image_name)*

Type

HyperMesh Template Command

Description

Starts a contactsurfs block. Contactsurfs with a card image matching the specified card image name are considered for the block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

card_image_name

A 32-character string enclosed in double quotes that defines the card image name of the contactsurfs in the block. If specified as double quotes "", all contactsurfs are considered for the block.

Example

To write out all contactsurfs with the format:

```
*contactsurfs(id,"name",color)
*contactsurfelement(id,facecode,reversecode)
*contactsurfelement(id,facecode,reversecode)
...
```

```
*contactsurfs("")
*format()
*string("*contactsurfs(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,color,0)
*string(")")
*if([facesmax != 0])
*end()
*endif()
*counterset(counter1, 0)
*loopif([counter1 < facesmax])
*pointerset(pointer1, faces, counter1)
*string("*contactsurfelement(")
*field(integer, pointer1.element.id, 0)
*string(",")
*field(integer, pointer1.facecode, 0)
*string(",")
*field(integer, pointer1.reversecode, 0)
*string(")")
*if([counter1 != facesmax-1])
*end()
*endif()
*counterinc(counter1)
*endloop()
*end()
*output()
```

*contactbehaviors()

Starts a contactbehaviors block.

Syntax

`*contactbehaviors` *(config)*

Type

HyperMesh Template Command

Description

Starts a contactbehaviors block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all contactbehaviors with the format:

*contactbehaviors(id,"name")

```
*contactbehaviors()
  *format()
    *string("*contactbehaviors(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020.1

*contactgroups()

Starts a contactgroups block.

Syntax

`*contactgroups` *(config)*

Type

HyperMesh Template Command

Description

Starts a contactgroups block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all contactgroups with the format:

*contactgroups(id,"name")

```
*contactgroups()
  *format()
    *string("*contactgroups(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017.1

*controlvols()

Starts a control volume block.

Syntax

`*controlvols` *(card_image_name,?idpool_name?)*

Type

HyperMesh Template Command

Description

Starts a control volume block. Control volumes with a card image matching the specified card image name are considered for the block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

card_image_name
: A 32-character string enclosed in double quotes that defines the card image name of the control volume. If specified as double quotes "", all control volumes are considered for the block.

idpool_name
: An optional 32-character string enclosed in double quotes that defines the name of the ID pool that the control volume belongs to. If not needed, use double quotes "" or omit the argument. The ID pool must be defined using the `*defineidpool()` command.

Example

To write out all control volumes with the format:

*controlvol(id,"name",entityid,type,elemidsmax,refnodesmax)

```
*controlvols("","")
*format()
*string("*controlvol(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,entityid,0)
*string(",")
*field(integer,type,0)
*string(",")
*field(integer,elemidsmax,0)
*string(",")
*field(integer,refnodesmax,0)
*string(")")
*end()
*output()
```

*counterinc()

Increments a counter.

Syntax

`*counterinc` *(counter)*

Type

HyperMesh Template Command

Inputs

counter
: Value from counter1 to counter20 indicating the counters to be incremented.

*counterset()

Sets the initial value of the global counter.

Syntax

`*counterset` *(counter, value)*

Type

HyperMesh Template Command

Inputs

counter
: Value from counter1 to counter20 indicating the counter(s) to set to the next parameter.

value
Value of the counter.

Example

Counters can be useful to specify continuation cards on some analysis codes. For an example, see theNastran template file in HyperMesh.

*createmergedloadloadsteptable()

Creates the table containing merged loads for a loadstep.

Syntax

`*createmergedloadloadsteptable` *()*

Type

HyperMesh Template Command

Description

Creates the table containing merged loads for a loadstep. The loads which are applied on the same entities and have the same configuration, type and load step will be merged. This is used for force, moment, velocity and acceleration only. This command will merge only when the combine loads in loadstep flag of custom export is utilized.

This command must be inside a `*loadsteps()` block and should be called in the `*before()` section. It must be followed by a call to `*deletemergedloadloadsteptable()`.

Example

To write out loadsteps using merged loads:

```
*loadsteps()
*before()
*createmergedloadloadsteptable()
*format()
...
*after()
*deletemergedloadloadsteptable()
*output()
```

Version History

11.0.101

*crosssections()

Starts a crosssections block.

Syntax

`*crosssections` *(config)*

Type

HyperMesh Template Command

Description

Starts a crosssections block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
: An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all crosssections with the format:

*crosssections(id,"name")

```
*crosssections()
  *format()
    *string("*crosssections(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

14.0

*curves()

Starts a curves block.

Syntax

`*curves` *()*

Type

HyperMesh Template Command

Description

Starts a curves block.

This command must be accompanied by a `*output()` command at the end of the block.

Example

To write out all curves with the format:

*curve(id,"name","title")

```
*curve(id,"name","title")
*curves()
*format()
*string("*curve(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(quotedstring,title,0)
*string(")")
*end()
*output()
```

*dampings()

Starts a damping block.

Syntax

`*dampings` *(config)*

Type

HyperMesh Template Command

Description

Starts a damping block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all dampings with the format:

*dampings(id,"name")

```
*dampings()
  *format()
```

```
    *string("*dampings(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019.1

*define()

Defines a dictionary item.

Syntax

`*define` *(name, type, value, active)*

Type

HyperMesh Template Command

Inputs

name
: The name of the dictionary item.

type
: The type of the dictionary item.

value
: The initial value assigned to the data item. This should be a number except if the item type is string.

active
: Determines the activity of the dictionary item.

Example

The available types are listed below:

none	No value is associated with the item.
string	The item has a string assigned to it.
Integer	The item has an integer assigned to it.
real	The item has a real value assigned to it.
-1	Always active.

0	Not active but user can toggle.
1	Active and user can toggle.

Items whose activity is set to -1 appear white on the dictionary edit menu. If the activity is set to 0 or 1, the menu item appears in cyan, and you may choose to star the item.

*defineattribute()

Defines an attribute for a solver.

Syntax

`*defineattribute` *(name, identifier, type, behavior)*

Type

HyperMesh Template Command

Inputs

name
The attribute name (maximum of 31 characters).

identifier
The number associated with the attribute. The z can be in the range of 1 to 32768.

type
The attribute type. Legal values are:

integer	Attribute contains an integer
arrayofinteger	Attribute contains an array of integer numbers
arrayofinteger2d	Attribute contains a 2d array of integer numbers
real	Attribute contains a floatingpointnumber
arrayofreal	Attribute contains an array of floating point numbers
arrayofreal2d	Attribute contains a 2d array of floating point numbers
string	Attribute contains a string
arrayofstring	Attribute contains an array of strings
entity	Attribute contains a reference to an entity (its ID and type)
arrayofentity	Attribute contains an array of entities

arrayofentity2d	Attribute contains a 2d array of entities

behavior
Determines how attributes are treated when the entity that owns it is changed. This field has not been implemented. The only value for *behavior* is none.

Example

All entities in the HyperMesh database may point to attributes. Attributes are defined with the `*defineattribute` command.

*defineentitytypealiasname()

Defines an entity type alias name for a solver.

Syntax

`*defineentitytypealiasname` *(entity_type, alias_name)*

Type

HyperMesh Template Function

Description

Defines an entity type alias name for a solver.

Inputs

entity_type
The entity type to create the alias for.

alias_name
The alias name.

Examples

To define an alias for components as "PART COMPONENT":

```
*defineentitytypealiasname(comps, "PART COMPONENT")
```

Version History

2019

*deletemassthicknesstable()

Deletes the element mass/thickness table.

Syntax

`*deletemassthicknesstable` *()*

Type

HyperMesh Template Command

Description

Deletes the element mass/thickness table created by `*populatemassthicknesstable()`.

This command must be inside either a `*components()` or `*elements()` block and should be called in the `*after()` section. It must be preceded by a call to `*populatemassthicknesstable()`.

Example

To write out the mass of every component in the format

id,"name",mass

```
*components("","")
*before()
*populatemassthicknesstable()
*format()
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(real,mass,0)
*end()
*after()
*deletemassthicknesstable()
*output()
```

Version History

11.0

*deletemergedloadloadsteptable()

Deletes the table containing merged loads for a loadstep.

Syntax

`*deletemergedloadloadsteptable` *()*

Type

HyperMesh Template Command

Description

Deletes the table containing merged loads for a loadstep created by `*createmergedloadloadsteptable()`.

This command must be inside a `*loadsteps()` block and should be called in the`*after()` section. It must be preceeded by a call to `*createmergedloadloadsteptable()`.

Example

To write out loadsteps using merged loads:

```
*loadsteps()
*before()
*createmergedloadloadsteptable()
*format()
...
*after()
*deletemergedloadloadsteptable()
*output()
```

Version History

11.0.101

*dequations()

Starts a dequations output block.

Syntax

`*dequations` *()*

Type

HyperMesh Template Command

Example

Requires a `*output()` at the end of the block.

*designpointmethods()

Starts a designpointmethod block.

Syntax

`*designpointmethods` *(config)*

Type

HyperMesh Template Command

Description

Starts a designpointmethod block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all designpointmethods with the format:

*designpointmethods(id,"name")

```
*designpointmethods()
  *format()
    *string("*designpointmethods(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019.1

*designpoints()

Starts a designpoint block.

Syntax

`*designpoints` *(config)*

Type

HyperMesh Template Command

Description

Starts a designpoint block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all designpoints with the format:

*designpoints(id,"name")

```
*designpoints()
  *format()
    *string("*designpoints(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019.1

*designpointsets()

Starts a designpointset block.

Syntax

`*designpointsets` *(config)*

Type

HyperMesh Template Command

Description

Starts a designpointset block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all designpointsets with the format:

*designpointsets(id,"name")

```
*designpointsets()
  *format()
    *string("*designpointsets(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
```

```
    *end()
*output()
```

Version History

2019.1

*designvars()

Starts a design variable output block.

Syntax

`*designvars` *()*

Type

HyperMesh Template Command

Example

Requires an `*output()` command at the end of the block.

*desvarlinks()

Starts a dlink output block.

Syntax

`*desvarlinks` *()*

Type

HyperMesh Template Command

Example

Requires an `*output()` command at the end of the block.

*directmatrixinputs()

Starts a direct matrix input block.

Syntax

`*directmatrixinputs` *(config)*

Type

HyperMesh Template Command

Description

Starts a direct matrix input block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all direct matrix inputs with the format:

*directmatrixinputs(id,"name")

```
*directmatrixinputs()
  *format()
    *string("*directmatrixinputs(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017

*dvprels()

Starts a dvprel output block.

Syntax

`*dvprels` *()*

Type

HyperMesh Template Command

Example

Requires an `*output()` command at the end of the block.

*elementareacalculation()

Determines how to calculate the area of an element.

Syntax

`*elementareacalculation` *(type, num)*

Type

HyperMesh Template Command

Inputs

type
: The element type. Only "quad4" is supported.

num
: Can be 1 (one point gaussian quadrature) or 4 (four point gaussian quadrature).

*elementclusters()

Starts an elementclusters block.

Syntax

`*elementclusters` *(config)*

Type

HyperMesh Template Command

Description

Starts an elementclusters block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
: An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all elementclusters with the format:

*elementclusters(id,"name")

```
*elementclusters()
  *format()
    *string("*elementclusters(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
```

```
    *string(")")
    *end()
*output()
```

Version History

2019

*elementresultstore()

Stores an element value for the current element.

Syntax

`*elementresultstore` *(value)*

Type

HyperMesh Template Command

Inputs

value
: The element value to be stored.

Example

```
*function("HM_CALC_TIMESTEP",variable16,variable17, variable18,variable19,variable20)
    *elements(60,0,"BEAM","")
        ... put result into variable1
        *elementresultstore(variable1)
    *output()
*return()
```

This command can be used in the template function, HM_CALC_TIMESTEP, to store the initial time step for each element. When time steps have been saved, an assigned plot can be created in the Check Elements panel.

This function must be called in the `*format()` section of an `*elements()` block.

*elements()

Starts an element block.

Syntax

`*elements` *(config, type, user_name, prop_card_image_name,?idpool_name?)*

Type

HyperMesh Template Command

Description

Starts an element block. Elements matching the specified config and type are considered for the block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

config
: Defines the element config that should be used for this block. If specified as 0, elements of all configs are considered for the block.

type
: Defines the element type that should be used for this block. If specified as 0, all elements of the specified config are considered for the block.

user_name
: A 32-character string enclosed in double quotes that defines the name of the element config/type combination.

prop_card_image_name
: A 32-character string enclosed in double quotes that defines the card image name of the property that the elements in the block require. If not needed, use empty double quotes "".
The name is also used to link to the `*components()` and `*properties()` commands.

idpool_name
: An optional 32-character string enclosed in double quotes that defines the name of the ID pool that the elements belong to. If not needed, use double quotes "" or omit the argument.
The ID pool must be defined using the `*defineidpool()` command.

Example

To write out all quad4 elements with the format:

*quad4(id,type,node 1 ID,node 2 ID,node 3 ID,node 4 ID,property ID)

```
*elements(104,0,"QUAD4","")
*format()
*string("*quad4(")
*field(integer,id,0)
*string(",")
*field(integer,type,0)
*string(",")
*field(integer,node1.id,0)
*string(",")
*field(integer,node2.id,0)
*string(",")
*field(integer,node3.id,0)
*string(",")
*field(integer,node4.id,0)
*string(",")
*if([propertyidflag == 1])
*field(integer,propertyid,0)
*else()
*string("0")
*endif()
*string(")")
   *end()
```

```
*output()
```

*else()

Used to define false conditions for an `*if()` block.

Syntax

`*else` *()*

Type

HyperMesh Template Command

Description

Used to define false conditions for an `*if()` block. Additional `*if()` commands can be used to create else-if conditions.

This command must be accompanied by a `*if()` command at the beginning of the block and a `*endif()` command at the end of the block.

Example

To output a portion of a CQUAD4 element with conditions depending on the property assignment:

```
*elements(104,1,"CQUAD4","")
*format()
*string("CQUAD4  ")
*field(integer,id,8)
*if([propertyid == 0])
*field(integer,collector.propertyid,8)
*else()
*field(integer,propertyid,8)
*endif()
*field(integer,node1.id,8)
*field(integer,node2.id,8)
*field(integer,node3.id,8)
*field(integer,node4.id,8)
   *end()
*output()
```

*enabledatabase()

Used to scan for entities in the HyperMesh database.

Syntax

`*enabledatabase` *(flag)*

Type

HyperMesh Template Command

Inputs

flag

The flag can be set to all or selected.
If you set the flag to all, the entity output commands (such as `*nodes()`) scan all entities in the database, even if you select displayed on the Export panel or Summary panel.
If you set the flag to selected, and you select displayed on the Export panel or Summary panel, the entity output commands scan the database for the displayed commands only.

Example

When writing export and summary templates, it may be necessary to scan the entire database before processing the displayed entities. To do this, use `*enabledatabase(all)`. To scan the database for only the entities that you selected (this can be all or displayed), use `*enabledatabase(selected)`. `*enabledatabase()` should be used outside of any other command blocks and can be used more than once.

*encryptions()

Starts an encryptions block.

Syntax

`*encryptions` *(config)*

Type

HyperMesh Template Command

Description

Starts an encryptions block.

This command must be accompanied by an `*output()` command at the end of the block

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all encryptions with the format:

*encryptions(id,"name")

```
*encryptions()
  *format()
    *string("*encryptions(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
```

```
*output()
```

Version History

14.0

*end()

Syntax

`*end` *()*

Type

HyperMesh Template Command

Example

This command ends the current line and places the output cursor at the beginning of the next line in the output file.

*endif()

Used to define the end of an `*if()` block.

Syntax

`*endif` *()*

Type

HyperMesh Template Command

Description

Used to define the end of an `*if()` block.

This command must be accompanied by a `*if()` command at the beginning of the block.

Example

To output a portion of a CQUAD4 element with conditions depending on the property assignment:

```
*elements(104,1,"CQUAD4","")
*format()
*string("CQUAD4  ")
*field(integer,id,8)
*if([propertyid == 0])
*field(integer,collector.propertyid,8)
*else()
*field(integer,propertyid,8)
*endif()
*field(integer,node1.id,8)
*field(integer,node2.id,8)
```

```
*field(integer,node3.id,8)
*field(integer,node4.id,8)
   *end()
*output()
```

*endlink()

Ends a link.

Syntax

`*endlink` *()*

Type

HyperMesh Template Command

*endloop()

Indicates the end of a block that was initialized with the `*loopif()` command.

Syntax

`*endloop` *()*

Type

HyperMesh Template Command

*endsegments()

Ends a segment block.

Syntax

`*endsegments` *()*

Type

HyperMesh Template Command

*entitypointerset()

Sets the initial value of a specified entity.

Syntax

`*entitypointerset` *(entity type, entity id, pointer number, pointer, value)*

Type

HyperMesh Template Command

Inputs

entity type
The type of entity, such as sets, elems, or nodes, to which *entity id* refers.

entity id
The ID of the entity you want to reference.

pointer number
Value from pointer1 to pointer10, indicating which of the 10 possible pointers to use as the *value* parameter.

pointer
Points to the data object that is accessed. Only certain data types may use pointers. These are described in the template commands in which they are valid.

value
The value of the pointer.

Example

`*entitypointerset()` can be used on any entity specified by *entity type* and *entity id*.

*equations()

Starts an equation output block. The equations in the HyperMesh database with a type equal to the *type* argument are output according to the user-defined format in this block.

Syntax

`*equations` *(type, user name)*

Type

HyperMesh Template Command

Inputs

type
The user-defined type that is output.

user name
The user-defined name for *type*.

Example

The following example outputs equations in a format similar to Abaqus:

```
*equations(0,"EQUATION")
*format()
  *string("*EQUATION") *end()
```

```
  *counterset(counter1,[independentnodesmax+1])
  *field(integer,counter1,0)
  *end()
  *field(integer,dependentnode.id,0)
  *string(",")
  *field(integer,dependentdof,0)
  *string(",")
  *field(real,dependentcoeff,0)
  *string(",")
  *counterset(counter1,0)
  *loopif([counter1 < independentnodesmax])
    *pointerset(pointer1,independentnodes,counter1)
    *field(integer,pointer1.pointervalue,0)
    *string(",")
    *pointerset(pointer1,independentdofs,counter1)
    *field(integer,pointer1.pointervalue,0)
    *string(",")
    *pointerset(pointer1,independentcoeffs,counter1)
    *field(real,pointer1.pointervalue,0)
    *counterinc(counter1)
  *endloop()
  *end()
*output()
```

*errormessage()

Displays an error message on the menu bar.

Syntax

`*errormessage` *(string)*

Type

HyperMesh Template Command

Inputs

string
: String to be displayed.

Example

The example below displays an error for each quad4 that has a jacobian less than .7:

```
*elements(104,0,"","")
  *format()
    *if([jacobian < .7])
      *errormessage("jacobian less than .7")
    *endif()
*output()
```

Each time `*errormessage()` is called, it overwrites the last error message. If the right mouse button is pressed while printing an error message, HyperMesh stops processing the template.

*executetclscript()

Executes a Tcl script from within a template, and optionally calls a procedure defined in the script.

Syntax

`*executetclscript` *(filename, proc_name)*

Type

HyperMesh Template Command

Description

Executes a Tcl script from within a template, and optionally calls a procedure defined in the script.

This command should be called in the `*before()` section.

Inputs

filename
: Full path of the Tcl file to be executed. The path should be always defined with two back slashes.

proc_name
: Name of the procedure to be executed. If not required, use two empty quotes.

Example

To execute a Tcl file

```
C:\TclScripts\test.tcl"
```

without a call to specific procedure:

```
*executetclscript("C:\\TclScripts\\test.tcl","")
```

To execute a Tcl file

```
C:\TclScripts\test.tcl
```

and call a procedure named TestProc:

```
*executetclscript("C:\\TclScripts\\test.tcl","TestProc")
```

*explorations()

Starts an explorations block.

Syntax

`*explorations` *(config)*

Type

HyperMesh Template Command

Description

Starts an explorations block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all explorations with the format:

*explorations(id,"name")

```
*explorations()
  *format()
    *string("*explorations(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*failures()

Starts a failures block.

Syntax

`*failures` *(config)*

Type

HyperMesh Template Command

Description

Starts a failures block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
: An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all failures with the format:

*failures(id,"name")

```
*failures()
  *format()
    *string("*failures(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*field()

Places a formatted value from the database into the output file.

Syntax

`*field` *(type, data name, width)*

Type

HyperMesh Template Command

Inputs

type
: Either integer, unsigned, real, exponential, string, hexadecimal, or quoted. For simplicity, the first letter is required; all others are optional but recommended.

data name
: The name of the data to be accessed.

width
: The width of the formatted field. In the case of real, HyperMesh uses scientific notation in order to make the value printed fit.

*fieldleft()

Places a left-justified, formatted value from the database into the output file.

Syntax

`*fieldleft` *(type, data name, width)*

Type

HyperMesh Template Command

Inputs

type
: Either integer, unsigned, real, exponential, string, hexadecimal, or quoted. For simplicity, the first letter is required; all others are optional but recommended.

data name
: The name of the data to be accessed.

width
: Width of the formatted field. In the case of real, HyperMesh uses scientific notation in order to make the value printed fit.

*fieldleftwithcomments()

Places a left-justified, formatted field name and value from the database into the output file. The field name is written just above the value line.

Syntax

`*fieldleftwithcomments` *(field_name, type, data_name, width)*

Type

HyperMesh Template Command

Description

Places a left-justified, formatted field name and value from the database into the output file. The field name is written just above the value line.

Inputs

field_name
: The solver field name of characters. If the string contains a space, an asterisk, or a comma, the string must be enclosed by double quotes. If the string name length is less than the width, spaces are appended after the string. If the string name length is more than the width, the name is truncated such that the length is equal to width.

type
Either integer, unsigned, real, exponential, string, hexadecimal, or quoted. For simplicity, the first letter is required; all others are optional but recommended.

data_name
The name of the data to be accessed.

width
Width of the formatted field. In the case of real, scientific notation is used in order to make the value fit.

Example

To write the left-justified field name "My field" with the integer data name myint using a width of 10:

```
*fieldleftwithcomments("My field",integer,myint,10)
```

Version History

2020

*fieldright()

Places a right-justified, formatted value from the database into the output file.

Syntax

`*fieldright` *(type, data name, width)*

Type

HyperMesh Template Command

Inputs

type
Either integer, unsigned, real, exponential, string, hexadecimal, or quoted. For simplicity, the first letter is required; all others are optional but recommended.

data name
The name of the data to be accessed.

width
The width of the formatted field. In the case of real, HyperMesh uses scientific notation in order to make the value printed fit.

*fieldrightwithcomments()

Places a right-justified, formatted field name and value from the database into the output file. The field name is written just above the value line.

Syntax

`*fieldrightwithcomments` *(field_name, type, data_name, width)*

Type

HyperMesh Template Command

Description

Places a right-justified, formatted field name and value from the database into the output file. The field name is written just above the value line.

Inputs

field_name
: The solver field name of characters. If the string contains a space, an asterisk, or a comma, the string must be enclosed by double quotes. If the string name length is less than the width, spaces are appended after the string. If the string name length is more than the width, the name is truncated such that the length is equal to width.

type
: Either integer, unsigned, real, exponential, string, hexadecimal, or quoted. For simplicity, the first letter is required; all others are optional but recommended.

data_name
: The name of the data to be accessed.

width
: Width of the formatted field. In the case of real, scientific notation is used in order to make the value fit.

Example

To write the right-justified field name "My field" with the integer data name myint using a width of 10:

```
*fieldrightwithcomments("My field",integer,myint,10)
```

Version History

2020

*fields()

Starts a fields block.

Syntax

`*fields` *(config)*

Type

HyperMesh Template Command

Description

Starts a fields block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all fields with the format:

*fields(id,"name")

```
*fields()
  *format()
    *string("*fields(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

14.0

*fieldwithcomments()

Places a formatted field name and value from the database into the output file. The field name is written just above the value line.

Syntax

`*fieldwithcomments` *(field_name, type, data_name, width)*

Type

HyperMesh Template Command

Description

Places a formatted field name and value from the database into the output file. The field name is written just above the value line.

Inputs

field_name
The solver field name of characters. If the string contains a space, an asterisk, or a comma, the string must be enclosed by double quotes. If the string name length is less than the width, spaces are appended after the string. If the string name length is more than the width, the name is truncated such that the length is equal to width.

type
Either integer, unsigned, real, exponential, string, hexadecimal, or quoted. For simplicity, the first letter is required; all others are optional but recommended.

data_name
The name of the data to be accessed.

width
Width of the formatted field. In the case of real, scientific notation is used in order to make the value fit.

Example

To write the -justified field name "My field" with the integer data name myint using a width of 10:

```
*fieldwithcomments("My field",integer,myint,10)
```

Version History

2020

*format()

Indicates that the following commands should be executed on the format level.

Syntax

`*format` *()*

Type

HyperMesh Template Command

*freebodygroups()

Starts a freebodygroup block.

Syntax

`*freebodygroups` *(config)*

Type

HyperMesh Template Command

Description

Starts a freebodygroup block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all freebodygroups with the format:

*freebodygroups(id,"name")

```
*freebodygroups()
  *format()
    *string("*freebodygroups(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019.1

*freebodysections()

Starts a freebodysection block.

Syntax

`*freebodysections` *(config)*

Type

HyperMesh Template Command

Description

Starts a freebodysection block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all freebodysections with the format:

*freebodysections(id,"name")

```
*freebodysections()
  *format()
    *string("*freebodysections(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019.1

*frictions()

Starts a friction block.

Syntax

`*frictions` *(config)*

Type

HyperMesh Template Command

Description

Starts a friction block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all frictions with the format:

*frictions(id,"name")

```
*frictions()
  *format()
    *string("*frictions(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
```

```
    *end()
*output()
```

Version History

2020

*function()

Starts a function block.

Syntax

`*function` *(name, variables)*

Type

HyperMesh Template Command

Inputs

name

The name of the function. Names should not begin with HM_ (these are reserved for use by HyperMesh).

variables

The variables to be returned to HyperMesh (variable1 - variable20). The number of variables depends on the function. Functions can be used by HyperMesh to calculate information needed in some panels.

Example

Obtain material data utilized for time step calculations for Abaqus shell and solid elements:

```
//variable1 = Young's modulus for the material
//variable2 = Poisson's ratio for the material
//variable3 = mass density for the material
*function("MATERIAL_DATA",variable1,variable2,variable3)
    *materials("")
        *format()
            *variableset(variable1,0)
            *variableset(variable2,0)
            *if([$UseElasticCard == 1])
                *if([$ElasticTypeEnumField <= 1]) //ISOTROPIC
                    *variableset(variable1,[@attributearrayvalue($Young,1)])
                    *variableset(variable2,[@attributearrayvalue($Poiss,1)])
                *endif()
                *if([$ElasticTypeEnumField == 2]) //ENGINEERING CONSTANTS
                    *variableset(variable1,[@attributearrayvalue($E1,1)])
                    *variableset(variable1,[variable1+@attributearrayvalue($E2,1)])
                    *variableset(variable1,[variable1+@attributearrayvalue($E3,1)])
                    *variableset(variable1,[variable1/3.0])
                    *variableset(variable2,[@attributearrayvalue($Nu12,1)])
                    *variableset(variable2,[variable2+@attributearrayvalue($Nu13,1)])
                    *variableset(variable2,[variable2+@attributearrayvalue($Nu23,1)])
                    *variableset(variable2,[variable2/3.0])
                *endif()
```

```
                *if([$ElasticTypeEnumField == 3]) //LAMINA
                    *variableset(variable1,[@attributearrayvalue($EL1,1)])
                    *variableset(variable1,[variable1+@attributearrayvalue($EL2,1)])
                    *variableset(variable1,[variable1/2.0])
                    *variableset(variable2,[@attributearrayvalue($NuL12,1)])
                *endif()
                *if([$ElasticTypeEnumField == 4 || $ElasticTypeEnumField == 5]) //
ORTHOTROPIC and ANISOTROPIC
                    *variableset(variable1,[@attributearrayvalue($D1111,1)])
                    *variableset(variable1,
[variable1+@attributearrayvalue($D2222,1)])
                    *variableset(variable1,
[variable1+@attributearrayvalue($D3333,1)])
                    *variableset(variable1,[variable1/3.0])
                *endif()
            *endif()
            *variableset(variable3,0)
            *if([$UseDensityCard == 1])
                *variableset(variable3,[@attributearrayvalue($Density,1)])
            *endif()
        *output()
*return()
```

*geometricrepresentations()

Starts a geometricrepresentation block.

Syntax

`*geometricrepresentations` *(config)*

Type

HyperMesh Template Command

Description

Starts a geometricrepresentation block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all geometricrepresentations with the format:

*geometricrepresentations(id,"name")

```
*geometricrepresentations()
  *format()
    *string("*geometricrepresentations(")
    *field(integer,id,0)
    *string(",")
```

```
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2021

*geometryoverride()

Overrides the lines option on the Export Data panel and activates lines for output.

Syntax

`*geometryoverride` *()*

Type

HyperMesh Template Command

*groups()

Starts a group output block. The groups in the HyperMesh database whose configuration and type are equal to the parameters are output according to the user-defined format in this block.

Syntax

`*groups` *(configuration, type, user name)*

Type

HyperMesh Template Command

Inputs

configuration

Defines the HyperMesh group that should be output using this block definition. The possible values are:

ConfigGroup	Output
1	Interface with master and slave elements
2	Interface with master elements and slave nodes
3	Interface with slave elements
4	Interface with slave nodes

ConfigGroup	Output
5	Rigid walls

type
Defines the group type that should be output using this block definition. The possible values are user-defined so that more types can be defined by the user. When groups are built, the template file is read automatically to determine the type of the group.

user name
A 32-character string enclosed in double quotes holding the name of the group as defined by the user. The name is used by the appropriate panels and displayed to the user for selection.

Example

Requires an `*output()` command at the end of the block.

*hm_features()

Starts a feature block.

Syntax

`*hm_features` *(config)*

Type

HyperMesh Template Command

Description

Starts a feature block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all features with the format:

*feature(id,"name")

```
*hm_features()
  *format()
    *string("*features(")
    *field(integer,id,0)
    *string(",")
```

```
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Errors

Incorrect usage results in an import error.

Version History

14.0.120

*hourglass()

Starts an hourglass block.

Syntax

`*hourglass` *(config)*

Type

HyperMesh Template Command

Description

Starts an hourglass block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all hourglasses with the format:

*hourglass(id,"name")

```
*hourglass()
  *format()
    *string("*hourglass(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017.1

*if()

Used to conditionally execute branches of code.

Syntax

`*if` *([expression])*

Type

HyperMesh Template Command

Description

Used to conditionally execute branches of code. This is useful for testing and acting on a condition. The following operators are available:

==, =	equal
!=	not equal
<=	less than or equal
<	less than
>=	greater than or equal
>	greater than
%	modulus

This command must be accompanied by a `*endif()` command at the end of the block.

Inputs

expression
: Relational expression. This must be enclosed in square brackets.

Example

To output a portion of a CQUAD4 element with conditions depending on the property assignment:

```
*elements(104,1,"CQUAD4","")
*format()
*string("CQUAD4  ")
*field(integer,id,8)
*if([propertyid == 0])
*field(integer,collector.propertyid,8)
*else()
*field(integer,propertyid,8)
*endif()
*field(integer,node1.id,8)
*field(integer,node2.id,8)
```

```
*field(integer,node3.id,8)
*field(integer,node4.id,8)
   *end()
*output()
```

*ignorecomment()

Sets the characters of strings which are ignored as comment lines.

Syntax

`*ignorecomment` *("comment_string")*

Type

HyperMesh Template Command

Description

Sets the characters of lines which are ignored as comment lines. These strings are then ignored for export/do not export of comments.

Inputs

comment_string
: The complete string or the starting characters of the string.

Example

To add as comment lines that start with "**H" (**HWCOLOR COMP, **HMNAME, etc...) except "**HM_UNSUPPORTED_CARDS_START":

```
*addcomment("**H")
*ignorecomment("**HM_UNSUPPORTED_CARDS_START")
```

Version History

11.0.101

*ignoreelemconfigtype()

Sets the characters of strings which are ignored as comment lines.

Syntax

`*ignoreelemconfigtype` *(config, type)*

Type

HyperMesh Template Command

Description

By default, within the ANSYS profile, a check is performed during certain operations to see if the ET type (sensor) associated with a component matches the config/type of element within the component. This command can be used to skip that check for specific element config/types. It is applicable only for the ANSYS user profile.

Inputs

config

Defines the element config that should be used for this command.

type

Defines the element type that should be used for this command.

Example

To ignore the ET type from element type of configuration 5 and type 0:

```
*ignoreelemconfigtype(5,0)
```

Version History

14.0.110

*include()

Includes a file from the include directory.

Syntax

`*include` *(filename)*

Type

HyperMesh Template Command

Inputs

filename

The file identified by must be in the include directory where the template is located.
`*include()` can be used to insert a series of template commands that are used by multiple templates.
`*include()` files can reference other `*include()` files in the same directory, but make sure you do not create an infinite `*include()` loop.

*includefiles()

Starts an include file output block. The include file references in the HyperMesh database are output according to the user-defined format in this block.

Syntax

`*includefiles(` *(solver_defined_flag1, solver_defined_flag2)*

Type

HyperMesh Template Command

Inputs

solver_defined_flag1, solver_defined_flag2

Both of these flags are defined based on the solver used. For example, flag1 for Nastran and OptiStruct represent the location of the include block. Either the include file is to be placed in the executive control section of the data file, and so on.

Example

To place an include in the executive controls section, flag 1 is 1:

```
*includefiles(1,0)
  *format()
    *string("INCLUDE '")
    *field(string,fullname,0)
    *string(')
    *end()
*output()
```

*interfacecomponents()

Starts an interfacecomponents block.

Syntax

`*interfacecomponents` *(config)*

Type

HyperMesh Template Command

Description

Starts an interfacecomponents block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all interfacecomponents with the format:

*interfacecomponents(id,"name")

```
*interfacecomponents()
  *format()
    *string("*interfacecomponents(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*interfacelinkings()

Starts an interfacelinkings block.

Syntax

`*interfacelinkings` *(config)*

Type

HyperMesh Template Command

Description

Starts an interfacelinkings block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all interfacelinkings with the format:

*interfacelinkings(id,"name")

```
*interfacelinkings()
  *format()
    *string("*interfacelinkings(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*joints()

Starts a joints block.

Syntax

`*joints` *(config)*

Type

HyperMesh Template Command

Description

Starts a joints block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
: An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all joints with the format:

*joint(id,"name")

```
*joint()
  *format()
    *string("*joint(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Errors

Incorrect usage results in an import error.

Version History

14.0.120

*laminates()

Starts a laminate block.

Syntax

`*laminates` *(card_image_name)*

Type

HyperMesh Template Command

Description

Starts a laminate block. Laminates with a card image matching the specified card image name are considered for the block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

card_image_name
: A 32-character string enclosed in double quotes that defines the card image name of the laminates. If specified as double quotes "", all laminates are considered for the block.

Example

To write out all laminates with the format:

*laminate(id,"name",color)

```
*laminates("")
*format()
*string("*laminate(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,color,0)
*string(")")
   *end()
*output()
```

Version History

11.0

*lines()

Starts a line output block. All of the lines in the database are output according to the user-defined format in the block following the `*lines()` command.

Syntax

`*lines` *()*

Type

HyperMesh Template Command

Example

Requires an `*output()` at the end of the block.

*lists()

Starts a lists block.

Syntax

`*lists` *(config)*

Type

HyperMesh Template Command

Description

Starts a lists block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all lists with the format:

*lists(id,"name")

```
*lists()
  *format()
    *string("*lists(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
```

```
*output()
```

Version History

2020

*loadcols()

Starts a load collector block.

Syntax

`*loadcols` *(card_image_name, ?idpool_name?)*

Type

HyperMesh Template Command

Description

Starts a load collector block. Load collectors with a card image matching the specified card image name are considered for the block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

card_image_name
: A 32-character string enclosed in double quotes that defines the card image name of the load collector. If specified as double quotes "", all load collectors are considered for the block.

idpool_name
: An optional 32-character string enclosed in double quotes that defines the name of the ID pool that the load collectors belongs to. If not needed, use double quotes "" or omit the argument. The ID pool must be defined using the `*defineidpool()` command.

Example

To write out all load collectors with the format:

*loadcollector(id,"name",color)

```
*loadcols("","")
*format()
*string("*loadcollector(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,color,0)
*string(")")
   *end()
*output()
```

*loads()

Starts a load output block. The loads in the HyperMesh database whose configuration is equal to the parameter configuration, and whose type is equal to the parameter type are output according to the user-defined format in this block.

Syntax

`*loads` *(config, type, user name)*

Type

HyperMesh Template Command

Inputs

config

Defines the configuration of the load that is output using this block definition and has the following possible values:

ConfigLoad	Output
0	Any load
1	Forces
2	Moments
3	Constraints
4	Pressures
5	Temperatures
6	Fluxes
8	Velocities
9	Accelerations

If the config supplied is 0 (zero), all loads of the given type are output.

type

Defines the type of load being defined. The *type* parameter allows users to define multiple types of loads per configuration. If the type supplied is 0 (zero), all loads of the given config are output.

user name

Defines the name of the load being defined.

Example

Requires an `*output()` at the end of the block.

*loadsteps()

Starts a loadsteps output block.

Syntax

`*loadsteps` *()*

Type

HyperMesh Template Command

Example

Each loadstep contains a list of IDs for the load collectors within that step.

Requires an `*output()` at the end of the block.

*loopif()

Conditionally executes a block of code while a condition is true.

Syntax

`*loopif` *(expression)*

Type

HyperMesh Template Command

Inputs

expression
: A relational expression.

Example

If *expression* evaluates to a nonzero value, then the statements contained within the loop block are executed. The example below shows the usage of the loop:

```
*counterset(counter1,1)
  *loopif([counter1 <= 5])
  *end()
  *counterinc(counter1)
  *endloop()
```

*markfailed()

Marks an element as failed when used in the Check Elements panel (user subpanel).

Syntax

`*markfailed` *()*

Type

HyperMesh Template Command

Description

To mark quads that have a side shorter than .1, the following commands may be used:

```
*elements(104,0,"","")
 *format()
  *if([shortestside < .1])
   *markfailed()
  *endif()
*output()
```

The command must only be used in a template file used with the user subpanel of the Check Elements panel.

It is used to mark an element that has failed a user-defined element check (the element will be highlighted). It can only be used within a `*elements` block. The element is also put in the user mark.

*masses()

Starts a mass entity block.

Syntax

`*masses` *(config)*

Type

HyperMesh Template Command

Description

Starts a mass entity block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all mass entities with the format:

*masses(id,"name")

```
*masses()
  *format()
    *string("*masses(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017.2

*materials()

Starts a material block.

Syntax

`*materials` *(card_image_name)*

Type

HyperMesh Template Command

Description

Starts a material block. Materials with a card image matching the specified card image name are considered for the block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

card_image_name

A 32-character string enclosed in double quotes that defines the card image name of the materials in the block. If specified as double quotes "", all materials are considered for the block. If a `*components()` or `*properties()` block has specified a *mat_card_image_name* value matching this string, the material collector to which those components or properties point is marked.

Example

To write out all materials with the format:

*material(id,"name",color)

```
*materials("","")
*format()
```

```
*string("*material(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,color,0)
*string(")")
   *end()
*output()
```

*mechanisms()

Starts a mechanisms block.

Syntax

`*mechanisms` *(config)*

Type

HyperMesh Template Command

Description

Starts a mechanisms block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out standard mechanisms and their associated objects:

```
*mechanisms(1)
  *format()
    *string("*MECHANISM")
    *end()
    *fieldright(integer,id,10)
    *fieldleft(string,name,80)
  *end()

  *bodies()
    *format()
      *string("*BODY")
      *end()
      *fieldright(integer,id,10)
      *fieldleft(string,name,80)
      *end()
  *output()

  *joints()
    *format()
```

```
      *string("*JOINT")
      *end()
      *fieldright(integer,id,10)
      *fieldleft(string,name,80)
      *end()
  *output()

  *constraints()
    *format()
      *string("*CONSTRAINT")
      *end()
      *fieldright(integer,id,10)
      *fieldleft(string,name,80)
      *end()
  *output()

  *positions(1)
    *format()
      *string("*POSITION")
      *end()
      *fieldright(integer,id,10)
      *fieldleft(string,name,80)
      *end()
  *output()
*output()
```

*meshcontrols()

Starts a meshcontrols block.

Syntax

`*meshcontrols` *(config)*

Type

HyperMesh Template Command

Description

Starts a meshcontrols block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all meshcontrols with the format:

*meshcontrols(id,"name")

```
*meshcontrols()
```

```
  *format()
    *string("*meshcontrols(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

14.0

*metadata()

Starts a metadata output block. All of the metadata entries in the database are output according to the user-defined format in the block following the `*metadata()` command.

Syntax

`*metadata` *()*

Type

HyperMesh Template Command

Example

Requires an `*output()` at the end of the output block.

*modelcheckchecks()

Starts a model check checks block.

Syntax

`*modelcheckchecks` *(config)*

Type

HyperMesh Template Command

Description

Starts a model check checks block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all model check checks with the format:

*modelcheckchecks(id,"name")

```
*modelcheckchecks()
  *format()
    *string("*modelcheckchecks(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*modelcheckcorrections()

Starts a model check corrections block.

Syntax

`*modelcheckcorrections` *(config)*

Type

HyperMesh Template Command

Description

Starts a model check corrections block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all model check corrections with the format:

*modelcheckcorrections(id,"name")

```
*modelcheckcorrections()
  *format()
    *string("*modelcheckcorrections(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
```

```
    *end()
*output()
```

Version History

2019

*modelMOI()

Computes the inertia tensor for the global model and store the results into a table.

Syntax

`*modelMOI` *(table)*

Type

HyperMesh Template Command

Description

Computes the inertia tensor for the global model and store the results into a table. This is valid only for LS-DYNA.

The different values are stored in this order in the table:

- totalIXX
- totalIYY
- totalIZZ
- totalIXY
- totalIXZ
- totalIYZ

This command must be called in the `*before()` section.

Inputs

table

A value between 1 and 20 indicating which of the 20 possible tables should be manipulated.

Example

To calculate the MOI values and store them in table 11:

```
*components("", "")
*before()
*tablenreset(11)
*modelMOI(11)
*format()
...
*after()
*string("Moment of Inertia for Model (Using Center of Gravity As Center):")
*end()
```

```
*string("                                                                  --
                          --")
*end()
*string("                                                                |  ")
*fieldright(real, [@nlookup(11,1)], 12)
*string(" ")
*fieldright(real, [@nlookup(11,4)], 12)
*string(" ")
*fieldright(real, [@nlookup(11,5)], 12)
*string("  |")
*end()
*string("                                                                |  ")
*fieldright(real, [@nlookup(11,4)], 12)
*string(" ")
*fieldright(real, [@nlookup(11,2)], 12)
*string(" ")
*fieldright(real, [@nlookup(11,6)], 12)
*string("  |")
*end()
*string("                                                                |  ")
*fieldright(real, [@nlookup(11,5)], 12)
*string(" ")
*fieldright(real, [@nlookup(11,6)], 12)
*string(" ")
*fieldright(real, [@nlookup(11,3)], 12)
*string("  |")
*end()
*string("                                                                  --
                          --")
*end()
```

Version History

14.0.110

*modules()

Starts a modules block.

Syntax

`*modules` *(config)*

Type

HyperMesh Template Command

Description

Starts a modules block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all modules with the format:

*modules(id,"name")

```
*modules()
  *format()
    *string("*modules(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

14.0

*nodes()

Starts a node output block. All of the nodes in the database are output according to the user-defined format in the block following the `*nodes()` command.

Syntax

`*nodes` *(configuration)*

Type

HyperMesh Template Command

Inputs

configuration

Defines the type of node to output in this block. Since HyperMesh only allows one type of node, this parameter is ignored but is allowed for future compatibility.

Example

The following commands are available to sort nodes within the template system:

```
*sortnodes(byinputsystem)
*sortnodes(byoutputsystem)
*sortnodes(byid)
*sortnodes(none)
```

The first three commands turn sorting on until it is turned off with `*sortnodes(none)`.

This command requires an `*output()` at the end of the block.

*objectives()

Starts an objectives output block.

Syntax

`*objectives` *()*

Type

HyperMesh Template Command

Example

Requires an `*output()` at the end of the block.

*optiresponses()

Starts an optiresponses output block.

Syntax

`*optiresponses` *()*

Type

HyperMesh Template Command

Example

Creates both DRESP1 and DRESP2 type responses. Requires an `*output()` at the end of the block.

*output()

Outputs the data defined in the preceding block.

Syntax

`*output` *()*

Type

HyperMesh Template Command

Example

Requires a preceding block definition.

*outputblocks()

Starts an outputblocks block.

Syntax

`*outputblocks` *()*

Type

HyperMesh Template Command

Example

`*outputblocks` contain a list of element or node IDs. An example is shown below:

```
*outputblocks()
 *format()
  *if([type == 1])
   *counterset(counter1,1)
   *loopif([counter1 <= idsmax])
    *pointerset(pointer1,ids,[counter1-1])
    *string("NOD: ")
    *field(integer,pointer1.pointervalue,8)
    *counterinc(counter1)
   *endloop()
  *endif()
  *if([type == 2])
   *counterset(counter1,1)
   *loopif([counter1 <= idsmax])
    *pointerset(pointer1,ids,[counter1-1])
    *string("ELE: ")
    *field(integer,pointer1.pointervalue,8)
    *counterinc(counter1)
   *endloop()
   *endif()
*output()
```

*outputparameterizeddata()

Specifies whether to output in parameterized or unparameterized format.

Syntax

`*outputparamaterizeddata` *(flag)*

Type

HyperMesh Template Command

Description

Specifies whether to output in parameterized or unparameterized format.

Inputs

flag

0 - Output in unparameterized format
1 - Output in parameterized format

Example

To output in parameterized format:

```
*outputparameterizeddata(1)
```

Version History

14.0

*outputrequests()

Starts an outputrequests block.

Syntax

`*outputrequests` *(config)*

Type

HyperMesh Template Command

Description

Starts an outputrequests block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all outputrequests with the format:

*outputrequests(id,"name")

```
*outputrequests()
  *format()
    *string("*outputrequests(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*panels()

Starts a panels block.

Syntax

`*panels` *(config)*

Type

HyperMesh Template Command

Description

Starts a panels block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all panels with the format:

*panels(id,"name")

```
*panels()
  *format()
    *string("*panels(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017.1

*parameters()

Starts a parameters block.

Syntax

`*parameters` *(card_image_name)*

Type

HyperMesh Template Command

Description

Starts a parameters block.

This command must be accompanied by an `*output()` command at the end of the block

Inputs

card_image_name

A 32-character string enclosed in double quotes that defines the card image name of the parameter. If specified as double quotes "", all parameters are considered for the block.

Example

To write out all parameters with the format:

*parameters(id,"name")

```
*parameters("")
  *format()
    *string("*parameters(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

14.0

*partsets()

Starts a partset block.

Syntax

`*partsets` *(config)*

Type

HyperMesh Template Command

Description

Starts a partset block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all partsets with the format:

*partsets(id,"name")

```
*partsets()
  *format()
    *string("*partsets(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017.1

*physicalquantites()

Starts a physicalquantity block.

Syntax

`*physicalquantites` *(config)*

Type

HyperMesh Template Command

Description

Starts a physicalquantity block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all physicalquantites with the format:

*physicalquantites(id,"name")

```
*physicalquantites()
  *format()
    *string("*physicalquantites(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2021

*plies()

Starts a ply block.

Syntax

`*plies` *(card_image_name)*

Type

HyperMesh Template Command

Description

Starts a ply block. Plies with a card image matching the specified card image name are considered for the block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

card_image_name
: A 32-character string enclosed in double quotes that defines the card image name of the plies. If specified as double quotes "", all plies are considered for the block.

Example

To write out all plies with the format:

*ply(id,"name",color)

```
*plies("")
*format()
*string("*ply(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,color,0)
*string(")")
```

```
   *end()
*output()
```

Version History

11.0

*plots()

Starts a plot output block. The plots in the HyperMesh database are output according to the user-defined format in this block.

Syntax

`*plots` *()*

Type

HyperMesh Template Command

Example

To output the IDs of the curves in a plot, the following commands should be used:

```
*counterset(counter1,0)
  *loop([counter1 != numberofcurves])
    *pointerset(pointer1,curves,counter1)
    *field(integer,pointer1.pointervalue,0)
    *counterinc(counter1)
*endloop()
```

Requires an `*output()` at the end of the block.

*pointerset()

Sets the initial value of a pointer.

Syntax

`*pointerset` *(pointer number, pointer, value)*

Type

HyperMesh Template Command

Inputs

pointer number

Value from pointer1 to pointer10 indicating which of the 10 possible pointers should be set to the*value* parameter.

pointer
The pointer to the data object to be accessed. Only certain data types may use pointers. These are described in the template commands in which they are valid.

value
The value of the pointer.

Example

For example, to output all the dictionary entries for a component, the following commands could be used while in the component block:

```
*counterset(counter1,0)
*loopif([counter1 != dictionarymax])
   *pointerset(pointer1,dictionary,counter1)
   *field(string,pointer1.name,0)
   *field(integer,pointer1.type,0)
   *field(string,pointer1.string,0)
   *field(real,pointer1.value,8)
   *counterinc(counter1)
*endloop()
```

*points()

Starts a points output block.

Syntax

`*points` *()*

Type

HyperMesh Template Command

Example

Requires an `*output()` command at the end of the block.

*populatemassthicknesstable()

Populates the element mass/thickness table.

Syntax

`*populatemassthicknesstable` *()*

Type

HyperMesh Template Command

Description

Populates the element mass/thickness table. This table is used to improve the performance of summary calculations. This function is only available for "core" solver templates routinely maintained by Altair, including Abaqus, ANSYS, LS-DYNA, MADYMO, Marc, Nastran, OptiStruct, PAM-CRASH 2G, Radioss, and Samcef. It cannot be used with custom templates.

This command must be inside either a `*components()` or `*elements()` block and should be called in the `*before()` section. It must be followed by a call to `*deletemassthicknesstable()`.

Example

To write out the mass of every component in the format

id,"name",mass

```
*components("","")
*before()
*populatemassthicknesstable()
*format()
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(real,mass,0)
*end()
*after()
*deletemassthicknesstable()
*output()
```

Version History

11.0

*positions()

Starts a positions block.

Syntax

`*positions` *(config)*

Type

HyperMesh Template Command

Description

Starts a positions block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

config
An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all positions with the format:

*positions(id,"name")

```
*positions()
  *format()
    *string("*positions(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

14.0

*pretensioners()

Starts a pretensioners block.

Syntax

`*pretensioners` *(config)*

Type

HyperMesh Template Command

Description

Starts a pretensioners block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all pretensioners with the format:

*pretensioners(id,"name")

```
*pretensioners()
  *format()
    *string("*pretensioners(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017.1

*properties()

Starts a property block.

Syntax

`*properties` *(card_image_name, mat_card_image_name,?idpool_name?)*

Type

HyperMesh Template Command

Description

Starts a property block. Properties with a card image matching the specified card image name are considered for the block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

card_image_name

A 32-character string enclosed in double quotes that defines the card image name of the property. If specified as double quotes "", all properties are considered for the block.
If a `*elements()` block has specified a *prop_card_image_name* value matching this string, the property collector to which those elements point is marked.

mat_card_image_name

A 32-character string enclosed in double quotes that defines the name of the material that the properties in the block require. If not needed, use double quotes "".
The name is also used to link to the `*materials()` command.

idpool_name

An optional 32-character string enclosed in double quotes that defines the name of the ID pool that the properties belongs to. If not needed, use double quotes "" or omit the argument.
The ID pool must be defined using the `*defineidpool()` command.

Example

To write out all properties with the format:

*property(id,"name",material ID,color)

```
*properties("","")
*format()
*string("*property(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,materialid,0)
*string(",")
*field(integer,color,0)
*string(")")
   *end()
*output()
```

*quote()

Writes a quotation (") character.

Syntax

`*quote` *()*

Type

HyperMesh Template Command

*rangeadd()

Add a number to a list so that ranges can be found with `@rangecount()`, `@rangestart()`, and `@rangeadd()`.

Syntax

`*rangeadd` *(number)*

Type

HyperMesh Template Command

Inputs

number
: The integer number to add to the list.

Example

```
*elements(104,0,"","")
  *format()
```

```
    *rangeadd(id)
  *after()
    *counterset(counter1,1)
    *loopif([counter1 <= @rangecount()])
      *string("start of range = ")
      *field(integer,[@rangestart(counter1)],0)
      *end()
      *field(integer,[@rangeend(counter1)],0)
      *end()
      *counterinc(counter1)
    *endloop()
    *rangereset()
*output()
```

Use `*rangeadd()` to add numbers to a list. Once all numbers have been added, `@rangecount()` returns the number of ranges (for example 1-5, 10-20) that are in the list. Use the functions `@rangestart()` and `@rangeend()` to get the actual ranges.

*rangereset()

Resets the list of numbers stored with `*rangeadd()`.

Syntax

`*rangereset` *()*

Type

HyperMesh Template Command

Example

```
*elements(104,0,"","")
  *format()
    *rangeadd(id)
  *after()
    *counterset(counter1,1)
    *loopif([counter1 <= @rangecount()])
      *string("start of range = ")
      *field(integer,[@rangestart(counter1)],0)
      *end()
      *string("end of range = ")
      *field(integer,[@rangeend(counter1)],0)
      *end()
      *counterinc(counter1)
    *endloop()
    *rangereset()
*output()
```

Use `*rangeadd()` to add numbers to a list. Once all numbers have been added, `@rangecount` returns the number of ranges (for example 1-5, 10-20) that are in the list. Use the functions `@rangestart()` and `@rangeend()` to get the actual ranges. `*rangereset()` removes all numbers from the list.

*realprecision()

Sets the number of significant figures after the decimal point for real numbers.

Syntax

`*realprecision` *(digits)*

Type

HyperMesh Template Command

Inputs

digits

The number of significant figures to be used for all real values after the command in the template file. A zero (default) uses all available spaces in the field width.

Example

This command results in values being rounded, according to IEEE specifications, to fit in the specified precision. Enabling the `*compressreal()` option truncates trailing zeros that are produced by this setting.

*regions()

Starts a regions block.

Syntax

`*regions` *(config)*

Type

HyperMesh Template Command

Description

Starts a regions block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all regions with the format:

*regions(id,"name")

```
*regions()
```

```
  *format()
    *string("*regions(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

14.0

*registerparameterizeddataappendstring()

Registers the string to be added before parameter names written in place of parameterized values.

Syntax

`*registerparametrizeddataappendstring` *("string")*

Type

HyperMesh Template Command

Description

Registers the string to be added before parameter names written in place of parameterized values.

Inputs

string
: The string to be added before the parameter names.

Example

In Abaqus, parameter names are written within angular brackets and are registered as:

```
*registerparametrizeddataappendstring("<")
*registerparametrizeddataendstring(">")
```

In LS-DYNA, parameter names are preceded by & and are registered as:

```
*registerparametrizeddataappendstring("&")
```

Version History

14.0

*registerparameterizeddataendstring()

Registers the string to be added after parameter names written in place of parameterized values.

Syntax

`*registerparametrizeddataendstring` *("string")*

Type

HyperMesh Template Command

Description

Registers the string to be added after parameter names written in place of parameterized values.

Inputs

string
The string to added after the parameter names.

Examples

In Abaqus, parameter names are written within angular brackets and are registered as:

```
*registerparametrizeddataappendstring("<")
*registerparametrizeddataendstring(">")
```

In LS-DYNA, parameter names are preceded by & and are registered as:

```
*registerparametrizeddataappendstring("&")
```

Version History

14.0

*registersolvercommentsyntaxstring()

Registers the string to be added in the beginning of the solver comment lines.

Syntax

`*registersolvercommentsyntaxstring` *("string")*

Type

HyperMesh Template Command

Description

Registers the string to be added in the beginning of the solver comment lines.

Inputs

string
The string to be added in the beginning of the solver comment lines.

Example

To register the solver comment lines to start with $:

```
*registerparametrizeddataappendstring("$")
```

Version History

2020

*responses()

Starts a responses block.

Syntax

`*responses` *(config)*

Type

HyperMesh Template Command

Description

Starts a responses block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all responses with the format:

*responses(id,"name")

```
*responses()
  *format()
    *string("*responses(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*results()

Starts a results block.

Syntax

`*results` *(config)*

Type

HyperMesh Template Command

Description

Starts a results block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all results with the format:

*results(id,"name")

```
*results()
  *format()
    *string("*results(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*retractors()

Starts a retractors block.

Syntax

`*retractors` *(config)*

Type

HyperMesh Template Command

Description

Starts a retractors block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all retractors with the format:

*retractors(id,"name")

```
*retractors()
  *format()
    *string("*retractors(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017.1

*return()

Ends a function block.

Syntax

`*return` *()*

Type

HyperMesh Template Command

*rigidbodies()

Starts a rigidbodies block.

Syntax

`*rigidbodies` *(config)*

Type

HyperMesh Template Command

Description

Starts a rigidbodies block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all rigidbodies with the format:

*rigidbodies(id,"name")

```
*rigidbodies()
  *format()
    *string("*rigidbodies(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017.2

*rigidwalls()

Starts a rigid wall block.

Syntax

`*rigidwalls` *(config)*

Type

HyperMesh Template Command

Description

Starts a rigid wall block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
: An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all rigid walls with the format:

*rigidwalls(id,"name")

```
*rigidwalls()
  *format()
    *string("*rigidwalls(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017

*scalefieldwidth()

Scales the width of a field.

Syntax

`*scalefieldwidth` *(field type, scalefactor)*

Type

HyperMesh Template Command

Inputs

field type
: Specifies which field type to scale. Valid values are "integer", "real", and "string".

scalefactor
: The scale factor to apply to each field of *field type*. Typically, this value is 1 or 2.

Example

To print the node IDs and globalx values with a width of 16 instead of 8:

```
*nodes()
  *before()
    *variableset(variable1,2)
      *scalefieldwidth(real,variable1)
      *scalefieldwidth(integer,variable1)

  *format()
    *string("*")
    *field(integer,id,8)
    *string("*")
    *end()

    *string("*")
    *fieldleft(real,globalx,8)
    *string("*")
    *end()

*output()
```

This command is typically used to write a deck that contains double precision numbers which have twice the field width specified by the `*field` command.

*seatbeltcontrolpoints()

Starts a seatbeltcontrolpoint block.

Syntax

`*seatbeltcontrolpoints` *(config)*

Type

HyperMesh Template Command

Description

Starts a seatbeltcontrolpoint block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all seatbeltcontrolpoints with the format:

*seatbeltcontrolpoints(id,"name")

```
*seatbeltcontrolpoints()
```

```
  *format()
    *string("*seatbeltcontrolpoints(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*seatbelts()

Starts a seatbelt block.

Syntax

`*seatbelts` *(config)*

Type

HyperMesh Template Command

Description

Starts a seatbelt block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all seatbelts with the format:

*seatbelts(id,"name")

```
*seatbelts()
  *format()
    *string("*seatbelts(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*segments()

Starts a segment output block.

Syntax

`*segments` *()*

Type

HyperMesh Template Command

Example

Requires an *endsegments() command.

*sensors()

Starts a sensors block.

Syntax

`*sensors` *(card_image_name, ?idpool_name?)*

Type

HyperMesh Template Command

Description

Starts a sensors block. Sensors with a card image matching the specified card image name are considered for the block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

card_image_name
: A 32-character string enclosed in double quotes that defines the card image name of the sensor. If specified as double quotes "", all sensors are considered for the block.

idpool_name
: An optional 32-character string enclosed in double quotes that defines the name of the ID pool that the sensor belongs to. If not needed, use double quotes "" or omit the argument.
The ID pool must be defined using the `*defineidpool()` command

Example

To write out all sensors with the format:

*sensor(id,"name")

```
*sensors("","")
*format()
*string("*sensor(")
```

```
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(")")
   *end()
*output()
```

*sequences()

Starts a sequences block.

Syntax

`*sequences` *(config)*

Type

HyperMesh Template Command

Description

Starts a sequences block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all sequences with the format:

*sequences(id,"name")

```
*sequences()
  *format()
    *string("*sequences(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020.1

*setcollector()

Sets the current collector based on the last `*pointerset()` command. This is commonly used with loadsteps. You can also use this command with sets that contain components.

Syntax

`*setcollector` *()*

Type

HyperMesh Template Command

Example

The loadsteps entity contains the ids of loadcollectors for that step. To output the loads within each loadcollector:

```
*loadsteps()
  *format()
    *string("SUBCASE")
    *field(integer,id,3)
    *end()
    *counterset(counter1,0)
    *loopif([counter1 < idsmax ])
       *pointerset(pointer1,ids,counter1)
       *setcollector()
       *loads(3,1,"SPC")
         *before()
            *counterset(counter2,0)
         *format()
           *if([counter2 == 0])
             *string("SPC")
             *field(integer,pointer1.pointervalue,8)
             *end()
            *endif()
            *counterinc(counter2)
      *output()
      *counterinc(counter1)
   *endloop()
*output()
```

*setcurrentbagentitytype()

Sets the type of entity within a bag for which information is to be queried.

Syntax

`*setcurrentbagentitytype` *(entity_type)*

Type

HyperMesh Template Command

Description

This command is used in the `*format` level of a template to set the queried entity type of the bag to *entity_type*. Subsequent references to bag entity data names entitylist and entitylistmax in the template will then be based on the entity type that was set using this command, until `*setcurrentbagentitytype` is called again with a different *entity_type*.

Example

The following code in the HMASCII template loops through all the entity types in the database for each bag and writes out the ids of the entities, if present:

```
*bags()
  *before()
    *end()
    *string("BEGIN BAGS")
    *end()
  *format()
    *string("*bags(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(",")
    *field(integer,config,0)
    *string(",")
    *field(integer,color,0)
    *string(")")
    *end()
    *if([attributesmax > 0])
       *string("  *attributesforentity(BAGS,")
        *field(integer,id,0)
        *string(",")
        *field(integer,attributesmax,0)
        *string(")")
        *end()
        *include(attribs)
    *endif()
    *counterset(counter1,1)
    *loopif([counter1 <= globalentitytypesmax])
      *setcurrentbagentitytype([@getdatabaseentitytypename(counter1)])
      *if([entitylistmax > 0])
        *end()
        *string("*bagentitytype(")
        *field(string,[@getdatabaseentitytypename(counter1)],0)
        *string(")")
        *end()
        *string("*bagentitylistmax(")
        *field(integer,entitylistmax,0)
        *string(")")
        *end()
        *counterset(counter2,0)
        *string("*bagentityidlist(")
        *loopif([counter2 < entitylistmax])
          *pointerset(pointer1,entitylist,counter2)
          *field(integer,pointer1.pointervalue,0)
          *if([counter2 < (entitylistmax - 1)])
            *string(",")
          *endif()
          *counterinc(counter2)
          *if([counter2 == entitylistmax])
            *string(")")
```

```
            *end()
          *else()
            *if([counter2 % 9 == 0])
              *end()
            *endif()
          *endif()
        *endloop()
        *end()
      *endif()
      *counterinc(counter1)
    *endloop()
    *end()
  *after()
    *string("END BAGS")
    *end()
*output()
```

*setettypeelemtypereference()

Sets the element configuration corresponding to the ET type specified in the sensor card image.

Syntax

`*setettypeelemtypereference` *(sensor_card_image_name, config)*

Type

HyperMesh Template Command

Description

Sets the element configuration corresponding to the ET type specified in the sensor card image. This is applicable only for ANSYS.

Inputs

sensor_card_image_name
: A 32-character string enclosed in double quotes that indicates the card image name of the sensor for which the element configuration applies.

config
: The configuration number of the elements in the sensor card image.

Example

```
*setettypeelemtypereference("SHELL51",60)
*setettypeelemtypereference("CONTAC52",70)
*setettypeelemtypereference("PLANE53",106)
*setettypeelemtypereference("PLANE53",108)
*setettypeelemtypereference("SOLID46",208)
```

Version History

14.0

*setformattype()

Sets the export format type for the LS-DYNA profile.

Syntax

`*setformattype` *(?type?)*

Type

HyperMesh Template Command

Description

Sets the export format type for the LS-DYNA profile.

Inputs

type

The export format type. Valid values are:
1 - Standard format
2 - Long format
3 - I10 format
When no type is passed, the export will be based on the import deck. For instance, when importing a long format deck, by default the export will happen in long format.

Example

To set the export format to long:

```
*setformattype(2)
```

Version History

2020

*setidmanagerexcludedidpools()

Defines list of ID pools not to be supported by ID Manager.

Syntax

`*setidmanagerexcludedidpools` *(pool_id1, ?pool_id2?, ... , ?pool_idN?)*

Type

HyperMesh Template Command

Description

Defines list of ID pools not to be supported by ID Manager.

Inputs

pool_id
The IDs of the pools that are not to be supported.

Examples

To define pool number 16 and 56 to not be supported by ID Manager:

```
*setidmanagerexcludedidpools(16, 56)
```

Version History

2019

*setidmanagersupportedentitytypes()

Defines the list of entity types to be supported by ID Manager.

Syntax

`*setidmanagersupportedentitytypes` *(entity_type_id1, ?entity_type_id2?, ..., ?entity_type_idN?)*

Type

HyperMesh Template Command

Description

Defines the list of entity types to be supported by ID Manager.

Inputs

entity_type_id
The entity type IDs that are to be supported.

Examples

To define nodes (1), elems (2) and comps (3) to be supported by ID Manager:

```
*setidmanagersupportedentitytypes(1, 2, 3)
```

Version History

2019

*sets()

Starts a set block.

Syntax

`*sets` *(?card_image_name?, ?idpool_name?,?subentity_idpool_name?)*

Type

HyperMesh Template Command

Description

Starts a set block. Sets with a card image matching the specified card image name are considered for the block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

card_image_name
: An optional 32-character string enclosed in double quotes that defines the card image name of the sets in the block. If not needed, use double quotes "" or omit the argument.

idpool_name
: An optional 32-character string enclosed in double quotes that defines the name of the ID pool that the sets belong to. If not needed, use double quotes "" or omit the argument.
The ID pool must be defined using the `*defineidpool()` command.

subentity_idpool_name
: An optional 32-character string enclosed in double quotes that defines the name of the ID pool that the entities in the set must belong to. If not needed, use double quotes "" or omit the argument.
The ID pool must be defined using the `*defineidpool()` command.

Example

To write out all sets with the format:

*set(id,"name",color,"typename",ordered)

*setid(entity 1 ID)

*setid(entity 2 ID)

...

```
*sets()
*format()
*string("*set(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,color,0)
*string(",")
*field(quotedstring,typename,0)
*string(",")
*field(integer,ordered,0)
*string(")")
*end()

*counterset(counter1,0)
*loopif([counter1 != idsmax])
*pointerset(pointer1,ids,counter1)
*string("*setid(")
```

```
*field(integer,pointer1.pointervalue,0)
*string(")")
*end()
*counterinc(counter1)
*endloop()
*output()
```

*setsolverusessegmentsets()

Adds support for segment sets to a template.

Syntax

`*setsolverusessegmentsets` *(flag)*

Type

HyperMesh Template Command

Description

Adds support for segment sets to a template.

Currently supported for OptiStruct, Radioss, Abaqus, EXODUS and Nastran.

Inputs

flag

0 - Not supported
1 - Supported

Example

To enable support:

```
*setsolverusessegmentsets(1)
```

Version History

2020

*shape3ds()

Starts a shape3d block.

Syntax

`*shape3ds` *(config)*

Type

HyperMesh Template Command

Description

Starts a shape3d block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all shape3ds with the format:

*shape3ds(id,"name")

```
*shape3ds()
  *format()
    *string("*shape3ds(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019.1

*shapes()

Starts a shape output block. All the shapes are output according to the user-defined format in this block.

Syntax

`*shapes` *()*

Type

HyperMesh Template Command

Example

Requires an `*output()` command at the end of the output block definition.

*sliprings()

Starts a sliprings block.

Syntax

`*sliprings` *(config)*

Type

HyperMesh Template Command

Description

Starts a sliprings block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

config
: An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all sliprings with the format:

*sliprings(id,"name")

```
*sliprings()
  *format()
    *string("*sliprings(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2017.1

*solvermasses()

Starts a solvermasses block.

Syntax

`*solvermasses` *(config)*

Type

HyperMesh Template Command

Description

Starts a solvermasses block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all solvermasses with the format:

*solvermasses(id,"name")

```
*solvermasses()
  *format()
    *string("*solvermasses(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*solversubmodels()

Starts a solversubmodels block.

Syntax

`*solversubmodels` *(config)*

Type

HyperMesh Template Command

Description

Starts a solversubmodels block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all solversubmodels with the format:

*solversubmodels(id,"name")

```
*solversubmodels()
  *format()
    *string("*solversubmodels(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*sortelements()

Sets the order that elements are output in an `*elements()` block.

Syntax

`*sortelements` *(order)*

Type

HyperMesh Template Command

Description

Sets the order that elements are output in an `*elements()` block.

This command must be inside a standalone `*elements()` block (that is a block not inside another block like `*components()`) and should be called in the `*before()` section.

Inputs

order

Valid values are:
bycomponentid - Ordering is performed based on component ID.
byid - Ordering is performed by ID.
bymaterialid - Ordering is performed based on material ID.
bypropertyid - Ordering is performed based on property ID.
none - No ordering is performed.

Example

To output elements in order based on material ID:

```
*elements(0,0,"","")
  *before()
    *variableset(variable1,0)
```

```
    *sortelements(bymaterialid)
  *format()
    *if([variable1 != collector.materialid])
      *string("Elements in material")
      *field(integer,collector.materialid,0)
      *end()
      *variableset(variable1,collector.materialid)
    *endif()
    *string("id = ")
    *field(integer,id,0)
    *end()
*output()
```

To output elements in order based on ID:

```
*elements(0,0,"","")
  *before()
    *sortelements(byid)
  *format()
    *string("id = ")
    *field(integer,id,0)
    *end()
*output()
```

*sortentity()

Changes the order that entities are output in the corresponding `*entity()` block.

Syntax

`*sortentity` *(entity_type, order)*

Type

HyperMesh Template Command

Description

Changes the order that entities are output in the corresponding `*entity()` block.

Currently only supported for the

Inputs

entity_type
: The type of entity to sort.

order
: Valid values are:
byid - Ordering is performed based on ID
none - No ordering is performed

Example

To output parameters based on their ID:

```
*sortentity(parameters,byid)
*parameters()
  *format()
    *string("*parameter(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*sortloads()

Changes the order that loads are output in the `*loads()` block.

Syntax

`*sortloads` *(order)*

Type

HyperMesh Template Command

Inputs

order

Valid values are:
bycomp1 - Ordering is performed based on component 1.
bycomp2 - Ordering is performed based on component 2.
bycomp3 - Ordering is performed based on component 3.
bycomp4 - Ordering is performed based on component 4.
bycomp5 - Ordering is performed based on component 5.
bycomp6 - Ordering is performed based on component 6.
bycomps - Ordering is performed by all components.
none - No ordering is performed.

Example

The following example outputs all forces in the same x direction, followed by the same y direction and then the same z direction:

```
*loads(1,0,"")
  *sortloads(bycomp1)
  *before()
    *variableset(variable1,-999999)
  *format()
    *if([comp1 != variable1])
```

```
      *end() *end()
      *string("FORCE x = ") *field(real,comp1,0) *end()
      *variableset(variable1,comp1)
    *endif()
    *string(" ") *field(integer,id,0) *string(" ")
  *after()
    *end()
*output()

*loads(1,0,"")
  *sortloads(bycomp2)
  *before()
    *variableset(variable1,-999999)
  *format()
    *if([comp2 != variable1])
      *end() *end()
      *string("FORCE y = ") *field(real,comp2,0) *end()
      *variableset(variable1,comp2)
    *endif()
    *string(" ") *field(integer,id,0) *string(" ")
  *after()
    *end()
*output()

*loads(1,0,"")
  *sortloads(bycomp3)
  *before()
    *variableset(variable1,-999999)
  *format()
    *if([comp3 != variable1])
      *end() *end()
      *string("FORCE z = ") *field(real,comp3,0) *end()
      *variableset(variable1,comp3)
    *endif()
    *string(" ") *field(integer,id,0) *string(" ")
  *after()
    *end() *end()
*output()
```

*sortloadsteps()

Changes the order that loadsteps are output in a `*loadsteps()` block.

Syntax

`*sortloadsteps` *(sort type)*

Type

HyperMesh Template Command

Inputs

sort type

byid	sort loadsteps by ID

none sorting is not performed

Example

The following example outputs all loadsteps by ID:

```
*loadsteps(1,0,"")
  *before()
    *sortloadsteps(byid)
  *format()
    *string("Loadstep ID")
    *field(real,id,0)
    *end()
*output()
```

*sortnodes()

Sets the order that nodes are output in a `*nodes()` block.

Syntax

`*sortnodes` *(order)*

Type

HyperMesh Template Command

Description

Sets the order that nodes are output in a `*nodes()` block.

This command must be inside a standalone `*nodes()` block (that is a block not inside another block like `*components()`) and should be called in the `*before()` section.

Inputs

order

Valid values are:
byid - Ordering is performed by ID.
byinputsystem - Ordering is performed based on input system ID.
byoutputsystem - Ordering is performed based on output system ID.
bysurfaceid - Ordering is performed based on surface ID.
none - No ordering is performed.

Example

To output nodes in order based on their ID:

```
*nodes()
  *before()
    *sortnodes(byid)
  *format()
```

```
    *string("id = ")
    *field(integer,id,0)
    *end()
*output()
```

*sortsets()

Sets the order that sets are output in a `*sets()` block.

Syntax

`*sortsets` *(order)*

Type

HyperMesh Template Command

Description

Sets the order that sets are output in a `*sets()` block.

Inputs

order

Valid values are:
byid - Ordering is performed by ID.
none - No ordering is performed.

Example

To output elements in order based on ID:

```
*sets()
  *sortsets(byid)
  *format()
    *string("id = ")
    *field(integer,id,0)
    *end()
*output()
```

Version History

14.0

*specialidruletoignoreincomingids()

Sets an ID pool to ignore incoming IDs for when there is an ID range defined.

Syntax

`*specialidruletoignoreincomingids` *(entity_type, pool_id)*

Type

HyperMesh Template Command

Description

Sets an ID pool to ignore incoming IDs for when there is an ID range defined.

This is applicable for entities without and ID definition in the deck.

Inputs

entity_type
: The type of entity to consider.

pool_id
: The solver ID pool number to use.

Example

To ignore the incoming element IDs on the CONSTRAINED_NODAL_RIGIDBODY_IDPOOL ID pool (13) in the LS-DYNA user profile:

```
*specialidruletoignoreincomingids(elements, 13)
```

Version History

2017

*string()

Outputs a string to the output file.

Syntax

`*string` *(string)*

Type

HyperMesh Template Command

Inputs

string
: A string of characters. If the string contains a space, an asterisk, or a comma, the string must be enclosed by double quotes.

*stringtablereset()

Resets the string lookup table.

Syntax

`*stringtablereset` *()*

Type

HyperMesh Template Command

*stringtablestore()

Stores an entry into the string lookup table.

Syntax

`*stringtablestore` *(key, value)*

Type

HyperMesh Template Command

Inputs

key

Assigned to a string lookup table entry and is used by the `@stringlookup()` function to find stored entries. *key* can be a data name or a literal string enclosed in double quotes.

value

The value assigned to an entry in the string lookup table and is returned by `@stringlookup()`.

Example

The example below stores the string shells in the string lookup table with a value of 10:

```
*stringtablestore("shells",10)
```

The example below saves component names with a value of 1 in the string table:

```
*components("","")
*format()
*stringtablestore(name,1)
*output()
```

*stringwithcomments()

Outputs a string field name and string to the output file. The field name is written just above the string line.

Syntax

`*stringwithcomments` *(field_name, string)*

Type

HyperMesh Template Command

Description

Outputs a string field name and string to the output file. The field name is written just above the string line.

Inputs

field_name
: A string of characters. If the string contains a space, an asterisk, or a comma, the string must be enclosed by double quotes. If the string name length is less than the width, spaces are appended after the string. If the string name length is more than the width, the name is truncated such that the length is equal to width.

string
: A string of characters. If the string contains a space, an asterisk, or a comma, the string must be enclosed by double quotes.

Example

To write the string field "My field" with the string "My string":

```
*stringwithcomments("My field","My string")
```

Version History

2020

*structuralproperties()

Starts a structuralproperty block.

Syntax

`*structuralproperties` *(config)*

Type

HyperMesh Template Command

Description

Starts a structuralproperty block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all structuralproperties with the format:

*structuralproperties(id,"name")

```
*structuralproperties()
  *format()
    *string("*structuralproperties(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019.1

*studies()

Starts a studies block.

Syntax

`*studies` *(config)*

Type

HyperMesh Template Command

Description

Starts a studies block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all studies with the format:

*studies(id,"name")

```
*studies()
  *format()
    *string("*studies(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*subsystemconfigurations()

Starts a subsystemconfigurations block.

Syntax

`*subsystemconfigurations` *(config)*

Type

HyperMesh Template Command

Description

Starts a subsystemconfigurations block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all subsystemconfigurations with the format:

*subsystemconfigurations(id,"name")

```
*subsystemconfigurations()
  *format()
    *string("*subsystemconfigurations(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
```

```
    *end()
*output()
```

Version History

2020

*subsystems()

Starts a subsystems block.

Syntax

`*subsystems` *(config)*

Type

HyperMesh Template Command

Description

Starts a subsystems block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all subsystems with the format:

*subsystems(id,"name")

```
*subsystems()
  *format()
    *string("*subsystems(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*subsystemsets()

Starts a subsystemsets block.

Syntax

`*subsystemsets` *(config)*

Type

HyperMesh Template Command

Description

Starts a subsystemsets block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all subsystemsets with the format:

*subsystemsets(id,"name")

```
*subsystemsets()
  *format()
    *string("*subsystemsets(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*surfaces()

Starts a surface output block.

Syntax

`*surfaces` *()*

Type

HyperMesh Template Command

Example

Requires an `*output()` command at the end of the block.

*symmetrypivots()

Starts a symmetrypivot block.

Syntax

`*symmetrypivots` *(config)*

Type

HyperMesh Template Command

Description

Starts a symmetrypivot block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all symmetrypivots with the format:

*symmetrypivots(id,"name")

```
*symmetrypivots()
  *format()
    *string("*symmetrypivots(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019.1

*systcols()

Starts a system collector output block.

Syntax

`*systcols` *(name)*

Type

HyperMesh Template Command

Inputs

name
The card image name with which to filter the scanned system collectors.

Example

Requires an `*output()` at the end of the block.

*systems()

Starts a systems block.

Syntax

`*systems` *()*

Type

HyperMesh Template Command

Description

Starts a systems block.

This command must be accompanied by an `*output()` command at the end of the block.

Example

To write out all systems with the format:

*system(id,type)

```
*systems()
*format()
*string("*system(")
*field(integer,id,0)
*string(",")
*field(integer,type,0)
*string(")")
*end()
*output()
```

*tablenreset()

Resets the nth lookup table.

Syntax

`*tablenreset` *(table)*

Type

HyperMesh Template Command

Inputs

table
: A value between table1 and table20 indicating which of the 20 possible tables should be manipulated.

*tablenstore()

Stores an entry in the nth lookup table.

Syntax

`*tablenstore` *(table, key, value)*

Type

HyperMesh Template Command

Inputs

table
: A value between table1 and table20 indicating which of the 20 possible tables should be manipulated.

key
: Assigned to a lookup table entry and is used by the `@nlookup()` function to find stored entities.

value
: The value assigned to an entry in the lookup table and is returned by @nlookup().

*tablereset()

Resets the lookup table.

Syntax

`*tablereset` *()*

Type

HyperMesh Template Command

*tables()

Starts a tables block.

Syntax

`*tables` *(card_image_name, ?idpool_name?)*

Type

HyperMesh Template Command

Description

Starts a tables block. Tables with a card image matching the specified card image name are considered for the block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

card_image_name
: A 32-character string enclosed in double quotes that defines the card image name of the table. If specified as double quotes "", all tables are considered for the block.

idpool_name
: An optional 32-character string enclosed in double quotes that defines the name of the ID pool that the table belongs to. If not needed, use double quotes "" or omit the argument.
The ID pool must be defined using the `*defineidpool()` command.

Example

To write out all tables with the format:

*table(id,"name",rows,columns)

```
*tables("","")
*format()
*string("*table(")
*field(integer,id,0)
*string(",")
*field(quotedstring,name,0)
*string(",")
*field(integer,rows,0)
*string(",")
*field(integer,columns,0)
*string(")")
*end()
*output()
```

Version History

11.0

*tablestore()

Stores an entry in the lookup table.

Syntax

`*tablestore` *(key, value)*

Type

HyperMesh Template Command

Inputs

key
: Assigned to a lookup table entry and is used by the `@lookup()` function to find stored entries.

value
: The value assigned to an entry in the lookup table and is returned by `@lookup()`.

*terminations()

Starts a termination block.

Syntax

`*terminations` *(config)*

Type

HyperMesh Template Command

Description

Starts a termination block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config
: An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all terminations with the format:

*terminations(id,"name")

```
*terminations()
  *format()
    *string("*terminations(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2019

*text()

Starts a block that contains text.

Syntax

`*text` *()*

Type

HyperMesh Template Command

Example

Requires an `*output` command at the end of the block.

This block provides an easy method of outputting a series of strings.

*timestepcontrols()

Starts a timestepcontrol block.

Syntax

`*timestepcontrols` *(config)*

Type

HyperMesh Template Command

Description

Starts a timestepcontrol block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all timestepcontrols with the format:

*timestepcontrols(id,"name")

```
*timestepcontrols()
  *format()
    *string("*timestepcontrols(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2021

*titles()

Starts a title output block.

Syntax

`*titles` *()*

Type

HyperMesh Template Command

Example

Requires an `*output` at the end of the output block definition.

*transformations()

Starts a transformations block.

Syntax

`*transformations` *(config)*

Type

HyperMesh Template Command

Description

Starts a transformations block.

This command must be accompanied by an `*output()` command at the end of the block.

Inputs

config
: An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all transformations with the format:

*transformations(id,"name")

```
*transformations()
  *format()
    *string("*transformations(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

14.0

*uservariableset()

Sets a user variable to a specific value.

Syntax

`*uservariableset` *(variable, value)*

Type

HyperMesh Template Command

Inputs

variable
: A user-defined variable. The name must begin with ' # '.

value
: The value of the variable.

Example

```
*uservariableset("#YOUNG_MODUL_E", 0)
```

*variableset()

Sets a variable to a specific value.

Syntax

`*variableset` *(variable, value)*

Type

HyperMesh Template Command

Inputs

variable
: A value from variable1 to variable20 indicating which of the 20 possible variables should be set to the next parameter.

value
: The value of the variable.

Example

Variables can be used to hold real or integer values. For example, to add the value of variable5 to the current value of variable1, the following command could be used:

```
*variableset(variable1,[variable1+variable5])
```

*vectorcols()

Starts a vector collector output block.

Syntax

`*vectorcols` *(name)*

Type

HyperMesh Template Command

Inputs

name
: Vector collectors with this card image name will be output.
Used as a key to distinguish different types of vector collectors.

Example

Requires an `*output` command at the end of the block.

*vectors()

Starts a vectors block.

Syntax

`*vectors` *()*

Type

HyperMesh Template Command

Description

Starts a vectors block.

This command must be accompanied by an `*output()` command at the end of the block.

Example

To write out all vectors with the format:

*vectorentity(id,systemid,basenodeid,farnodeid,xcomp,ycomp,zcomp,magnitude)

```
*vectors()
*format()
*string("  *vectorentity(")
*field(integer,id,0)
*string(",")
*field(integer,systemid,0)
*string(",")
*field(integer,basenodeid,0)
*string(",")
*if([farnodeid > 0])
*field(integer,farnodeid,0)
*string(", 0.0, 0.0, 0.0, 0.0")
*else()
*field(integer,0,0)
*string(",")
*field(real,xcomp,8)
*string(",")
*field(real,ycomp,8)
*string(",")
*field(real,zcomp,8)
*string(",")
*field(real,magnitude,8)
*endif()
*output()
```

*vectortablereset()

Resets the vector lookup table.

Syntax

`*vectortablereset` *(type, tolerance)*

Type

HyperMesh Template Command

Inputs

type

The type of the vector lookup table. This determines how the vectors in the lookup table are compared against the key values sent by the `@vectorlookup()` function. Set this to 0, if the vectors represent node locations in space and the distance between the two nodes is the criteria for vectors being equal. Set this to 1, if the vectors represent vectors and the angle between the two vectors is the criteria for vectors being equal.

tolerance

The tolerance used to determine if a vector in the lookup table is equal to a key vector.

*vectortablestore()

Stores an entry into the vector lookup table.

Syntax

`*vectortablestore` *(key, x comp, y comp, z comp, value)*

Type

HyperMesh Template Command

Inputs

key

The key assigned to the lookup table entry. This is used by the `@vectorlookup()` function to find stored entries.

x comp

The x component of the vector assigned to a lookup table entry. This is used by the `@vectorlookup()` function to find stored entries.

y comp

The y component of the vector assigned to a lookup table entry. This is used by the `@vectorlookup()` function to find stored entries.

z comp

The z component of the vector assigned to a lookup table entry. This is used by the `@vectorlookup()` function to find stored entries.

value

The value assigned to an entry in the lookup table. This value is returned by `@vectorlookup()`.

*weldlines()

Starts a weldline block.

Syntax

`*weldlines` *(config)*

Type

HyperMesh Template Command

Description

Starts a weldline block.

This command must be accompanied by a `*output()` command at the end of the block.

Inputs

config

An integer that defines the config of entities in the block. If not specified, all entities are considered for the block.

Example

To write out all weldlines with the format:

*weldlines(id,"name")

```
*weldlines()
  *format()
    *string("*weldlines(")
    *field(integer,id,0)
    *string(",")
    *field(quotedstring,name,0)
    *string(")")
    *end()
*output()
```

Version History

2020

*writegeometry()

Outputs the Altair geometry database in an internal ASCII format.

Syntax

`*writegeometry` *(string)*

Type

HyperMesh Template Command

Inputs

string
A small string to be inserted at the beginning of each line to serve as a comment character.

Example

This format is designed to be read in with the hminlib function `HM_writegeomdata()`.

*writenamedbuffer()

Copies the content from an include file to the current output file.

Syntax

`*writenamedbuffer` *("name")*

Type

HyperMesh Template Command

Description

Copies the content from an include file to the current output file.

Inputs

name
The short name of an include file whose content needs to be copied to current output file.

Examples

To output the contents of include "front" to the current output file:

```
*writenamedbuffer("front")
```

Version History

2019

Deprecated Solver Template Commands

The list of deprecated solver template commands, and the new commands to use.

Deprecated Commands	New Commands to Use
*cubiclines()	
*optitableentrs()	*tables()

Undocumented Solver Template Commands

The list of undocumented solver template commands.

*assigndictionarytogroup()

*attribute()

*beamsect_attribid()

*collisions()

*connectors()

*ddvals()

*defineidpool()

*dobjrefs()

*domains()

*elementorient2d()

*ellipsoids()

*getcardthicknessproc()

*handles()

*includeoffsetvalues()

*mbjoints()

*mbplanes()

*multibodies()

*opticonstraints()

*opticontrols()

*optidscreens()

*registerparametrizeddataappendstring()

*selectbyidproc()

*setcardthicknessproc()

*setelementcolorbymatsmethod()

*setelementcolorbypropsmethod()

*setoffsetinfo()

*setpropertyelementreference()

*sortdictionarynames()

*summaryrowcol()

*symmetrys()

*tags()

*treataslocal()

*writeconnectors()

1.3.3 Solver Template Functions

@acos()

Trigonometric arc cosine of x, the result is expressed in radians between 0 and pp.

Syntax

`@acos` *(x)*

Type

HyperMesh Template Function

Inputs

x
: Value of type real.

@activateNlgeomImpdyn()

Returns 1 if the environment variable HM_ACTIVATE_NLGEOM_IMPDYN is set to YES.

Syntax

`@activateNlgeomImpdyn()`

Type

HyperMesh Template Function

Description

Returns 1 if the environment variable HM_ACTIVATE_NLGEOM_IMPDYN is set to YES.

Examples

Remove NLGEOM and IMPDYN from ANLYSIS TYPE:

```
*menuif([@activateNlgeomImpdyn()])
    *menufield(TYPE,string,$NAST_ANALYSIS_TYPE,8)
    *menulegalvalue(STATICS)
    *menulegalvalue(MODES)
    *menulegalvalue(BUCK)
    *menulegalvalue(DFREQ)
    *menulegalvalue(MFREQ)
    *menulegalvalue(MBD)
    *menulegalvalue(DTRAN)
    *menulegalvalue(MTRAN)
```

```
    *menulegalvalue(DFOUR)
    *menulegalvalue(MFOUR)
    *menulegalvalue(MCEIG)
    *menulegalvalue(HEAT)
    *menulegalvalue(FATIGUE)
    *menulegalvalue(NLGEOM)
    *menulegalvalue(NLSTAT)
    *menulegalvalue(RSPEC)
    *menulegalvalue(EXPDYN)
    *menulegalvalue(IMPDYN)
    *menulegalvalue(SUBCOM)
    *menulegalvalue(NLHEAT)
    *menulegalvalue(RANDOM)
*menuelse()
    *menufield(TYPE,string,$NAST_ANALYSIS_TYPE,8)
    *menulegalvalue(STATICS)
    *menulegalvalue(MODES)
    *menulegalvalue(BUCK)
    *menulegalvalue(DFREQ)
    *menulegalvalue(MFREQ)
    *menulegalvalue(MBD)
    *menulegalvalue(DTRAN)
    *menulegalvalue(MTRAN)
    *menulegalvalue(DFOUR)
    *menulegalvalue(MFOUR)
    *menulegalvalue(MCEIG)
    *menulegalvalue(HEAT)
    *menulegalvalue(FATIGUE)
    *menulegalvalue(NLSTAT)
    *menulegalvalue(RSPEC)
    *menulegalvalue(EXPDYN)
    *menulegalvalue(SUBCOM)
    *menulegalvalue(NLHEAT)
    *menulegalvalue(RANDOM)
*menuendif()
```

Errors
None.

Version History
2019

@allowduplicateids()

Returns 1 if the allow duplicate IDs option is enabled via `*allowduplicateids`, 0 otherwise.

Syntax
`@allowduplicateids` *()*

Type
HyperMesh Template Function

Description
Returns 1 if the allow duplicate IDs option is enabled via `*allowduplicateids`, 0 otherwise.

Example

```
*if([@allowduplicateids() ==0]) //do not allow duplicate IDs
*string("#Data1")
*end()
*endif()

*if([@allowduplicateids() ==1]) //allow duplicate IDs
*string("#Data2")
*end()
*endif()
```

Version History

11.0.130

@asin()

Trigonometric arc sine of x, the result is expressed in radians between -pp/2 and pp/2.

Syntax

`@asin` *(x)*

Type

HyperMesh Template Function

Inputs

x

Value of type real.

@atan()

Trigonometric arc tangent of x, the result is expressed in radians between -pp/2 and pp/2.

Syntax

`@atan` *(x)*

Type

HyperMesh Template Function

Inputs

x

Value of type real.

@atan2()

Trigonometric arc tangent of x/y, with the result is expressed in radians between -pp and pp.

Syntax

`@atan2` *(x, y)*

Type

HyperMesh Template Function

Inputs

x
: Value of type real, and should be expressed in radians.

y
: Value of type real, and should be expressed in radians.

@attributearray2dcols()

Returns the number of columns in a 2D array attribute.

Syntax

`@attributearray2dcols` *(attribute)*

Type

HyperMesh Template Function

Inputs

attribute
: The name of the attribute (must start with '$').

@attributearray2drows()

Returns the number of rows in a 2D array attribute.

Syntax

`@attributearray2drows` *(attribute)*

Type

HyperMesh Template Function

Inputs

attribute
: The name of the attribute (must start with '$').

@attributearray2dvalue()

Returns the value of a 2D array attribute.

Syntax

`@attributearray2dvalue` *(attribute, row, column)*

Type

HyperMesh Template Function

Inputs

attribute
: The name of the attribute (must start with '$').

row
: The row number (starting at 1).

column
: The column number (starting at 1).

@attributearraylength()

Returns the length of a 1D array attribute.

Syntax

`@attributearraylength` *(attribute)*

Type

HyperMesh Template Function

Inputs

attribute
: The name of the attribute (must start with '$').

@attributearrayvalue()

Returns the value of 1D array attribute.

Syntax

`@attributearrayvalue` *(attribute, index)*

Type

HyperMesh Template Function

Inputs

attribute
The name of the attribute (must start with '$').

index
Index into the array (starting at 1).

@attributearrayvalueinternalid()

Returns the internal ID of a 1D array attribute.

Syntax

`@attributearrayvalueinternalid` *(attribute, index)*

Type

HyperMesh Template Function

Description

Returns the internal ID of a 1D array attribute.

Inputs

attribute
The name of the attribute to query, starting with $.

index
The index into the attribute array, starting at 1.

Examples

To get the entity ID for attribute $Test at index 3:

```
@attributearrayvalueinternalid($Test, 3)
```

Version History

2019

@attributeindexarray2dcols()

Returns the number of rows for a 2D array attribute on an entity.

Syntax

`@attributeindexarray2dcols` *(<entity index>)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

@attributeindexarray2drows()

Returns the number of rows for a 2D array attribute on an entity.

Syntax

`@attributeindexarray2drows` *(entity index)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

@attributeindexarray2dvalue()

Returns the value of a 2D array attribute on an entity.

Syntax

`@attributeindexarray2dvalue` *(entity index, row, col)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

row
Indices into the attribute array (both start at 1).

col
Indices into the attribute array (both start at 1).

@attributeindexarraylength()

Returns the length of a 1D array attribute on an entity.

Syntax

`@attributeindexarraylength` *(entity index)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

@attributeindexarrayvalue()

Returns the value of a 1D array attribute on an entity.

Syntax

`@attributeindexarrayvalue` *(entity index, array index)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

array index
The index into the attribute array (starting at 1).

@attributeindexbehavior()

Returns the behavior of an attribute on an entity.

Syntax

`@attributeindexbehavior` *(entity index)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

@attributeindexentityid()

Returns the entity ID of an entity attribute on an entity.

Syntax

`@attributeindexentityid` *(entity index)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

@attributeindexentitytype()

Returns the entity type (number) of an entity attribute on an entity.

Syntax

`@attributeindexentitytype` *(entity index)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

@attributeindexentitytypename()

Returns the entity type (string) of an entity attribute on an entity.

Syntax

`@attributeindexentitytypename` *(entity index)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

@attributeindexidentifier()

Returns the identifier of an attribute on an entity.

Syntax

`@attributeindexidentifier` *(entity index)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

@attributeindexsolver()

Returns the solver of an attribute on an entity.

Syntax

`@attributeindexsolver` *(entity index)*

Type

HyperMesh Template Function

Inputs

entity index
The index of the attribute owned by the current entity (starting at 1).

@attributeindexstatus()

Returns the status of an attribute on an entity.

Syntax

`@attributeindexstatus` *(entity index)*

Type

HyperMesh Template Function

Inputs

entity index

The index of the attribute owned by the current entity (starting at 1).

@attributeindextype()

Returns the type of attribute on an entity.

Syntax

`@attributeindextype` *(index)*

Type

HyperMesh Template Function

Inputs

index

The index of the attribute owned by the current entity (starting at 1). The return types are:

1 integer
2 double
3 string
4 1D integer array
5 1D double array
6 entity
7 (not supported)
8 (not supported)
9 2D integer array
10 2D double array
11 string array
12 1D entity array
13 2D entity array

Example

If you want to print the types of all attributes on nodes:

```
*nodes()
*format()
 *counterset(counter1,1)
 *loopif([counter1 <= attributesmax])
   *field(integer,[@attributeindextype(counter1)],5)
   *end()
   *coutnerinc(counter1)
 *endloop()
*output()
```

@attributeindexvalue()

Returns the value of an attribute on an entity.

Syntax

`@attributeindexvalue` *(entity index)*

Type

HyperMesh Template Function

Inputs

entity index
: The index of the attribute owned by the current entity (starting at 1).

@attributereferencecount()

Returns the number of times an entity is referenced by an attribute.

Syntax

`@attributereferencecount` *(entity type, id)*

Type

HyperMesh Template Function

Inputs

entity type
: The type of entity referenced.

id
: The entity ID.

Example

If you want to output only curves that are referenced, such as by a material or load, use an `*if` statement as follows:

```
*curves()
  *format()
   *if([@attributereferencecount(curves,id) > 0)])
      *string("Load Curve #")
      *field(integer,id,0)
      *end()
    *endif()
*output()
```

The block above writes out only referenced curves, and those generated via post-processing operations are omitted (if they are not pointed to by an attribute).

@attributesmaxforsolver()

Returns the number of attributes on the current entity that belong to a solver.

Syntax

`@attributesmaxforsolver` *(solver)*

Type

HyperMesh Template Function

Inputs

solver

The solver number. This function will count the number of attributes that are used for 'solver'. A result of zero indicates that there is no card image for 'solver' on the current entity.

Example

```
*nodes()
  *format()
    *uservariableset(#max,[@attributesmaxforsolver(1)])
    *if([#max != 0])
      *string("Node ")
      *field(integer,id,0)
      *string(" contains ")
      *field(integer,#max,0)
      *string(" nastran attributes")
      *end()
    *endif()
*output()
```

@autocreateproperty()

Returns the status of the "auto create property" export option.

Syntax

`@autocreateproperty` *()*

Type

HyperMesh Template Function

Description

Returns 1 if the "auto create property" export option is enabled via `*feoutputwithdata`, 0 otherwise.

Example

```
*if([@autocreateproperty() ==0]) //do not export unresolved IDs
*string("#Data1")
*end()
*endif()
```

```
*if([@autocreateproperty() ==1]) //export unresolved IDs
*string("#Data2")
*end()
*endif()
```

Version History

13.0

@checkfile()

Checks for a file.

Syntax

`@checkfile` *(file name)*

Type

HyperMesh Template Function

Inputs

file name
File name to be checked.

Example

```
*text()
  *if([@checkfile(\tmp\file)=1])
     *string("file exists")
  *else()
     *string("file does not exist")
  *endif()
*output()
```

Return 1 if the file exists. On a PC, use backslashes for the path (`\tmp\file`). On Unix, use forward slashes for the path (`/tmp/file`).

@checkmergeinclude()

Returns the state of the "merge include" export option.

Syntax

`@checkmergeinclude` *()*

Type

HyperMesh Template Function

Description

Returns 1 if the "merge include" export option is enabled via `*feoutputmergeincludefiles`, 0 otherwise.

Example

```
*if([@checkmergeinclude() ==0]) //do not merge include
*string("#Data1")
*end()
*endif()

*if([@checkmergeinclude() ==1]) //merge include
*string("#Data2")
*end()
*endif()
```

Version History

11.0.130

@compressNodeElems()

Returns 0 if HM_NODEELEMS_SET_COMPRESS_SKIP is passed via `*feoutputwithdata`.

Syntax

`@compressNodeElems` *()*

Type

HyperMesh Template Function

Description

Returns 0 if HM_NODEELEMS_SET_COMPRESS_SKIP is passed via `*feoutputwithdata`.

Example

```
*if([@compressNodeElems()])
// counter7 is 1 that means for resp. type we have to compress node/elems
*counterset(counter7,1)
*counterset(counter1,0)
*loopif([counter1 < idsmax])
*pointerset(pointer1,ids,counter1)
*rangeadd(pointer1.pointervalue)
*counterinc(counter1)
*endloop()
*counterset(counter1,[@rangecount()])
*counterset(counter2,0)
*counterset(counter3,0)
*counterset(counter4,0)
*counterset(counter5,0)
*counterset(counter6,0)
*counterset(counter8,0)
*if([counter1 > 0])
*loopif([counter2 < counter1])
```

```
*counterset(counter3,[@rangestart(counter2+1)])
*counterset(counter4,[@rangeend(counter2+1)])
// Store ids
*if([counter3==counter4])
*tablestore(counter6,counter3)
*counterinc(counter6)
*endif()
*if([(counter3+1)==counter4])
*tablestore(counter6,counter3)
*counterinc(counter6)
*tablestore(counter6,counter4)
*counterinc(counter6)
*endif()
//Write Ranges first
*if([counter4>(counter3+1)])
// Check if it is first line or next continuation line
*if([counter8==1])
*end()
*string("+       ")
*endif()
*fieldleft(integer,counter3,8)
*string("  THRU  ")
*fieldleft(integer,counter4,8)
*counterset(counter8,1)
*endif()
*counterinc(counter2)
*endloop()
// Write ids
*if([counter6>=1])
*counterset(counter9,2)
// Check if it is first line or next continuation line
*if([counter8==1])
*counterset(counter9,0)
*endif()
*loopif([counter5 < counter6])
*if([(counter9 % 8) == 0])
*end()
*string("+       ")
*endif()
*fieldleft(integer,[@lookup(counter5)],8)
*counterinc(counter5)
*counterinc(counter9)
*endloop()
*tablereset()
*endif()
*endif()
*rangereset()
*if([counter1== 0])
*end()
*endif()
*endif()
```

Version History

14.0.110

@controlcardattributedefined()

Returns 1 if an attribute exists, 0 otherwise.

Syntax

`@controlcardattributedefined` *(ctrl card name, attribute name)*

Type

HyperMesh Template Function

Inputs

ctrl card name

attribute name

Example

If a Control Card or attribute is not defined, a 0 is returned.

@cos()

Trigonometric cosine of x, where x is expressed in radians.

Syntax

`@cos` *(x)*

Type

HyperMesh Template Function

Inputs

x

A value of type real.

@count()

Counts the entities in the database.

Syntax

`@count` *(entity type, config, type)*

Type

HyperMesh Template Function

Inputs

entity type
The type of entity to be counted. This parameter may be set to any of the entities in the database.

config
The configuration number of the entities to be counted. This parameter is used only if *entity type* is set to elements or loads. If set to zero, the entities are counted regardless of their configuration.

type
The type number of the entities to be counted. This parameter is used only if *entity type* is set to elements. If set to zero, all of the entities are counted regardless of their type.

Example

If the displayed option is selected (active) on the Export Data panel, the value returned by `@count()` includes only those entities that are currently displayed.

@defaultstatus()

Returns the default status of an attribute.

Syntax

`@defaultstatus` *(attribute name)*

Type

HyperMesh Template Function

Example

If the attribute is set to the default value (grayed out in the card previewer), the function returns 1; otherwise, it returns 0.

@defined()

Tests to see if a dictionary item or an attribute is defined.

Syntax

`@defined` *(dictionary item|attribute_name)*

Type

HyperMesh Template Function

Inputs

dictionary item
The name of a dictionary item. This function returns 1 if the dictionary item is active or 0 if the dictionary item is not active. If the dictionary item does not exist, the function returns 0.

attribute_name
The name of the attribute.

Example

This function allows dictionary items to be set as a toggle that you can turn off and on. During translation, `@defined()` can be used to see if the item has been toggled on or off.

If the attribute with the given name is defined on this entity, the function returns 1; otherwise, it returns 0. If it cannot find the attribute on the current entity, `@defined` will also check the dictionaries.`hm_defined` does not check the dictionaries because the dictionaries cannot be accessed outside the template.

@dofs()

Extracts individual degrees of freedom from an integer and returns the status.

Syntax

`@dofs` *(dof, position)*

Type

HyperMesh Template Function

Inputs

dof
Generally an integer beam end release code that returns 1 or 0 (on or off) if the integer in *position* is contained within the *dof* field.

position

Example

```
@dofs(123,1) = 1, @dofs(456,1) = 0
```

@elemcountperinclude()

Returns the number of elements of a specific configuration and type for a specific component being exported to an include file.

Syntax

`@elemcountperinclude` *(entity_type, config, type, include_id,component_id)*

Type

HyperMesh Template Function

Description

This function returns the number of elements of a specific configuration and type for a specific component being exported to an include file.

Inputs

entity type
: The type of entity to be counted. Currently, this is restricted to be only elements.

config
: The configuration of the elements to be counted. If set to zero, entities are counted regardless of their configuration.

type
: The type of the elements to be counted. If set to zero, entities of the specified config are counted regardless of their type.

include_id
: The ID of the include file.

component_id
: The ID of the component.

Example

To query the elements of config 104 and type 1 in include 1 and component 100:

```
*variableset(variable1,[@elemcountperinclude(104,1,1,100)])
```

Version History

14.0

@entitygettype()

Returns the user-assigned entity type. The user-assigned entity type is set in the template.

Syntax

`@entitygettype` *(entity type, entity id)*

Type

HyperMesh Template Function

Inputs

entity type
: The type of the queried entity.

entity id
The ID of the queried entity.

@entityincollector()

Returns the number of entities in a collector.

Syntax

`@entityincollector` *(entity type, config_num, type_num)*

Type

HyperMesh Template Function

Inputs

entity type
The type of entity to be counted.

config_num
The configuration number of the entities being counted. If set to 0, entities are counted regardless of their configuration.

type_num
The type number of the entities being counted. If set to 0, all the entities are counted regardless of their type.

@entitymaxid()

Returns the maximum ID in use from a type of entity.

Syntax

`@entitymaxid` *(entity type)*

Type

HyperMesh Template Function

Inputs

entity type
The type of the entity.

@enum()

Returns the value of an enumeration.

Syntax

`@enum` *(enum name, enum index)*

Type

HyperMesh Template Function

Inputs

enum name
: The name of the enumeration.

enum index
: The index into the enumeration (starting at 1).

@exists()

Indicates if a pointer is pointing to an entity or if it is set to NULL.

Syntax

`@exists` *(pointer)*

Type

HyperMesh Template Function

Inputs

pointer
: A pointer to an entity. If the pointer is pointing to an entity, the function returns 1. If not, the function returns 0.

@exp()

Exponential of x.

Syntax

`@exp` *(x)*

Type

HyperMesh Template Function

Inputs

x

A value of type real.

Example

```
@exp(2) = e2 = (2.718281828...)2 = 7.389056...
```

@exportdmiginlongformat()

Returns 0 if EXPORT_DMIG_LONGFORMAT is passed via `*feoutputwithdata`.

Syntax

`@exportdmiginlongformat()`

Type

HyperMesh Template Function

Description

Returns 0 if EXPORT_DMIG_LONGFORMAT is passed via `*feoutputwithdata`.

Examples

Example:

```
*if([@exportdmiginlongformat()])
  *if([@writehmcomments()])
    *string("$$")
    *end()
    *string("$HMNAME DIRECTMATRIXINPUTS*        ")
    *field(integer,id,16)
    *quote()
    *field(string,name,0)
    *quote()
    *end()
    *string("$HWCOLOR DIRECTMATRIXINPUTS*       ")
    *field(integer,id,16)
    *field(integer,color,16)
    *end()
    *string("$$")
    *end()
  *endif()
*else()
  *if([@writehmcomments()])
     *string("$$")
     *end()
     *string("$HMNAME DIRECTMATRIXINPUTS         ")
     *field(integer,id,8)
     *quote()
     *field(string,name,0)
     *quote()
     *end()
     *string("$HWCOLOR DIRECTMATRIXINPUTS        ")
     *field(integer,id,8)
```

```
      *field(integer,color,8)
      *end()
      *string("$$")
      *end()
    *endif()
*endif()
```

Version History

2021

@exportsysteminlongformat()

Returns the value of the "Export CORD2 and CORD4 in long format" export option.

Syntax

`@exportsysteminlongformat()`

Type

HyperMesh Template Function

Description

Returns the value of the "Export CORD2 and CORD4 in long format" export option. This option will export the CORD2R and CORD4R system in long format even if the template loaded is of regular format. A return of 1 means export system in long format, and 0 means export system in regular format.

Examples

```
*if([@exportsysteminlongformat() == 1])
   ...write system in long format...
*else()
   ...write system in regular format...
*endif()
```

Version History

2019

@fabs()

Absolute value of x.

Syntax

`@fabs` *(x)*

Type

HyperMesh Template Function

Inputs

x
A value of type real.

@getattributearraylength()

Retrieves the length of a 1D array attribute attached to a specific entity.

Syntax

`@getattributearraylength` *(entity type, entity id, attribute)*

Type

HyperMesh Template Function

Inputs

entity type
The type of entity, such as sets, elements, or nodes, to which *entity id* refers.

entity id
The ID of the entity from which you want to get information.

attribute
The name of the attribute (must start with '$').

@getattributearrayvalue()

Retrieves the value of a 1D array attribute attached to a specific entity.

Syntax

`@getattributearrayvalue` *(entity type, entity id, attribute, index)*

Type

HyperMesh Template Function

Inputs

entity type
The type of entity, such as sets, elements, or nodes, to which *entity id* refers.

entity id
The ID of the entity from which you want to get information.

attribute
The name of the attribute (must start with '$').

index
The index into the array (starting at 1).

@getattributearrayvaluebyinternalid1()

Returns the internal ID of a 1D array attribute attached to an entity.

Syntax

`@getattributearrayvaluebyinternalid1` *(entity_type, id, attribute, index)*

Type

HyperMesh Template Function

Description

Returns the internal ID of a 1D array attribute attached to an entity.

Inputs

entity_type
The type of entity to query.

id
The internal ID of the entity to query.

attribute
The name of the solver attribute array, starting with $.

index
The index into the array to query, starting with 1.

Example

```
*variableset(variable2,[@getattributearrayvaluebyinternalid1(loadcols,variable20,
$DSLoad_Groups_Name,counter5)])
```

Version History

2017.1

@getattributearrayvaluebyinternalid2()

Returns the value (active ID) of a 1D array attribute attached to an entity.

Syntax

`@getattributearrayvaluebyinternalid2` *(entity_type, id, attribute, index)*

Type

HyperMesh Template Function

Description

Returns the internal ID of a 1D array attribute attached to an entity.

Inputs

entity_type
The type of entity to query.

id
The internal ID of the entity to query.

attribute
The name of the solver attribute array, starting with $.

index
The index into the array to query, starting with 1.

Example

```
*variableset(variable2,[@getattributearrayvaluebyinternalid2(loadcols,variable20,
$DSLoad_Groups_Name,counter5)])
```

Version History

2017.1

@getattributevalueinternalid()

Returns the internal ID value of an attribute attached to a specific entity.

Syntax

`@getattributevalueinternalid` *(entity_type, entity_id, attribute, index)*

Type

HyperMesh Template Function

Description

Returns the internal ID value of an attribute attached to a specific entity.

Inputs

entity_type
The type of entity to query.

entity_id
The ID of the entity to query.

attribute
The name of the attribute to query, starting with $.

index
The index into the attribute array, starting at 1.

Examples

To get the entity ID for attribute $Test at index 3 on component 10:

```
@getattributevalueinternalid(comps, 10, $Test, 3)
```

Version History

2019

@getcelltriplevalue()

Returns the values of a triple cell in a table.

Syntax

`@getcelltriplevalue` *(row_index, column_index, triple_index)*

Type

HyperMesh Template Function

Description

This function returns the values of a triple cell in a table. This can only be used inside of a `*tables()` block.

Inputs

row_index
The index of the row in the table, starting from 0.

column_index
The index of the column in the table, starting from 0.

triple_index
The index of the triple value. Valid values are 0-2.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@getcelltype(counter2,counter1)])
```

```
                *if([variable3 == 7])
                    *field(real,[@getcelltriplevalue(counter2,counter1,0)],0)
                    *string(" ")
                    *field(real,[@getcelltriplevalue(counter2,counter1,1)],0)
                    *string(" ")
                    *field(real,[@getcelltriplevalue(counter2,counter1,2)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@getcelltype()

Returns the type of a cell in a table.

Syntax

`@getcelltype` *(row_index, column_index)*

Type

HyperMesh Template Function

Description

This function returns the type of a cell in a table. This can only be used inside of a `*tables()` block.

Inputs

row_index
: The index of the row in the table, starting from 0.

column_index
: The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@getcelltype(counter2,counter1)])
                *if([variable3 == 7])
                    *field(real,[@getcelltriplevalue(counter2,counter1,0)],0)
                    *string(" ")
                    *field(real,[@getcelltriplevalue(counter2,counter1,1)],0)
```

```
                    *string(" ")
                    *field(real,[@getcelltriplevalue(counter2,counter1,2)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@getcellvalue()

Returns the values of a bool, double, float, int, string, unsigned int, or entity ID cell in a table.

Syntax

`@getcellvalue` *(row_index, column_index)*

Type

HyperMesh Template Function

Description

This function returns the values of a bool, double, float, int, string, unsigned int, or entity ID cell in a table. This can only be used inside of a `*tables()` block.

Inputs

row_index
: The index of the row in the table, starting from 0.

column_index
: The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@getcelltype(counter2,counter1)])
                *if([variable3 == 3 || variable3 == 4])
                    *field(real,[@getcellvalue(counter2,counter1)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
```

```
        *endloop()
*output()
```

Version History

11.0

@getcollectorcardimage()

Retrieves the card image of a collector (such as PSHELL, Part, and so on).

Syntax

`@getcollectorcardimage` *(collector type, collector id)*

Type

HyperMesh Template Function

Inputs

collector type
: The type of collector, such as properties or components.

collector ID
: The ID of the collector.

Example

The following template code outputs the name and card image of every component in the model:

```
*components("","")
*format()
*string("component: ")
*field(quotedstring,name,0)
*string("card image: ")
*field(quotedstring,[@getcollectorcardimage(comps,id)],0)
*end()
*output()
```

@getcollectorname()

Retrieves the name of a collector.

Syntax

`@getcollectorname` *(collector type, collector id)*

Type

HyperMesh Template Function

Inputs

collector type
The type of collector (such as properties or components).

collector id
The ID of the collector.

@getcollectornamebyinternalid()

Returns the name of a collector by internal ID.

Syntax

`@getcollectornamebyinternalid` *(entity_type, entity_id)*

Type

HyperMesh Template Function

Description

Returns the name of a collector by internal ID.

Inputs

entity_type
The type of collector entity to query.

entity_id
The ID of the collector entity to query.

Examples

To query component 100:

```
@getcollectornamebyinternalid(comps, 100)
```

Version History

2019

@getcolumnentitytype()

Returns the entity type ID of a column in a table.

Syntax

`@getcolumnentitytyp` *(column_index)*

Type

HyperMesh Template Function

Description

This function returns the entity type ID of a column in a table. This can only be used inside of a `*tables()` block. The entity type ID can be converted to the entity type name using `@getdatabaseentitytypename`.

Inputs

column_index
The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@getcolumntype(counter1)])
                *variableset(variable4,[@getcolumnentitytype(counter1)])
                *if([variable3 == 2 && variable4 > 0])
                    *field(integer,variable4,0)
                    *field(integer,[@getcellvalue(counter2,counter1)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@getcolumnlabel()

Returns the label of a column in a table.

Syntax

`@getcolumnlabel` *(column_index)*

Type

HyperMesh Template Function

Description

This function returns the label of a column in a table. This can only be used inside of a `*tables()` block.

Inputs

column_index
The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *field(string,[@getcolumnlabel(counter1)],0)
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@getcolumntype()

Returns the type ID of a column in a table.

Syntax

`@getcolumntype` *(column_index)*

Type

HyperMesh Template Function

Description

This function returns the type ID of a column in a table. This can only be used inside of a `*tables()` block. The following values are valid for return:

1 - integer
2 - unsigned integer/entity ID
3 - float
4 - double
5 - bool
6 - string
7 - triple

Inputs

column_index
The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@getcolumntype(counter1)])
                *if([variable3 == 7])
                    *field(real,[@getcelltriplevalue(counter2,counter1,0)],0)
                    *string(" ")
                    *field(real,[@getcelltriplevalue(counter2,counter1,1)],0)
                    *string(" ")
                    *field(real,[@getcelltriplevalue(counter2,counter1,2)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@getcontrolcardattribute()

Returns the value of an attribute on a Control Card.

Syntax

`@getcontrolcardattribute` *(control card, attribute)*

Type

HyperMesh Template Function

Inputs

control card
: The name of the Control Card.

attribute
: The attribute name.

Example

If a Control Card and/or attribute is not present, zero is returned for the menu portion, and an error occurs for the format portion. Use `@controlcardattributedefined()` to verify the existence of a Control Card or attribute.

@getcurrentincludeexportformat()

Returns the export format type set using `*setformattype()`.

Syntax

`@getcurrentincludeexportformat` *()*

Type

HyperMesh Template Function

Description

Returns the export format type set using `*setformattype()`.

Examples

```
*if([@getcurrentincludeexportformat() == 1])
   ...write in standard format...
*elseif([@getcurrentincludeexportformat() == 2])
   ...write in long format...
*else()
   ...write in I10 format...
*endif()
```

Version History

2020

@getdatabaseentitytypename()

Gets the type name of an entity from its index.

Syntax

`@getdatabaseentitytypename` *(index)*

Type

HyperMesh Template Function

Description

This function is used to get the HyperMesh entity type name corresponding to index, where index is a value between 1 and globalentitytypesmax (both inclusive).

Example

The following code prints all HyperMesh entity type names by calling the function `@getdatabaseentitytypename` for every valid index:

```
*text()
 *string("--------------List of entity types---------------")
 *end()
 *string("     Index            HM entity type name        ")
```

```
  *end()
  *string("-------------------------------------------------")
  *end()
  *counterset(counter1,1)
  *loopif([counter1 <= globalentitytypesmax])
    *string("         ")
    *field(integer,counter1,0)
    *string("                    ")
    *if([counter1 < 10])
      *string(" ")
    *endif()
    *field(string,[@getdatabaseentitytypename(counter1)],0)
    *end()
    *counterinc(counter1)
  *endloop()
  *string("-------------------------------------------------")
  *end()
*output()
```

Errors

If an error occurs in the execution of this command during export, the export is stopped at that point with an error message.

@getentityarrayvalue()

Returns the array value of an entity.

Syntax

`@getentityarrayvalue` *(entity_type, entity_id, data_name, index)*

Type

HyperMesh Template Function

Description

Returns the array value of an entity.

Inputs

entity_type
: The type of entity to query.

entity_id
: The ID of the entity to query.

data_name
: The array data name to query. Note that "." extentions for data names are not supported.

index
: The index into the array to query, starting at 0.

Examples

To export all the IDs of a set:

```
*counterset(counter1,0)
*loopif([counter1 != idsmax])
    *field(int,[@getentityarrayvalue(sets,id,ids,counter1)],10)
    *counterinc(counter1)
*endloop()
```

Version History

2019

@getentitycount()

Returns the entity count of an entity array type data name.

Syntax

`@getentitycount` *(data_name)*

Type

HyperMesh Template Function

Description

Returns the entity count of an entity array type data name.

Inputs

data_name
The data name to query.

Examples

To store the count of slavecompids into variable16:

```
*variableset(variable16,[@getentitycount(slavecompids)])
```

Version History

2019

@getentitytype()

Returns the entity type of an entity array type data name.

Syntax

`@getentitytype` *(data_name)*

Type

HyperMesh Template Function

Description

Returns the entity type of an entity array type data name.

Inputs

data_name
The data name to query.

Examples

To check the entity type of slavecompids:

```
*if([@getentitytype(slavecompids) == 3])
```

Version History

2019

@getentityvalue()

Retrieves the value of an entity.

Syntax

`@getentityvalue` *(entity type, entity id, data name)*

Type

HyperMesh Template Function

Inputs

entity type
The type of entity, such as sets, elems, or nodes, to which *entity id* refers.

entity id
The ID of the entity you want to reference.

data name
The data name of the entity. For example, ID, name, node1.globalx.

Example

To output the name of a set with the ID, 1:

```
*field (string,[@getentityvalue(sets, 1, name)], 32)
```

The `@getentityvalue` function allows you to get a value from an entity if you know the entity's type and ID. It allows you to get a value of the collector which contains an entity. This function searches the database and may access the data values slower than other commands, such as `*sets()`.

@getentityvaluebyinternalid()

Returns the value of an entity data name.

Syntax

`@getentityvaluebyinternalid` *(entity_type, entity_id, data_name)*

Type

HyperMesh Template Function

Description

Returns the value of an entity data name.

Inputs

entity_type
: The type of entity to query.

entity_id
: The ID of the entity to query.

data_name
: The data name to query.

Examples

To output the name of a set with the internal ID 1:

```
*field (string,[@getentityvaluebyinternalid(sets, 1, name)], 32)
```

Version History

2019

@getentityvalueinternalid()

Returns the internal ID of an entity.

Syntax

`@getentityvalueinternalid` *(entity_type, entity_id, data_name)*

Type

HyperMesh Template Function

Description

Returns the internal ID of an entity.

Inputs

entity_type
The type of entity to query.

entity_id
The ID of the entity to query.

data_name
The entity data name to query.

Examples

To output the node1 internal ID of the element with the internal ID 1:

```
*variableset(variable16,[@getentityvalueinternalid(elems,1,node1)]);
```

Version History

2019

@getidoffsetvalue()

Returns the ID offset value for an input entity with ID pool number in a specified include file ID.

Syntax

`@getidoffsetvalue` *(entity_type, pool_id, include_file_id)*

Type

HyperMesh Template Function

Description

Returns the ID offset value for an input entity with ID pool number in a specified include file ID.

Inputs

entity_type
The type of entity to query.

pool_id
The ID pool number of the entity to query, if any.

include_file_id
The include file ID for which the offset is required.

Examples

To get the ID offset for cards in include ID 5:

```
*variableset(variable1,[@getidoffsetvalue(cards,0,5)])
```

Version History

2019

@getincludefullname()

Returns the include file full name and path for a given include ID.

Syntax

`@getincludefullname` *(include_id)*

Type

HyperMesh Template Function

Description

Returns the include file full name and path for a given include ID.

Inputs

include_id
The ID of the include file to query.

Examples

To get the full name for include ID 10:

```
*field(quotedstring,[@getincludefullname(10)],0)
```

Version History

2019

@getincludeidbyfilename()

Returns the ID of an include file name.

Syntax

`@getincludeidbyfilename` *(filename)*

Type

HyperMesh Template Function

Description

Returns the ID of the include file if it exists, -1 if it does not, and 0 if it is the master file.

Inputs

filename
The name of the include file.

Example

```
*variableset(variable1,"[@getincludeidbyfilename($FileNameTran)]")
*if([ @getincludereferenceflag(variable1,1) == 1) ])
*string("*INCLUDE_PATH")
*end()
*field(string,[@inclstrsplit($FileNameTran,+)],72)
*end()
*endif()
```

Version History

14.0

@getincludereferenceflag()

Returns the referenced state of an include in the master file.

Syntax

`@getincludereferenceflag` *(include_id, exception)*

Type

HyperMesh Template Function

Description

Returns 1 if *include_id* is referenced into the master file, 0 otherwise.

Inputs

include_id
The ID of the include.

exception
0 - The include file reference depends on its export status.
1 - The include file reference depends on its export status, except for Include_Transform which is always referenced regardless its export status.

Example

```
*variableset(variable1,"[@getincludeidbyfilename($FileNameTran)]")
*if([ @getincludereferenceflag(variable1,1) == 1) ])
*string("*INCLUDE_PATH")
*end()
*field(string,[@inclstrsplit($FileNameTran,+)],72)
*end()
*endif()
```

Version History

14.0

@getinternalid()

This command, used in feoutput and summary templates, returns an internal ID based on the pool number and solver ID of an entity. This command needs to be used in conjunction with the command `@getentityvalue()`, which returns the solver ID. You must know the pool number corresponding to an entity type of attribute.

Syntax

`@getinternalid` *(poolnumber, solverid)*

Type

HyperMesh Template Function

Description

Inputs

poolnumber
: The pool number to which the entity belongs.

solverId
: The ID of the solver that you use.

Example

```
 @getinternalid( 22, 1)
```

Where 22 is pool number and 1 is solver ID.

Actual usage:

```
*variableset(variable1,[@getentityvalue(components, id, $LSD_SID)])
```

Where ID is the comp ID.

```
*variableset(variable2, [@getinternalid(20, variable1)])
```

Where the value 20 is the pool number belonging to the entity type attribute $LSD_SID defined the dyna.key template and variable1 is solver ID of the attribute $LSD_SID. The variable2 will be the internal ID of the attribute $LSD_SID.

@getnewid()

Returns the next valid ID by checking IDs of other entities.

Syntax

`@getnewid` *(entity_type, id, pool_id, entity_type_check, pool_id_check)*

Type

HyperMesh Template Function

Description

This function returns the values of a triple cell in a table. This can be used both inside and outside of a `*tables()` block.

Inputs

entity_type
: The type of the entity to query.

entity_id
: The ID of the entity to query.

pool_id
: The ID pool number to which the queried entity belongs, if any.

entity_type_check
: The type of the entity to check for a conflict with.

pool_id_check
: The ID pool number to which the conflict entity belongs, if any.

Example

```
*variableset(variable9,[@getnewid(CONTACTSURFS,id,0,SETS,40)])
```

Version History

14.0.110

@getsolverid()

Returns solver ID based on entity type and internal ID of an entity. This needs to be used in feoutput and summary templates.

Syntax

`@getsolverid` *(entity type, id)*

Type

HyperMesh Template Function

Inputs

entityType
The entity type of the entity for which you wish to retrieve the solver ID.

id
The entity ID of the entity for which you wish to retrieve the solver ID.

Example

```
@getsolverid(ELEMS,99)
@getsolverid(PROPS,1)
*variableset(variable1,[@getsolverid(ELEMS, 99)])
```

Where variable1 will be the solver ID corresponding to the internal ID 99.

@gettablecelltriplevalue()

Returns the values of a triple cell in a table.

Syntax

`@gettablecelltriplevalue` *(table_id, row_index, column_index,triple_index)*

Type

HyperMesh Template Function

Description

This function returns the values of a triple cell in a table. This can be used both inside and outside of a `*tables()` block.

Inputs

table_id
The ID of the table.

row_index
The index of the row in the table, starting from 0.

column_index
The index of the column in the table, starting from 0.

triple_index
The index of the triple value. Valid values are 0-2.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
```

```
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@gettablecelltype(id,counter2,counter1)])
                *if([variable3 == 7])
                    *field(real,[@gettablecelltriplevalue(id,counter2,counter1,0)],0)
                    *string(" ")
                    *field(real,[@gettablecelltriplevalue(id,counter2,counter1,1)],0)
                    *string(" ")
                    *field(real,[@gettablecelltriplevalue(id,counter2,counter1,2)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@gettablecelltype()

Returns the type of a cell in a table.

Syntax

`@gettablecelltype` *(table_id, row_index, column_index)*

Type

HyperMesh Template Function

Description

This function returns the type of a cell in a table. This can be used both inside and outside of a `*tables()` block.

Inputs

table_id
: The ID of the table.

row_index
: The index of the row in the table, starting from 0.

column_index
: The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
```

```
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@gettablecelltype(id,counter2,counter1)])
                *if([variable3 == 7])
                    *field(real,[@gettablecelltriplevalue(id,counter2,counter1,0)],0)
                    *string(" ")
                    *field(real,[@gettablecelltriplevalue(id,counter2,counter1,1)],0)
                    *string(" ")
                    *field(real,[@gettablecelltriplevalue(id,counter2,counter1,2)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@gettablecelltypebyinternalid()

Returns the type of a cell in a table.

Syntax

`@gettablecelltypebyinternalid` *(table_id, row_index, column_index)*

Type

HyperMesh Template Function

Description

Returns the type of a cell in a table.

Inputs

table_id
: The ID of the table entity to query.

row_index
: The row of the table to query, starting from 0.

column_index
: The colum of the table to query, starting from 0

Examples

To query table 100, row 3 and column 10:

```
@gettablecelltypebyinternalid(100, 3, 10)
```

Version History

2019

@gettablecellvalue()

Returns the values of a bool, double, float, int, string, unsigned int, or entity ID cell in a table.

Syntax

`@gettablecellvalue` *(table_id, row_index, column_index)*

Type

HyperMesh Template Function

Description

This function returns the values of a bool, double, float, int, string, unsigned int, or entity ID cell in a table. This can be used both inside and outside of a `*tables()` block.

Inputs

table_id
: The ID of the table.

row_index
: The index of the row in the table, starting from 0.

column_index
: The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@gettablecelltype(id,counter2,counter1)])
                *if([variable3 == 3 || variable3 == 4])
                    *field(real,[@gettablecellvalue(id,counter2,counter1)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@gettablecellvaluebyinternalid()

Returns the values of a bool, double, float, int, string, unsigned int, or entity ID cell in a table.

Syntax

`@gettablecellvaluebyinternalid` *(table_id, row_index, column_index)*

Type

HyperMesh Template Function

Description

Returns the values of a bool, double, float, int, string, unsigned int, or entity ID cell in a table.

Inputs

table_id
: The ID of the table entity to query.

row_index
: The row of the table to query, starting from 0.

column_index
: The colum of the table to query, starting from 0

Examples

To query table 100, row 3 and column 10:

```
@gettablecellvaluebyinternalid(100, 3, 10)
```

Version History

2019

@gettablecolumnentitytype()

Returns the entity type ID of a column in a table.

Syntax

`@gettablecolumnentitytype` *(table_id, column_index)*

Type

HyperMesh Template Function

Description

This function returns the entity type ID of a column in a table. This can be used both inside and outside of a `*tables()` block. The entity type ID can be converted to the entity type name using `@getdatabaseentitytypename`.

Inputs

table_id
The ID of the table.

column_index
The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@getcolumntype(counter1)])
                *variableset(variable4,[@gettablecolumnentitytype(id,counter1)])
                *if([variable3 == 2 && variable4 > 0])
                    *field(integer,variable4,0)
                    *field(integer,[@getcellvalue(counter2,counter1)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@gettablecolumnlabel()

Returns the label of a column in a table.

Syntax

`@gettablecolumnlabel` *(table_id, column_index)*

Type

HyperMesh Template Function

Description

This function returns the label of a column in a table. This can be used both inside and outside of a `*tables()` block.

Inputs

table_id
The ID of the table.

column_index
The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *field(string,[@gettablecolumnlabel(id,counter1)],0)
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@gettablecolumnsize()

Returns the size (number of rows) of a column in a table.

Syntax

`@gettablecolumnsize` *(table_id, column_index)*

Type

HyperMesh Template Function

Description

This function returns the size (number of rows) of a column in a table. This can be used both inside and outside of a `*tables()` block.

Inputs

table_id
The ID of the table.

column_index
The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@getcelltype(counter2,counter1)])
```

```
                *if([variable3 == 3 || variable3 == 4])
                    *field(real,[@getcellvalue(counter2,counter1)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@gettablecolumnsizebyinternalid()

Returns the size (number of rows) of a column in a table.

Syntax

`@gettablecolumnsizebyinternalid` *(table_id, column_index)*

Type

HyperMesh Template Function

Description

Returns the size (number of rows) of a column in a table.

Inputs

table_id
: The ID of the table entity to query.

column_index
: The colum of the table to query, starting from 0

Examples

To query table 100, column 10:

```
@gettablecolumnsizebyinternalid(100, 10)
```

Version History

2019

@gettablecolumntype()

Returns the type ID of a column in a table.

Syntax

`@gettablecolumntype` *(table_id, column_index)*

Type

HyperMesh Template Function

Description

This function returns the type ID of a column in a table. This can be used both inside and outside of a `*tables()` block. The following values are valid for return:

1 - integer
2 - unsigned integer/entity ID
3 - float
4 - double
5 - bool
6 - string
7 - triple

Inputs

table_id
The ID of the table.

column_index
The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@gettablecolumntype(id,counter1)])
                *if([variable3 == 7])
                    *field(real,[@getcelltriplevalue(counter2,counter1,0)],0)
                    *string(" ")
                    *field(real,[@getcelltriplevalue(counter2,counter1,1)],0)
                    *string(" ")
                    *field(real,[@getcelltriplevalue(counter2,counter1,2)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
```

```
*output()
```

Version History

11.0

@gettablecolumntypestring()

Returns the type string of a column in a table.

Syntax

`@gettablecolumntype` *(table_id, column_index)*

Type

HyperMesh Template Function

Description

This function returns the type string of a column in a table. This can be used both inside and outside of a `*tables()` block. The following values are valid for return:

int
unsignedint
float
double
bool
string
triple
HyperMesh entity types (for example elements, plies, components, and so on)

Inputs

table_id
The ID of the table.

column_index
The index of the column in the table, starting from 0.

Example

```
*tables()
    *format()
        *counterset(counter1,0)
        *variableset(variable1,columns)
        *loopif([counter1 < variable1])
            *counterset(counter2,0)
            *variableset(variable2,[@gettablecolumnsize(id,counter1)])
            *loopif([counter2 < variable3])
                *variableset(variable3,[@gettablecolumntypestring(id,counter1)])
                *if([variable3 == "triple"])
                    *field(real,[@getcelltriplevalue(counter2,counter1,0)],0)
```

```
                    *string(" ")
                    *field(real,[@getcelltriplevalue(counter2,counter1,1)],0)
                    *string(" ")
                    *field(real,[@getcelltriplevalue(counter2,counter1,2)],0)
                    *end()
                *endif()
                *counterinc(counter2)
            *endloop()
            *counterinc(counter1)
        *endloop()
*output()
```

Version History

11.0

@gettotalmass()

Returns different total mass values of the model.

Syntax

`@gettotalmass` *(?type?)*

Type

HyperMesh Template Function

Description

Returns different total mass values of the model.

Inputs

type

The type of mass to query. If not specified, the total mass is returned.
structural - The total structural mass of the model.
nonstructural - The total non-structural mass of the model.
lumped - The total lumped mass of the model.
rigid - The total rigid mass of the model.
transferred - The total transferred mass of the model.
engineering - The total engineering mass of the model.
distributed - The total distributed mass of the model.

Examples

To query the total mass:

```
@gettotalmass()
```

To query the rigid mass:

```
@gettotalmass("rigid")
```

Version History

2019

@icedependentcount()

Returns the number of ICE dependent nodes for an independent node.

Syntax

`@icedependentcount` *(independent_node_id)*

Type

HyperMesh Template Function

Description

Returns the number of ICE dependent nodes for an independent node.

Inputs

independent_node_id
The ID of the ICE independent node.

Example

```
*elements(57,0,"ICE","")
*format()
*string("*ice(")
*field(integer,id,0)
*string(",")
*field(integer,type,0)
*string(",")
*field(integer,independentnodesmax,0)
*string(",")
*field(integer,dependentnodesmax,0)
*string(")")
*end()

*counterset(counter1,0)
*loopif([counter1 < independentnodesmax])
*string("  *icelink(")
*pointerset(pointer1,independentnodes,counter1)
*field(integer,pointer1.pointervalue,0)
*string(",")
*counterset(counter2,[@icedependentcount(pointer1.pointervalue)])
*field(integer,counter2,0)
*counterset(counter3,0)
*loopif([counter3 < counter2])
*string(",")
*field(integer,[@icedependentnode(pointer1.pointervalue,counter3)],0)
*string(",")
*field(integer,[@icedependentdof(pointer1.pointervalue,counter3)],0)
*counterinc(counter3)
*endloop()
*counterinc(counter1)
*string(")")
```

```
*end()
*endloop()
*output()
```

Version History

14.0

@icedependentdof()

Returns ICE dependent node DOFs for an independent node.

Syntax

`@icedependentdof` *(independent_node_id, index)*

Type

HyperMesh Template Function

Description

Returns ICE dependent node DOFs for an independent node.

Inputs

independent_node_id
: The ID of the ICE independent node.

index
: The index of the dependent node, starting from 0.

Example

```
*elements(57,0,"ICE","")
*format()
*string("*ice(")
*field(integer,id,0)
*string(",")
*field(integer,type,0)
*string(",")
*field(integer,independentnodesmax,0)
*string(",")
*field(integer,dependentnodesmax,0)
*string(")")
*end()

*counterset(counter1,0)
*loopif([counter1 < independentnodesmax])
*string("  *icelink(")
*pointerset(pointer1,independentnodes,counter1)
*field(integer,pointer1.pointervalue,0)
*string(",")
*counterset(counter2,[@icedependentcount(pointer1.pointervalue)])
*field(integer,counter2,0)
*counterset(counter3,0)
*loopif([counter3 < counter2])
```

```
*string(",")
*field(integer,[@icedependentnode(pointer1.pointervalue,counter3)],0)
*string(",")
*field(integer,[@icedependentdof(pointer1.pointervalue,counter3)],0)
*counterinc(counter3)
*endloop()
*counterinc(counter1)
*string(")")
*end()
*endloop()
*output()
```

Version History

14.0

@icedependentnode()

Returns ICE dependent nodes for an independent node.

Syntax

`@icedependentnode` *(independent_node_id, index)*

Type

HyperMesh Template Function

Description

Returns ICE dependent nodes for an independent node.

Inputs

independent_node_id
: The ID of the ICE independent node.

index
: The index of the dependent node, starting from 0.

Example

```
*elements(57,0,"ICE","")
*format()
*string("*ice(")
*field(integer,id,0)
*string(",")
*field(integer,type,0)
*string(",")
*field(integer,independentnodesmax,0)
*string(",")
*field(integer,dependentnodesmax,0)
*string(")")
*end()

*counterset(counter9,0)
*loopif([counter9 < independentnodesmax])
```

```
*string("  *icelink(")
*pointerset(pointer1,independentnodes,counter9)
*field(integer,pointer1.pointervalue,0)
*counterset(counter4,[@icedependentcount(pointer1.pointervalue)])
*counterset(counter3,0)
*string(",")
*field(integer,counter4,0)
*loopif([counter3 < counter4])
*string(",")
*field(integer,[@icedependentnode(pointer1.pointervalue,counter3)],0)
*string(",")
*field(integer,[@icedependentdof(pointer1.pointervalue,counter3)],0)
*counterinc(counter3)
*endloop()
*counterinc(counter9)
*string(")")
*end()
*endloop()
*output()
```

Version History

14.0

@keywordrenumber()

Returns 1 if RENUMBER is passed with the queried keyword via `*feoutputwithdata`.

Syntax

`@keywordrenumber` *("keyword")*

Type

HyperMesh Template Function

Description

This function returns 1 if RENUMBER is passed with the queried keyword via `*feoutputwithdata`.

Inputs

keyword
: The keyword attribute name.

Example

```
*if([@keywordrenumber("MPC_GENERAL")])
*field(integer,[id+#maxMPCId],10)
*else()
*field(integer,id,10)
*endif()
```

Version History

10-SA1-110

@loadexistinmergedloadloadsteptable()

Returns 1 if the load exists in the table containing merged loads for a loadstep created by `*createmergedloadloadsteptable()`.

Syntax

`@loadexistinmergedloadloadsteptable` *(id)*

Type

HyperMesh Template Function

Description

This function returns 1 if the load exists in the table containing merged loads for a loadstep created by `*createmergedloadloadsteptable()`.

Inputs

id

The ID of the load to query.

Example

```
*if([@loadexistinmergedloadsteptable(id)])
*string("This is a summed load")
*else()
*string("This is not a summed load")
*endif()
```

Version History

11.0.101

@log()

Natural logarithm.

Syntax

`@log` *(x)*

Type

HyperMesh Template Function

Inputs

x

A value of type real.

Example

```
@log(2) = LN(2) = 0.693147
```

@log10()

Logarithm of x to the base 10.

Syntax

`@log10` *(x)*

Type

HyperMesh Template Function

Inputs

x

A value of type real.

Example

```
@log(100) = Log10(100) = 2.000
```

@lookup()

Retrieves a value stored in a lookup table.

Syntax

`@lookup` *(key)*

Type

HyperMesh Template Function

Inputs

key

Used to compare the keys found on the lookup table entries. If *key* matches one of the keys in the lookup table, the function returns the value associated with that entry. If a matching key is not found, the function returns 0.

@magnitude()

Returns the magnitude of a vector.

Syntax

`@magnitude` *(x comp, y comp, z comp)*

Type

HyperMesh Template Function

Inputs

x comp
: The components of the vector to be evaluated.

y comp
: The components of the vector to be evaluated.

z comp
: The components of the vector to be evaluated.

@namedentity()

Returns the value of the "Export as named entity" export option.

Syntax

`@namedentity()`

Type

HyperMesh Template Function

Description

Returns the value of the "Export as named entity" export option

Inputs

None.

Examples

Example:

```
*if([@namedentity()])
  // Code for named entity export
*endif()
```

Errors

Incorrect usage results in a Tcl error. To detect errors, you can use the `catch` command:

```
if { [ catch {command_name...} ] } {
```

```
    # Handle error
}
```

Version History

2021

@nlookup()

Retrieves a value stored in the nth lookup table.

Syntax

`@nlookup` *(table, key)*

Type

HyperMesh Template Function

Inputs

table

A value between table1 and table20 indicating which of the 20 possible tables should be manipulated.

key

Used to compare the keys found on the lookup table entries. If *key* matches one of the keys in the lookup table, the function returns the value associated with that entry. If a matching key is not found, the function returns 0.

@partinstancemode()

Returns 1 if the model is a part/instance model.

Syntax

`@partinstancemode` *()*

Type

HyperMesh Template Function

Description

Returns 1 if the model is a part/instance model.

Example

```
*if([@partinstancemode()])
...
*endif()
```

Version History

2020

@pow()

Returns the real value of X raised to the power Y.

Syntax

`@pow` *(x, y)*

Type

HyperMesh Template Function

Inputs

x
: A value of type real.

y
: A value of type real.

Example

```
@pow(2.0, 3.0) = 2.03 =8.0
```

@rangecount()

Returns the number of ranges for the numbers used with `*rangeadd()`.

Syntax

`@rangecount` *()*

Type

HyperMesh Template Function

Example

```
*elements(104,0,"","")
  *format()
    *rangeadd(id)
  *after()
    *counterset(counter1,1)
    *loopif([counter1 <= @rangecount()])
      *string("start of range = ")
      *field(integer,[@rangestart(counter1)],0)
      *end()
      *string("end of range = ")
      *field(integer,[@rangeend(counter1)],0)
      *end()
```

```
      *counterinc(counter1)
    *endloop()
    *rangereset()
*output()
```

Use *rangeadd() to add numbers to a list. Once all numbers have been added, @rangecount() returns the number of ranges (for example 1-5, 10-20) that are in the list. Use the functions @rangestart() and @rangeend() to get the actual ranges.

@rangeend()

Returns the ending range of a range of numbers.

Syntax

`@rangeend` *(range)*

Type

HyperMesh Template Function

Inputs

range
The range number, starting at 1.

Example

```
*elements(104,0,"","")
  *format()
    *rangeadd(id)
  *after()
    *counterset(counter1,1)
    *loopif([counter1 <= @rangecount()])
      *string("start of range = ")
      *field(integer,[@rangestart(counter1)],0)
      *end()
      *string("end of range = ")
      *field(integer,[@rangeend(counter1)],0)
      *end()
      *counterinc(counter1)
    *endloop()
    *rangereset()
*output()
```

Use *rangeadd() to add numbers to a list. Once all numbers have been added, @rangecount() returns how many ranges, such as 1-5, 10-20, are in the list. Use the functions @rangestart() and `@rangeend()` to get the actual ranges.

@rangestart()

Returns the starting range of a range of numbers.

Syntax

`@rangestart` *(range)*

Type

HyperMesh Template Function

Inputs

range
The range number, starting at 1.

Example

```
elements(104,0,"","")
  *format()
    *rangeadd(id)
  *after()
    *counterset(counter1,1)
    *loopif([counter1 <= @rangecount()])
      *string("start of range = ")
      *field(integer,[@rangestart(counter1)],0)
      *end()
      *string("end of range = ")
      *field(integer,[@rangeend(counter1)],0)
      *end()
      *counterinc(counter1)
    *endloop()
    *rangereset()
*output()
```

Use *rangeadd() to add numbers to a list. Once all numbers have been added, @rangecount() will return how many ranges, such as 1-5, 10-20, are in the list. Use the functions `@rangestart()` and @rangeend() to get the actual ranges.

@remsuppressedincludefrommaster()

Returns the value of the "Remove include file reference based on export status" export option.

Syntax

`@remsuppressedincludefrommaster` *()*

Type

HyperMesh Template Function

Description

Returns the value of the "Remove include file reference based on export status" export option, 1 or 0.

Example

```
*if([ @remsuppressedincludefrommaster() == 0 ])
*string("*INCLUDE_PATH")
*end()
*field(string,[@inclstrsplit($FileNameTran,+)],72)
*end()
*endif()
```

Version History

14.0

@repeatkeywordtitles()

Returns the status of the "repeat keyword titles" export option.

Syntax

`@autocreateproperty` *()*

Type

HyperMesh Template Function

Description

Returns 1 if the "repeat keyword titles" export option is enabled via `*feoutputwithdata`, 0 otherwise.

Example

```
*if([@repeatkeywordtitles() == 0])
    *if([counter14 == 0])
        *counterset(counter10,1)
        *counterset(counter14,1)
    *endif()
    *if([counter10 != -1])
        *string("*INTEGRATION_BEAM")
        *end()
        *counterset(counter10,-1)
    *endif()
*endif()

*if([@repeatkeywordtitles() == 1])
    *string("*INTEGRATION_BEAM")
    *end()
*endif()
```

Version History

13.0

@sin()

Trigonometric sine of x, where x is expressed in radians

Syntax

`@sin` *(x)*

Type

HyperMesh Template Function

Description

Trigonometric sine of x, where x is expressed in radians.

Inputs

x
: Value of type real.

Errors

None.

@sqrt()

Returns the square root of a number.

Syntax

`@sqrt` *(x)*

Type

HyperMesh Template Function

Description

Returns the square root of a number.

Example

The square root of z is returned. If x is negative, an error is reported.

Errors

None.

@stringequal()

Compares two strings, and returns 1 if they are equal; otherwise, 0.

Syntax

`@stringequal` *(string1,string2)*

Type

HyperMesh Template Function

Description

Compares two strings, and returns 1 if they are equal; otherwise, 0

Inputs

string 1

string 2

Errors

None.

@stringinstring()

Syntax

`@stringinstring` *(string1, string2, case sensitive flag)*

Type

HyperMesh Template Function

Description

Finds one string within another string.

Inputs

string1
: The string that is searched.

string2
: The string to be found.

case sensitive flag
: If set to 1, the search only succeeds if the case type of the two strings match exactly. For example, "Monday" will not match "monday".

Example

```
*materials("")
```

```
  *format()
    *if([@stringinstring(name,"Elastic",1) == 1])
      *string("Elastic material found: ")
      *field(string,name,0)
      *end()
    *endif()
*output()
```

Returns 1 if the string is found, otherwise, 0.

Errors

None.

@stringlookup()

Retrieves a value stored in a string lookup table.

Syntax

`@stringlookup` *(key)*

Type

HyperMesh Template Function

Description

Retrieves a value stored in a string lookup table.

Inputs

key

Used to compare the keys found in the string lookup table. *key* can be a data name or a literal string enclosed in double quotes.

Example

The following example looks for the string "shells" in the string lookup table.

```
*if([@stringlookup("shells")])
 *string("$ shells found") *end()
*endif()
```

The following example finds components whose names are in the string lookup table:

```
*components("","")
 *format()
  *if([@stringlookup(name)])
   *field(string,name,32) *end()
  *endif()
*output()
```

Errors

None.

@stringsplit()

Splits the given string using the given continuation character.

Syntax

`@stringsplit` *(string,continuation_character)*

Type

HyperMesh Template Function

Description

Splits the given string using the given continuation character. This is used in conjunction with the `*field`, `*fieldleft` or `*fieldright` commands.

Inputs

string
: The string to split.

continuation_character
: The character to use on the continuation line. If set to blank, no continuation character will be used.

Example

To split the string contained in the $eigvfile variable into a group of 60 characters, using no continuation character:

```
*field(string,[@stringsplit($eigvfile,)],60)
```

To split the string contained in the $eigvfile variable into a group of 60 characters, using + as the continuation character:

```
*field(string,[@stringsplit($eigvfile,+)],60)
```

Errors

None.

Version History

11.0.101

@stringstartswithnumericcharacter()

Checks if a string starts with a numeric character.

Syntax

`@stringstartswithnumericcharacter` *(string)*

Type

HyperMesh Template Function

Description

Checks if a string starts with a numeric character.

Inputs

string
The string to check.

Example

```
*components("")
*format()
   *if([@stringstartswithnumericcharacter (name)])
   *string("Component name starts with numeric character: ")
   *field(string,name,0)
   *end()
   *endif()
*output()
```

Returns 1 if the string starts with a numeric character, otherwise, 0.

Errors

None.

@tan()

Trigonometric tangent of x, where x is expressed in radians.

Syntax

`@tan` *(x)*

Type

HyperMesh Template Function

Description

Trigonometric tangent of x, where x is expressed in radians.

Inputs

(x)
A value of type real.

Errors

None.

@vectorlookup()

Retrieves a value stored in a vector lookup table.

Syntax

`@vectorlookup` *(key, x comp, y comp, z comp)*

Type

HyperMesh Template Function

Description

Retrieves a value stored in a vector lookup table.

Inputs

key
Used to compare the keys found on the lookup table entries.

x comp, y comp, z comp
The components of the vector that are used to compare the vectors found on the lookup table entries

Example

If both the keys match, and the vectors are within tolerance, then this function returns the value associated with the matching entry. If a match is not found, the function returns 0.

Errors

None.

@vectorlookupcomponent()

Retrieves a component of the vector stored in a lookup table.

Syntax

`@vectorlookupcomponent` *(comp, key)*

Type

HyperMesh Template Function

Description

Inputs

comp
The component of the vector (1 - x, 2 - y, 3 - z)

key
Used to compare the keys found on the lookup table entries.

Errors

None.

@vectorlookupnotkey()

Retrieves a value stored in a vector lookup table.

Syntax

`@vectorlookupnotkey` *(key,x comp,y comp,z comp)*

Type

HyperMesh Template Function

Description

Retrieves a value stored in a vector lookup table.

Inputs

key
Used to compare the keys found in the lookup table entries.

x comp,y comp,z comp
The components of the vector that is used to compare the vectors found in the lookup table entries.

Example

If the input vector matches the vector in the lookup table and the keys do not match, this function returns the value stored in the lookup table. When looking for a match between vectors, the tolerance set by *vectortablereset() is used. If no match is found, this function returns 0.

Errors

None.

@writehmcomments()

Returns 0 if HMCOMMENTS_SKIP is passed via `*feoutputwithdata`.

Syntax

`@writehmcomments`

Type

HyperMesh Template Function

Description

Returns 0 if HMCOMMENTS_SKIP is passed via `*feoutputwithdata`.

Example

```
*if([@writehmcomments()])
*string("$")
*end()
*string("$  CELAS1 Elements")
*end()
*string("$")
*end()
*endif()
```

Errors

None.

@writenastranoldcontact()

Returns the contact export option value for Nastran (MSC).

Syntax

`@writenastranoldcontact` *()*

Type

HyperMesh Template Function

Description

Returns the contact export option value for Nastran (MSC).

Before 2017.1 for Nastran (MSC), both the "new" (BCBODY1, BCONECT BCTABL1) and "old" (BCBODY, BCTABLE) contacts were mapped to entities in HyperMesh. As a result, users were allowed to create decks with both the version of the solver cards and exported. That is not valid for the solver. Solver either expects a "new" or "old" deck. With 2017.1, users have the option to create "new" version of the solver cards and yet choose to export them as "old" contact cards. A return of 1 means old, and 0 means new.

Example

```
*if([@writenastranoldcontact() == 0])
   ...write new contacts...
*else()
   ...write old contacts...
*endif()
```

Version History

2017.1

@writesolverfieldnames()

Returns 0 if HMSOLVERFILEDNAMES_SKIP is passed via `*feoutputwithdata`.

Syntax

`@writesolverfieldnames` *()*

Type

HyperMesh Template Function

Description

Returns 0 if HMSOLVERFILEDNAMES_SKIP is passed via `*feoutputwithdata`.

Example

```
*if([@writesolverfieldnames()])
*string("$        X          Y          Z")
*end()
*endif()
```

Version History

2020

@xpointelementoffsetvectorlocal()

Transforms a global vector into an elemental system and returns the x value of the transformation.

Syntax

`@xpointelementoffsetvectorlocal` *(element_id,x,y,z)*

Type

HyperMesh Template Function

Description

Transforms a global vector into an elemental system and returns the x value of the transformation.

Inputs

element_id
The ID of the element to use for the transformation.

x y z
The components of the vector to be transformed in the elemental system.

Example

```
*field(real,[@xpointelementoffsetvectorlocal(id,offsetax,offsetay,offsetaz)],8)
*field(real,[@ypointelementoffsetvectorlocal(id,offsetax,offsetay,offsetaz)],8)
*field(real,[@zpointelementoffsetvectorlocal(id,offsetax,offsetay,offsetaz)],8)
```

Errors

None.

Version History

13.0

@xpointlocal()

Transforms a global coordinate into a local system and returns the x value of the transformation.

Syntax

`@xpointlocal` *(system_id,x,y,z)*

Type

HyperMesh Template Function

Description

Transforms a global coordinate into a local system and returns the x value of the transformation.

Inputs

system_id
The ID of the coordinate system to use for the transformation.

x y z
The coordinates of the point to be transformed in the local system.

Example

```
*field(real,
[@xpointlocal(inputsystemid,globaloriginx,globaloriginy,globaloriginz)],8)
*field(real,
[@ypointlocal(inputsystemid,globaloriginx,globaloriginy,globaloriginz)],8)
*field(real,[90.0 -
 @zpointlocal(inputsystemid,globaloriginx,globaloriginy,globaloriginz)],8)
```

Errors

None.

@xpointvectorlocal()

Transforms a vector into a local system and returns the x value of the transformed vector.

Syntax

`@xpointvectorlocal` *(system id, x, y, z, vx, vy, vz)*

Type

HyperMesh Template Function

Description

Transforms a vector into a local system and returns the x value of the transformed vector.

Inputs

system i
: The ID of the system into which the point should be transformed.

x, y, z
: The coordinates of the point where the vector is located in the global system.

vx, vy, vz
: The components of the vector to be transformed.

Errors

None.

@ypointelementoffsetvectorlocal()

Transforms a global vector into an elemental system and returns the y value of the transformation.

Syntax

`@ypointelementoffsetvectorlocal` *(element_id,x,y,z)*

Type

HyperMesh Template Function

Description

Transforms a global vector into an elemental system and returns the y value of the transformation.

Inputs

element_id
The ID of the element to use for the transformation.

x y z
The components of the vector to be transformed in the elemental system.

Example

```
*field(real,[@xpointelementoffsetvectorlocal(id,offsetax,offsetay,offsetaz)],8)
*field(real,[@ypointelementoffsetvectorlocal(id,offsetax,offsetay,offsetaz)],8)
*field(real,[@zpointelementoffsetvectorlocal(id,offsetax,offsetay,offsetaz)],8)
```

Errors

None.

Version History

13.0

@ypointlocal()

Transforms a global coordinate into a local system and returns the y value of the transformation.

Syntax

`@ypointlocal` *(system_id,x,y,z)*

Type

HyperMesh Template Function

Description

Transforms a global coordinate into a local system and returns the y value of the transformation.

Inputs

system_id
The ID of the coordinate system to use for the transformation.

x y z
The coordinates of the point to be transformed in the local system.

Example

```
*field(real,
[@xpointlocal(inputsystemid,globaloriginx,globaloriginy,globaloriginz)],8)
*field(real,
[@ypointlocal(inputsystemid,globaloriginx,globaloriginy,globaloriginz)],8)
*field(real,[90.0 -
 @zpointlocal(inputsystemid,globaloriginx,globaloriginy,globaloriginz)],8)
```

Errors

None.

@ypointvectorlocal()

Transforms a vector into a local system and returns the y value of the transformed vector

Syntax

`@ypointvectorlocal` *(system id, x, y, z, vx, vy, vz)*

Type

HyperMesh Template Function

Description

Transforms a vector into a local system and returns the y value of the transformed vector

Inputs

system id
: The ID of the system into which the point should be transformed.

x, y, z
: The coordinates of the point where the vector is located in the global system.

vx, vy, vz
: The components of the vector to be transformed.

Errors

None.

@zpointelementoffsetvectorlocal()

Transforms a global vector into an elemental system and returns the z value of the transformation.

Syntax

`@zpointelementoffsetvectorlocal` *(element_id,x,y,z)*

Type

HyperMesh Template Function

Description

Transforms a global vector into an elemental system and returns the z value of the transformation.

Inputs

element_id
The ID of the element to use for the transformation.

x y z
The components of the vector to be transformed in the elemental system.

Example

```
*field(real,[@xpointelementoffsetvectorlocal(id,offsetax,offsetay,offsetaz)],8)
*field(real,[@ypointelementoffsetvectorlocal(id,offsetax,offsetay,offsetaz)],8)
*field(real,[@zpointelementoffsetvectorlocal(id,offsetax,offsetay,offsetaz)],8)
```

Errors

None.

Version History

13.0

@zpointlocal()

Transforms a global coordinate into a local system and returns the z value of the transformation.

Syntax

`@zpointlocal` *(system_id,x,y,z)*

Type

HyperMesh Template Function

Description

Transforms a global coordinate into a local system and returns the z value of the transformation.

Inputs

system_id
The ID of the coordinate system to use for the transformation.

x y z
The coordinates of the point to be transformed in the local system.

Example

```
*field(real,
[@xpointlocal(inputsystemid,globaloriginx,globaloriginy,globaloriginz)],8)
*field(real,
[@ypointlocal(inputsystemid,globaloriginx,globaloriginy,globaloriginz)],8)
*field(real,[90.0 -
 @zpointlocal(inputsystemid,globaloriginx,globaloriginy,globaloriginz)],8)
```

Errors

None.

@zpointvectorlocal()

Transforms a vector into a local system and returns the z value of the transformed vector.

Syntax

`@zpointvectorlocal` *(system id, x, y, z, vx, vy, vz)*

Type

HyperMesh Template Function

Description

Transforms a vector into a local system and returns the z value of the transformed vector.

Inputs

system id
: The ID of the system into which the point should be transformed.

x, y, z
: The coordinates of the point where the vector is located in the global system.

vx, vy, vz
: The components of the vector to be transformed.

Errors

None.

Undocumented Solver Template Functions

The list of undocumented solver template functions.

@attributearrayvaluewithoffset()

@attributeindexentitypoolid()

@componentfornode()

@entityexists()

@entitymaxsolverid()

@getabsolutename()

@getassemfromcomp()

@getentityclausevalue()

@getentitycount()

@getentitytype()

@getexporttype()

@inclstrsplit()

@indexbelongstoddvalrange()

@strlen()

@xcomppreslocal1d()

@ycomppreslocal1d()

@zcomppreslocal1d()

Index

Special Characters

A

C

D

E

F

G

M

N

O

P