# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>COREscript Constructs</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Control Constructs</strong></td>
<td>1</td>
</tr>
<tr>
<td>Block Construct</td>
<td>1</td>
</tr>
<tr>
<td>Disabled Code Construct</td>
<td>2</td>
</tr>
<tr>
<td>Hierarchy Iterate Construct</td>
<td>3</td>
</tr>
<tr>
<td>If Then Construct</td>
<td>5</td>
</tr>
<tr>
<td>If Then Else Construct</td>
<td>6</td>
</tr>
<tr>
<td>Include Script</td>
<td>7</td>
</tr>
<tr>
<td>Iterate Construct</td>
<td>8</td>
</tr>
<tr>
<td>Repeat Until Construct</td>
<td>9</td>
</tr>
<tr>
<td>Return</td>
<td>10</td>
</tr>
<tr>
<td>Update Block Construct</td>
<td>11</td>
</tr>
<tr>
<td>While Do Construct</td>
<td>12</td>
</tr>
<tr>
<td><strong>Database Query Constructs</strong></td>
<td>13</td>
</tr>
<tr>
<td>Children Query</td>
<td>13</td>
</tr>
<tr>
<td>Diagnostic Results Query</td>
<td>14</td>
</tr>
<tr>
<td>Element Attribute Query</td>
<td>15</td>
</tr>
<tr>
<td>Element Defined Parameters Query</td>
<td>16</td>
</tr>
<tr>
<td>Element Parameter Units Query</td>
<td>17</td>
</tr>
<tr>
<td>Element Parameter Value Query</td>
<td>18</td>
</tr>
<tr>
<td>Elements Query</td>
<td>19</td>
</tr>
<tr>
<td>Named Element Query</td>
<td>20</td>
</tr>
<tr>
<td>Parents Query</td>
<td>21</td>
</tr>
<tr>
<td>Relationship Attribute Query</td>
<td>22</td>
</tr>
<tr>
<td>Relationships Query</td>
<td>23</td>
</tr>
<tr>
<td>Root Folder Query</td>
<td>24</td>
</tr>
<tr>
<td>Targets Query</td>
<td>25</td>
</tr>
<tr>
<td><strong>Database Update Constructs</strong></td>
<td>27</td>
</tr>
<tr>
<td>Add Element To Folder</td>
<td>27</td>
</tr>
<tr>
<td>Delete Object</td>
<td>28</td>
</tr>
<tr>
<td>Lock Object</td>
<td>29</td>
</tr>
<tr>
<td>Move Element To Folder</td>
<td>30</td>
</tr>
<tr>
<td>New Element</td>
<td>31</td>
</tr>
<tr>
<td>New Folder</td>
<td>32</td>
</tr>
<tr>
<td>New Relationship</td>
<td>33</td>
</tr>
<tr>
<td>Purge Attribute Versions</td>
<td>34</td>
</tr>
<tr>
<td>Remove Element From Folder</td>
<td>35</td>
</tr>
<tr>
<td>Rename Element</td>
<td>36</td>
</tr>
<tr>
<td>Rename Folder</td>
<td>37</td>
</tr>
<tr>
<td>Renumber Elements</td>
<td>38</td>
</tr>
<tr>
<td>Set Element Attribute</td>
<td>39</td>
</tr>
<tr>
<td>Set Element Diagnostic Results</td>
<td>40</td>
</tr>
<tr>
<td>Set Element Parameter Units</td>
<td>41</td>
</tr>
<tr>
<td>Set Element Parameter Value</td>
<td>42</td>
</tr>
<tr>
<td>Set Relationship Attribute</td>
<td>43</td>
</tr>
<tr>
<td>Unlock Object</td>
<td>44</td>
</tr>
<tr>
<td><strong>File Management Constructs</strong></td>
<td>45</td>
</tr>
<tr>
<td>Close File</td>
<td>45</td>
</tr>
<tr>
<td>Open File Dialog</td>
<td>46</td>
</tr>
<tr>
<td>Open Input File</td>
<td>47</td>
</tr>
<tr>
<td>Open Output File</td>
<td>48</td>
</tr>
<tr>
<td>Open Temporary Output File</td>
<td>49</td>
</tr>
<tr>
<td>Save File Dialog</td>
<td>50</td>
</tr>
<tr>
<td>Set Default Input File</td>
<td>51</td>
</tr>
<tr>
<td>Set Default Output File</td>
<td>52</td>
</tr>
</tbody>
</table>
COREscript Constructs

Output Constructs ............................................................................................................. 53
Comma-Delimited Output ................................................................................................. 53
Diagram File Output ......................................................................................................... 54
Diagram Output ................................................................................................................ 57
Element Output ................................................................................................................ 60
Include External Boilerplate ............................................................................................. 63
Include External Graphic ................................................................................................. 64
Include Style Sheet .......................................................................................................... 65
RDT Output ....................................................................................................................... 66
Set Style ........................................................................................................................... 67
Table Output ..................................................................................................................... 68
Tag Output ......................................................................................................................... 70
Text Output ....................................................................................................................... 71
Variable Output ................................................................................................................. 72
Parsing Constructs ............................................................................................................ 73
At End ................................................................................................................................. 73
Get Position ....................................................................................................................... 74
Next .................................................................................................................................. 75
Peek .................................................................................................................................. 76
Set Position ....................................................................................................................... 77
Size ................................................................................................................................... 78
Skip ................................................................................................................................... 79
Up To ................................................................................................................................ 80
Project Query Constructs ................................................................................................. 81
Project Property ................................................................................................................ 81
Project Query .................................................................................................................... 83
Project Registry Query ..................................................................................................... 84
Projects ............................................................................................................................... 85
Remove Project Registry Key ........................................................................................... 86
Set Project .......................................................................................................................... 87
Set Project Registry Value ............................................................................................... 88
Prompting Constructs ........................................................................................................ 89
Composite Prompt ............................................................................................................ 89
Drop-Down Selection Prompt ............................................................................................ 90
Element Selection Prompt ............................................................................................... 91
Message Prompt ............................................................................................................... 93
Multiple Selection Prompt ............................................................................................... 94
Project Explorer Prompt .................................................................................................. 96
Prompt Separator ............................................................................................................. 97
Report Settings Prompt ................................................................................................... 98
Single Selection Prompt ................................................................................................. 99
Value Prompt .................................................................................................................... 100
YesOrNo Prompt .............................................................................................................. 101
Schema Query Constructs ............................................................................................... 103
Attribute Definition Property ......................................................................................... 103
Class Attributes ............................................................................................................... 104
Class Parameters ............................................................................................................. 105
Class Property ................................................................................................................. 106
Class Query ..................................................................................................................... 107
Class Relations ............................................................................................................... 108
Class Target Classes ...................................................................................................... 109
Database Classes ............................................................................................................. 110
Facilities ............................................................................................................................ 111
Facility Database Classes ............................................................................................... 112
Facility Property .............................................................................................................. 113
Facility Query .................................................................................................................. 114
<table>
<thead>
<tr>
<th>Corescript Constructs</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facility Schema Classes</td>
<td>115</td>
</tr>
<tr>
<td>Parameter Definition Property</td>
<td>116</td>
</tr>
<tr>
<td>Relation Attributes</td>
<td>117</td>
</tr>
<tr>
<td>Relation Property</td>
<td>118</td>
</tr>
<tr>
<td>Relation Query</td>
<td>119</td>
</tr>
<tr>
<td>Relations</td>
<td>120</td>
</tr>
<tr>
<td>Schema Classes</td>
<td>121</td>
</tr>
<tr>
<td>Simulation Constructs</td>
<td>123</td>
</tr>
<tr>
<td>Resource Level Query</td>
<td>123</td>
</tr>
<tr>
<td>Script Context Query</td>
<td>124</td>
</tr>
<tr>
<td>Simulator Property Query</td>
<td>125</td>
</tr>
<tr>
<td>Utilities Constructs</td>
<td>127</td>
</tr>
<tr>
<td>Available Diagrams Query</td>
<td>127</td>
</tr>
<tr>
<td>Beep</td>
<td>128</td>
</tr>
<tr>
<td>Breakpoint</td>
<td>129</td>
</tr>
<tr>
<td>Cancel Report</td>
<td>130</td>
</tr>
<tr>
<td>Change Indicator</td>
<td>131</td>
</tr>
<tr>
<td>Color Query</td>
<td>132</td>
</tr>
<tr>
<td>Comment</td>
<td>133</td>
</tr>
<tr>
<td>Expire Output</td>
<td>134</td>
</tr>
<tr>
<td>Open Transcript</td>
<td>135</td>
</tr>
<tr>
<td>Send Mail Message</td>
<td>136</td>
</tr>
<tr>
<td>System Property Query</td>
<td>138</td>
</tr>
<tr>
<td>Variable Manipulation Constructs</td>
<td>141</td>
</tr>
<tr>
<td>Database Object Sort</td>
<td>141</td>
</tr>
<tr>
<td>Element Sort</td>
<td>143</td>
</tr>
<tr>
<td>Element Sort Blocks</td>
<td>144</td>
</tr>
<tr>
<td>Filter</td>
<td>145</td>
</tr>
<tr>
<td>Random Number</td>
<td>147</td>
</tr>
<tr>
<td>Report Section</td>
<td>148</td>
</tr>
<tr>
<td>Report Setting</td>
<td>149</td>
</tr>
<tr>
<td>Sort</td>
<td>150</td>
</tr>
<tr>
<td>Table Definitions</td>
<td>151</td>
</tr>
<tr>
<td>Variable Assignment</td>
<td>152</td>
</tr>
<tr>
<td>Construct Index</td>
<td>155</td>
</tr>
</tbody>
</table>
Supporting users throughout their entire journey of learning MBSE is central to Vitech’s mission. For users looking for additional resources outside of this document, please refer to the links below. Alternatively, all links may be found at www.vitechcorp.com/resources.

**Webinars**
Webinar archive with over 40 hours of premium industry and tool-specific content.

**Screencasts**
Short videos to guide users through installation and usage of Vitech software.

**A Primer for Model-Based Systems Engineering**
Our free eBook and our most popular resource for new and experienced practitioners alike.

**Help Files**
Searchable online access to Vitech software help files.

**Technical Papers**
Library of technical and white papers for download, authored by Vitech systems engineers.

**MySupport**
Knowledge Base, Exclusive Webinars and Screencasts, Chat Support, Documents, Download Archive, etc.

Our team has also created resources libraries customized for your experience level:

- All Resources
- Advanced
- Beginner
- IT / Sys Admin
- Intermediate
- Student
CONTROL CONSTRUCTS

Block Construct

Description
This construct consists of two parts on different lines of the Report Editor’s Script pane that may enclose other constructs indented between them. A block is simply an artificial, non-functional construct for the purpose of promoting code decomposability, composability, reusability, and documentation. It encloses code that can then be copied and moved as a unit, and it breaks up the flow of the script into more manageable units.

Comment
An explanation of what is accomplished by the constructs enclosed within the block, block assumptions, etc.
Disabled Code Construct

Description
This construct consists of two parts on different lines of the Report Editor’s Script pane that may enclose other constructs indented between them. The enclosed constructs are not executed.

Comment
An explanation of why the enclosed constructs have been disabled.
COREscript Constructs
Control

Hierarchy Iterate Construct

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Relations:</td>
<td>nil</td>
</tr>
<tr>
<td>Target Classes:</td>
<td>nil</td>
</tr>
<tr>
<td>Sort Block:</td>
<td>nil</td>
</tr>
<tr>
<td>Maximum Levels:</td>
<td>nil</td>
</tr>
<tr>
<td>Iteration Variable:</td>
<td></td>
</tr>
<tr>
<td>Counter Variable:</td>
<td></td>
</tr>
<tr>
<td>Level Variable:</td>
<td></td>
</tr>
<tr>
<td>Relation Variable:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**
Given a starting point (the root) and a collection of valid relations and target classes, the Hierarchy Iterate Construct generates a tree of elements and iterates across the tree. The tree is expanded depth first (left to right) and uses a pre-ordered traversal. This construct is recursion-safe – the second time an element is encountered, it is reported but its child nodes are not traversed.

**Object**
An element specifying the root for the tree. This is the starting point from which the hierarchy will be built based upon the specified relations and target classes.

**Relations**
The collection of valid relations to use when building the hierarchy for iteration. This is equivalent to the set of relations you would specify if building a custom hierarchy view.

**Target Classes**
The collection of valid target classes to use when building the hierarchy for iteration. Only those targets within the classes specified will appear in the hierarchy. For example, if you wished to iterate through a Requirement hierarchy and all Functions that are specified by the Requirement hierarchy, specifying Requirement and Function as the valid target classes would prevent any other classes (e.g., Components) from appearing within the iteration.

**Sort Block**
Sort block definition of the sort block to be used to order the children of a node within the hierarchy. Child nodes are ordered first by relation and then by the criteria in the specified sort block.

**Maximum Levels**
An integer specifying the maximum depth of the tree. Nodes will be expanded until either the leaf-level nodes have no children (as defined by the set of valid relations and target classes) or the leaf nodes are at the maximum depth. The root of the tree is defined as level 1.
COREscript Constructs
Control

**Iteration Variable**
The variable which will hold the current element during the iteration.

**Counter Variable**
The variable, which will hold the total number of elements, iterated over so far.

**Level Variable**
The variable which will hold the level of the current element within the hierarchy (where the root of the tree is defined as level 1).

**Relation Variable**
The variable, which will hold the relation, used to reach the current element (the relation from the parent node to the child node).
COREscript Constructs
Control

If Then Construct

Description
This construct consists of two parts on different lines of the Report Editor’s Script pane that may enclose other constructs indented between them. The enclosed constructs are executed if and only if expression evaluates to true.

Expression
An expression consisting of operators in the CORE expression language, variables defined using the Edit Variables command under the Properties menu of the Report Editor, and constants that can be coded in some data types. The result of this expression must be a Boolean.

Validate
Validates the syntax of the expression to ensure that neither operators nor operands appear in unexpected locations, all operators are recognized, and all variables have been defined. No type or bounds checking can be performed.
If Then Else Construct

Description
This construct consists of three parts on different lines of the Report Editor’s Script pane that may enclose two groups of other constructs indented between them. This construct implements branch logic. The first enclosed group of constructs (the then branch) is executed if expression evaluates to true. The second enclosed group of constructs (the else branch) is executed if expression evaluates to false.

Expression
An expression consisting of operators in the CORE expression language, variables defined using the Edit Variables command under the Properties menu of the Report Editor, and constants that can be coded in some data types. The result of this expression must be a Boolean.

Validate
Validates the syntax of the expression to ensure that neither operators nor operands appear in unexpected locations, all operators are recognized, and all variables have been defined. No type or bounds checking can be performed.
Include Script

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Script</strong></td>
<td>Transfers control to the specified script file, which is executed in its entirety before control returns to the next construct in the current script (much like a subroutine or procedure call). The invoked script and any scripts it includes have access to global variables declared in the current report execution. Any duplicate variable declarations in the included script reference variables already defined in the current report execution. Note that double-clicking the left mouse button with the pointer over an Include Script construct in the Script pane of a Report Editor opens a new Report Editor containing the script file included by the construct, if any.</td>
</tr>
<tr>
<td><strong>Location</strong></td>
<td>The name of the script file to be executed. Can be specified either by selecting a variable from the drop-down on the left, manually entering as a parameter, or selecting the file using a standard Windows file dialog (via the Script Browse Button). If a manually entered file name does not currently exist the parameter field background will change to yellow.</td>
</tr>
<tr>
<td><strong>Script Arguments</strong></td>
<td>The ordered collection of variables whose values will be passed to the invoked script as arguments. If the number of variables does match exactly with the number of parameters declared in the script properties portion of the invoked script, an error will result. Any values assigned to the parameters from within the invoked script will not affect the corresponding arguments in the current script.</td>
</tr>
<tr>
<td><strong>Return</strong></td>
<td>Variable to hold the result, if any, from executing the include script. In order for an include script to return a value, it must complete with a Return construct that has an argument.</td>
</tr>
</tbody>
</table>

**Description**

Transfers control to the specified script file, which is executed in its entirety before control returns to the next construct in the current script (much like a subroutine or procedure call). The invoked script and any scripts it includes have access to global variables declared in the current report execution. Any duplicate variable declarations in the included script reference variables already defined in the current report execution. Note that double-clicking the left mouse button with the pointer over an Include Script construct in the Script pane of a Report Editor opens a new Report Editor containing the script file included by the construct, if any.

**Script**

The name of the script file to be executed. Can be specified either by selecting a variable from the drop-down on the left, manually entering as a parameter, or selecting the file using a standard Windows file dialog (via the Script Browse Button). If a manually entered file name does not currently exist the parameter field background will change to yellow.

**Script Browse Button**

The Script browse button opens a standard Windows file dialog. The user must specify a script file in the location directory as described below or in a subdirectory thereof. Canceling the dialog leaves the previous selection unchanged.

**Location**

The CORE directory where the script is located. When set to main, common, or user, the construct uses the settings established in the Report Directories portion of the CORE user preferences to locate the included script file in the appropriate directory.

**Script Arguments**

The ordered collection of variables whose values will be passed to the invoked script as arguments. If the number of variables does match exactly with the number of parameters declared in the script properties portion of the invoked script, an error will result. Any values assigned to the parameters from within the invoked script will not affect the corresponding arguments in the current script.

**Return**

Variable to hold the result, if any, from executing the include script. In order for an include script to return a value, it must complete with a Return construct that has an argument.
COREscript Constructs
Control

Iterate Construct

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection:</td>
<td>nil</td>
</tr>
<tr>
<td>Iteration Variable:</td>
<td></td>
</tr>
<tr>
<td>Counter Variable:</td>
<td></td>
</tr>
</tbody>
</table>

Description
This construct consists of two parts on different lines of the Report Editor’s Script pane that may enclose other constructs indented between them. The enclosed constructs are executed once for each object in the collection.

On each iteration, a different object in the collection is assigned to the iteration variable if one is defined. If the collection is an array or ordered collection, this construct iterates over the objects in their ordered defined by the collection. If the collection is a string or symbol, this construct iterates over its characters in order.

On the first iteration, the counter variable is assigned the value 1. On every subsequent iteration, its value is incremented by 1.

Collection
Name of a variable containing the collection over which to iterate, which can be of one of the following data types:

- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- String
- Symbol

If the collection is a dictionary, this construct iterates over the values stored in the dictionary, not the keys whereby the values are accessed.

Iteration Variable
Name of the iteration variable. This variable is declared within the scope of this construct but will not be accessible after this construct terminates.

Counter Variable
Name of the counter variable. This variable is declared within the scope of this construct but will not be accessible after this construct terminates.
Repeat Until Construct

**Description**
This construct consists of two parts on different lines of the Report Editor’s Script pane that may enclose other constructs indented between them. This construct implements loop logic. After each time the group of enclosed constructs is executed, the expression is tested and, if it does not evaluate to `true`, control flows back to the beginning. Note that the enclosed constructs are guaranteed to be executed at least once, since the test is performed at the end.

**Expression**
An expression consisting of operators in the CORE expression language, variables defined using the Edit Variables command under the Properties menu of the Report Editor, and constants that can be coded in some data types. The result of this expression must be a Boolean.

**Validate**
Validates the syntax of the expression to ensure that neither operators nor operands appear in unexpected locations, all operators are recognized, and all variables have been defined. No type or bounds checking can be performed.
**Return**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Terminates execution of an include script returning control to the calling script. If the Return Construct is encountered in the top-level script, the report is exited.

**Object**

Variable that contains the object to be returned from the script.
Update Block Construct

**Description**
This construct consists of three parts on different lines of the Report Editor’s Script pane that may enclose two groups of other constructs indented between them. This construct wraps database updates with an error handler to process errors encountered due to invalid update attempts. The first enclosed group is the block of constructs to be executed. If a database update error is encountered, the second enclosed group of constructs (the *error handler*) is executed.

**Comment**
An explanation of what is accomplished by the constructs enclosed within the block, block assumptions, etc.

**Error Message**
Variable to hold the error message if an error is encountered within the protected code of this construct.
While Do Construct

Expression

Validate

Description
This construct consists of two parts on different lines of the Report Editor’s Script pane that may enclose other constructs indented between them. This construct implements loop logic. If the expression evaluates to true, the enclosed group of constructs is executed repeatedly until the expression evaluates to false. Note that the enclosed constructs are not guaranteed to be executed, since the expression is tested before any of them are executed.

Expression
An expression consisting of operators in the CORE expression language, variables defined using the Edit Variables command under the Properties menu of the Report Editor, and constants that can be coded in some data types. The result of this expression must be a Boolean.

Validate
Validates the syntax of the expression to ensure that neither operators nor operands appear in unexpected locations, all operators are recognized, and all variables have been defined. No type or bounds checking can be performed.
DATABASE QUERY CONSTRUCTS

Children Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Target Classes:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Add   Remove

Sort Block:  nil   | Numeric by class |
Return:      nil   |

Description

Assigns to the return variable the collection of all the elements in the CORE database that are targets of parent-child relationships with the receiver element. The parent-child relationships are defined by the parent-child relation established in the CORE schema for the class of the receiver element. There may be no parent-child relation for the class of the receiver. The parent-child relation is used to do attribute value roll-ups, where an attribute value in one element is computed from the corresponding attribute values in its children.

The children elements may not all be of the same class. To filter out all children except those in one or more particular classes, specify the target classes collection. The returned collection will be ordered according to the specified sort block.

Object

Variable containing the element to query.

Target Classes

Variable containing a collection of filter classes. Can be specified either by a variable from the drop-down on the left or manual additions into the collection on the right. Can be of any of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection

Sort Block

Sort block definition of a sort block defined in the CORE project. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. This sort criteria continues to govern the behavior of the sorted collection that results from this construct.

Return

Variable to hold the sorted collection of elements that results from this construct.
Diagnostic Results Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Diagnostic:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Assigns to the `return` variable a collection of strings representing the diagnostic errors for the specified element. The returned collection will be a set of strings.

**Object**

Variable containing the element to query.

**Diagnostic**

Symbol specifying which of the three diagnostic results is desired: completeness, consistency, or custom. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**

Variable to hold the ordered collection of strings that is the result of this construct.
COREscript Constructs
Database Query

**Element Attribute Query**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Attribute:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the `return` variable the value of the specified `attribute` of the element in the CORE database specified by `object`. Returns `nil` if either the attribute has not been assigned a value or the user does not have sufficient permissions to view the value.

**Object**
Variable containing the element from the CORE database to query.

**Attribute**
Symbol naming the attribute which value is desired. The attribute should be defined in the CORE schema for the class of the `object`. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**
Variable to hold the result of this construct.
## Element Defined Parameters Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

### Description
Assigns to the `return` variable a collection of the names of all parameters defined for the specified element. This is the collection of parameters that have been defined for the element (with or without values), not the collection of available parameters for the class. The returned collection will be a set of symbols.

### Object
Variable containing the element to query.

### Return
Variable to hold the set of parameter name symbols that is the result of this construct.
**Element Parameter Units Query**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object:</strong></td>
<td>nil ▼</td>
</tr>
<tr>
<td><strong>Parameter:</strong></td>
<td>nil ▼</td>
</tr>
<tr>
<td><strong>Return:</strong></td>
<td>nil ▼</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the *return* variable the units string associated with the specified parameter for the specified element. For example, in the case of a parameter such as mass, the units might be kilograms. The construct will return nil if the parameter has not been defined for this element or the specified parameter does not support a units field.

**Object**
Variable containing the element to query.

**Parameter**
Symbol naming the parameter to be queried. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**
Variable to hold the string that is the result of this construct.
**Element Parameter Value Query**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Parameter:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the *return* variable the value associated with the specified parameter for the specified element. The construct will return nil if the parameter has not been defined for this element.

**Object**
Variable containing the element to query.

**Parameter**
Symbol naming the parameter to be queried. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**
Variable to hold the result of this construct.
Elements Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Class:</td>
<td>nil</td>
</tr>
<tr>
<td>Sort Block:</td>
<td>nil, Numeric</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the *return* variable a collection of all the elements in the CORE database that are of the specified *class*. The returned collection will be ordered according to the specified *sort block*.

Database Class
Symbol naming a class defined in the CORE schema. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Sort Block
Sort block definition of a sort block defined in the CORE project. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. This sort criteria continues to govern the behavior of the sorted collection that results from this construct.

Return
Variable to hold the sorted collection of elements that is the result of this construct.
Named Element Query

Description
Assigns to the return variable the element in the CORE database that has the specified name and is of the specified class. The construct will return nil if either the requested element is not in the database or the user does not have sufficient permissions to access the element.

Database Class
A class object defined in the CORE schema. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Element Name
Symbol naming the element to be returned. Can be specified either by a variable from the drop-down on the left or a manual selection using an element selection window opened by pressing the Select Element button.

Return
Variable to hold the result of the construct.
COREscript Constructs
Database Query

Parents Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object: nil</td>
<td></td>
</tr>
<tr>
<td>Target Classes: nil</td>
<td></td>
</tr>
<tr>
<td>Sort Block: nil</td>
<td>Numeric by class</td>
</tr>
<tr>
<td>Return: nil</td>
<td></td>
</tr>
</tbody>
</table>

Description
Assigns to the return variable the collection of all the elements in the CORE database that are sources (vice targets) of parent-child relationships with the receiver element (i.e., the receiver is the target of the relationships). The parent-child relationships are defined by parent-child relations established in the CORE schema with the class of the receiver element as the target class. The parent-child relation is used to do attribute value roll-ups, where attribute values of a child are used to compute the corresponding attribute values of its parents.

There is at most a single parent-child relation established for each class; however, since this operator views the relations in the opposite direction (i.e., their complements from child to parent), the class of the receiver may be a target class of multiple parent-child relations, each with a different source class. To filter out all parents except those in one or more particular classes, specify the target classes collection. The returned collection will be ordered according to the specified sort block.

Object
Variable containing the element to query.

Target Classes
Variable containing a collection of filter classes. Can be specified either by a variable from the drop-down on the left or manual additions into the collection on the right. Can be of any of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection

Sort Block
Sort block definition of a sort block defined in the CORE project. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. This sort criteria continues to govern the behavior of the sorted collection that results from this construct.

Return
Variable to hold the sorted collection of elements that results from this construct.
Relationship Attribute Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Attribute:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Assigns to the `return` variable the value of the specified `attribute` of the relationship in the CORE database specified by `object`.

**Object**

Variable containing the relationship from the CORE database to query.

**Attribute**

Symbol naming the attribute which value is desired. The attribute should be defined in the CORE schema for the relation of the `object`. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**

Variable to hold the result of this construct.
COREscript Constructs
Database Query

Relationships Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Relation:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Target Classes:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Return:</strong></td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the *return* variable a collection of all the relationships in the CORE database that are of the specified *relation* and have the element specified by *object* as their source. To filter out all relationships except those, which targets are in one or more particular classes, specify the *target classes* collection.

**Object**
Variable containing the element from the CORE database to query.

**Relation**
Symbol naming a relation defined in the CORE schema. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Target Classes**
Variable containing a collection of filter classes. Can be specified either by a variable from the drop-down on the left or manual additions into the collection on the right. Can be of any of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection

**Return**
Variable to hold the set of relationships that results from this construct.
COREscript Constructs
Database Query

Root Folder Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Class:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the return variable the element folder object for the specified class. Each class has a root-level folder whose name is the name of the class.

Database Class
A class object defined in the CORE schema. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Return
Variable to hold the result from this construct.
Targets Query

### Arguments
- **Object:** nil
- **Relation:** nil
- **Target Classes:** nil

### Parameters
- **Sort Block:** nil
- **Return:** nil

**Description**
Assigns to the `return` variable a collection of all the elements in the CORE database that are of the specified `relation` and have the element specified by `object` as their source. The returned collection will be ordered according to the specified `sort block`. To filter out all targets except those in one or more particular classes, specify the `target classes` collection.

**Object**
Variable containing the element from the CORE database to query.

**Relation**
Symbol naming a relation defined in the CORE schema. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Target Classes**
Variable containing a collection of filter classes. Can be specified either by a variable from the drop-down on the left or manual additions into the collection on the right. Can be of any of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection

**Sort Block**
Sort block definition of a sort block defined in the CORE project. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. This sort criteria continues to govern the behavior of the sorted collection that results from this construct.

**Return**
Variable to hold the sorted collection of elements that results from this construct.
DATABASE UPDATE CONSTRUCTS

Add Element To Folder

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element:</td>
<td>nil</td>
</tr>
<tr>
<td>Folder:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Adds the specified *element* to the *folder* without removing the *element* from any other folder. If the user does not have write permission for the *element*, an error will be signaled.

**Element**

Variable containing the element to add to the folder.

**Folder**

Variable containing the folder to which the element should be added.
Delete Object

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Deletes the specified object (either an element, relationship, or folder) from the database. If the user does not have permission to delete the object, an error will be signaled.

**Object**
Variable containing the object (an element, relationship or folder) to be deleted.
Lock Object

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description

Locks the specified object (either an element, attribute, or parameter) against any changes regardless of the user permission. If the user does not have permission to administer the object, an error will be signaled.

Object

Variable containing the object (an element, attribute, or parameter) to be locked.
Move Element To Folder

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element:</td>
<td>nil</td>
</tr>
<tr>
<td>Source Folder:</td>
<td>nil</td>
</tr>
<tr>
<td>Target Folder:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description

Removes the specified *element* from the *source folder* and adds it to the *target folder*. Any other folders containing the *element* are unaffected. Note that if the source folder does not contain the specified *element*, no error will be signaled and the element will be added to the *target folder*.

If the user does not have write permission for the element, an error will be signaled.

**Element**

Variable containing the element to move.

**Source Folder**

Variable containing the folder from which to remove the *element*.

**Target Folder**

Variable containing the folder to which the *element* should be added.
New Element

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Class:</td>
<td>nil</td>
</tr>
<tr>
<td>Element Name:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Creates a new element of the specified database class with the specified element name and assigns it to the return variable. An error is signaled and nil is returned if the element cannot be created due to a name conflict or insufficient user permissions to create elements.

**Database Class**

Symbol naming a class defined in the CORE schema. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Element Name**

Symbol naming the element to be created. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Return**

Variable to hold the result of this construct – the new element if the operation is successful or nil if it is not.
New Folder

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent Folder:</td>
<td>nil</td>
</tr>
<tr>
<td>Folder Name:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Creates a new folder with the specified *folder name* as a subfolder of *parent folder* and assigns it to the *return* variable. An error is signaled and *nil* is returned if the folder cannot be created due to a name conflict or insufficient user permissions to create folders.

**Parent Folder**

Variable containing the folder in which the new subfolder is to be created.

**Folder Name**

Symbol naming the folder to be created. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Return**

Variable to hold the result of this construct – the new folder if the operation is successful or *nil* if it is not.
New Relationship

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Element:</td>
<td>nil</td>
</tr>
<tr>
<td>Relation:</td>
<td>nil</td>
</tr>
<tr>
<td>Target Element:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description

Creates a new relationship of type relation between the specified elements and assigns it to the return variable. An error is signaled and nil is returned if the relationship cannot be created due to a schema conflict, a database constraint, or insufficient user permissions to create the relationship.

Source Element

Variable containing the source element of the new relationship. This is equivalent to the element selected in CORE when selecting a relation name and issuing the “Add Targets” command.

Relation

Relation defined in the CORE schema. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Target Element

Variable containing the target of the new relationship. This is equivalent to the element specified in the “Add Targets” dialog in CORE.

Return

Variable to hold the result of this construct – the new relationship if the operation is successful or nil if it is not.
Purge Attribute Versions

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elements</td>
<td>nil</td>
</tr>
<tr>
<td>Attributes</td>
<td>nil</td>
</tr>
<tr>
<td>Type</td>
<td>nil</td>
</tr>
<tr>
<td></td>
<td>Interim Versions</td>
</tr>
<tr>
<td></td>
<td>All Versions</td>
</tr>
</tbody>
</table>

**Description**

Purges the specified collection of attributes for the specified collection of elements. This construct provides scripting access to the purge attribute version and baseline attribute version commands in CORE. Unlike the commands available via the user interface, this construct allows purging attribute versions for a specific attribute or collection of attributes.

**Elements**

Variable containing the collection of elements for which you wish to purge attribute versions.

**Attributes**

Variable containing the names of the attributes for which you wish to purge attribute versions. If the collection is empty, CORE will purge versions for all attributes.

**Type**

Symbol indicating the type of purge to be performed. If the value is interim, the first version and current version will be maintained with all interim attribute versions being purged. For any other value, the current version will be marked as the baseline and all previous versions will be discarded.
Remove Element From Folder

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element:</td>
<td>nil</td>
</tr>
<tr>
<td>Folder:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Removes the specified *element* from the *folder*. If no other folder contains the specified *element*, it will automatically be added to the root folder for the database class.

If the user does not have write permission for the element, an error will be signaled.

**Element**

Variable containing the element to remove from the *folder*.

**Folder**

Variable containing the folder from which the *element* should be removed.
**Rename Element**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element:</td>
<td>nil</td>
</tr>
<tr>
<td>New Name:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Changes the name of *element* from its current value to *new name*. An error is signaled if the user does not have permission to change the element name or if the new element name conflicts with an existing name (element names must be unique within a database class).

**Element**
Variable containing the element to be renamed.

**New Name**
Symbol specifying the new name for the *element*. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
Rename Folder

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folder:</td>
<td>nil</td>
</tr>
<tr>
<td>New Name:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Changes the name of *folder* from its current value to *new name*. An error is signaled if the user does not have permission to change the folder name or if the new folder name conflicts with an existing name (folder names must be unique within a parent folder just as directory names are unique in Microsoft Windows).

**Folder**
Variable containing the folder to be renamed.

**New Name**
Symbol specifying the new name for the *folder*. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
**Renumber Elements**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root Element:</td>
<td>nil</td>
</tr>
<tr>
<td>Number:</td>
<td>nil</td>
</tr>
<tr>
<td>Sort Block:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Renumbers *root element* and all of its descendants. The number attribute for *root element* is set to *number* and new hierarchical numbers are generated for the entire descendent tree. This is equivalent to the Renumber Element command in CORE.

**Root Element**

Variable containing the starting element for the renumber command.

**Number**

Hierarchical number specifying the new number for the *root element* and starting value for renumbering all descendants. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Sort Block**

Sort block definition of a sort block defined in the CORE project. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. For elements without structures such as FFBDs, this sort block governs the order in which descendants are renumbered.
Set Element Attribute

<table>
<thead>
<tr>
<th></th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>nil</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>nil</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td>nil</td>
<td></td>
</tr>
</tbody>
</table>

Description
Sets the value of the specified attribute of the specified element to value. An error is signaled if the user does not have permission to change the attribute or if a type mismatch occurs (e.g., attempting to assign a string value to a numeric attribute).

Element
Variable containing the element from the CORE database to access.

Attribute
Symbol naming the attribute to be set. The attribute should be defined in the CORE schema for the class of the element. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Value
Variable containing the value to be assigned to the specified attribute.
Set Element Diagnostic Results

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element:</td>
<td>nil</td>
</tr>
<tr>
<td>Diagnostic Type:</td>
<td>consistency</td>
</tr>
<tr>
<td>Value:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Sets the value of the specified diagnostic type of the specified element to value. The value specified replaces any existing diagnostic results. An error is signaled if the user does not have permission to change the element or if a type mismatch occurs.

**Element**
Variable containing the element from the CORE database to access.

**Diagnostic Type**
Name of the diagnostic results being set: consistency or custom. Specified by manual selection from the drop-down on the right. Note that completeness diagnostic results cannot be set via script since these are automatically computed and maintained by CORE.

**Value**
Variable containing the value (an ordered collection of strings) to be assigned to the specified diagnostic type.
Set Element Parameter Units

<table>
<thead>
<tr>
<th>Argument</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>nil</td>
<td>nil</td>
</tr>
<tr>
<td>Parameter</td>
<td>nil</td>
<td>nil</td>
</tr>
<tr>
<td>Value</td>
<td>nil</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Sets the value of the units fields associated with the specified parameter of the specified element to value. An error is signaled if the user does not have permission to change the parameter or if the parameter does not support units.

**Element**
Variable containing the element from the CORE database to access.

**Parameter**
Symbol naming the parameter for which the units is to be set. The parameter should be defined in the CORE schema for the class of the element. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Value**
Variable containing the value (a string) to be assigned to the units for the specified parameter.
COREscript Constructs
Database Update

Set Element Parameter Value

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element:</td>
<td>nil</td>
</tr>
<tr>
<td>Parameter:</td>
<td>nil</td>
</tr>
<tr>
<td>Value:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Sets the value of the specified parameter of the specified element to value. An error is signaled if the user does not have permission to change the parameter or if a type mismatch occurs (e.g., attempting to assign a string value to a numeric parameter).

**Element**
Variable containing the element from the CORE database to access.

**Parameter**
Symbol naming the parameter to be set. The parameter should be defined in the CORE schema for the class of the element. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Value**
Variable containing the value to be assigned to the specified parameter.
Set Relationship Attribute

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship:</td>
<td>nil</td>
</tr>
<tr>
<td>Attribute:</td>
<td>nil</td>
</tr>
<tr>
<td>Value:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Sets the value of the specified *attribute* of the specified *relationship* to *value*. An error is signaled if the user does not have permission to change the attribute or if a type mismatch occurs (e.g., attempting to assign a string value to a numeric attribute).

**Relationship**
Variable containing the relationship from the CORE database to access.

**Attribute**
Symbol naming the attribute to be set. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Value**
Variable containing the value to be assigned to the specified *attribute*. 
Unlock Object

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Unlocks the specified object (either an element, attribute, or parameter) permitting future changes. If the user does not have permission to administer the object, an error will be signaled.

Object
Variable containing the object (an element, attribute, or parameter) to be locked.
FILE MANAGEMENT CONSTRUCTS

Close File

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name:</td>
<td>nil</td>
</tr>
<tr>
<td>Launch File:</td>
<td>nil</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Closes the specified file. If an Open Output File construct is used or if a Set Default Output File construct is used in conjunction with one of the data output constructs, then a corresponding Close File construct should subsequently be included. If it is missing, the file is closed automatically.

**File Name**

String naming the file to close. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Launch File**

Boolean designating whether or not to open the file in the associated application. Can also be nil in which case, the user will be prompted to launch the file or not. Can be specified either by a variable from the drop-down on the left or by selecting the desired option on the right.
Open File Dialog

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>nil</td>
</tr>
<tr>
<td>File Name:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Opens the common file dialog provided by the host operating system, allowing the user to navigate all accessible storage volumes and directory structures and to select a file to be opened for reading.

**Title**
Descriptive name appearing in the title bar of the common dialog. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The type of this parameter should be a string.

**File Name**
String specifying the default file name or filter, such as *.csv, to appear in the selection field of the common dialog. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The common dialog will automatically open on the last file location selected from within CORE. This can be overridden by specifying a variable that contains a full path name.

**Return**
Variable to hold the result of this construct, which is a string specifying the full path to the selected file, or nil if the user selects the Cancel button.
Open Input File

<table>
<thead>
<tr>
<th>File Name:</th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>nil</td>
<td></td>
</tr>
</tbody>
</table>

Description

Opens the specified file for reading. It does no harm to open a file multiple times for input, but an error results if the file is already open for output. A file must be opened via this construct before it can be referenced in parsing constructs.

File Name

String naming the file to be opened. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. If the full path is not specified, it is assumed to be relative to the working directory in which CORE was launched.
Open Output File

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name:</td>
<td>nil</td>
</tr>
<tr>
<td>File Type:</td>
<td>ASCII</td>
</tr>
<tr>
<td>Overwrite File:</td>
<td>Overwrite</td>
</tr>
</tbody>
</table>

**Description**

Opens the file specified by the *file name*, which can then be used as the destination of subsequent output constructs and should be closed afterward.

**File Name**

String naming the file to open. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**File Type**

Symbol specifying the desired file type. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

If Rich Text Format (RTF) is specified, all output will be in RTF format, which can include multiple font styles as well as embedded graphics and can be imported into word processors. If HTML is specified, appropriate text translations will be made automatically to conform to HTML requirements. You can also generate plain ASCII files to output the results of simple queries, to generate input for other parsers or tools, etc.

**Overwrite File**

Boolean object indicating whether an existing file should be overwritten. If the argument value is *true*, an existing file will be overwritten. If the argument value is *false*, the new contents will be appended to the bottom of any existing file contents. Can be specified either by a variable from the drop-down on the left or a manual selection using the radio buttons on the right.
## Open Temporary Output File

<table>
<thead>
<tr>
<th>Argument</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Extension:</td>
<td>nil TXT</td>
</tr>
<tr>
<td>File Type:</td>
<td>nil ASCII</td>
</tr>
<tr>
<td>Overwrite File:</td>
<td>nil Overwrite Append</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

### Description
Calculates a default file name based upon the report and the execution time and opens the file, which can then be used as the destination of subsequent output constructs and should be closed afterward. The file name is the concatenation of the report name, the current date, and the current time. The temporary file is automatically placed in a TEMP subfolder of the user’s default output folder.

### File Extension
String naming the file extension for the temporary file. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

### File Type
Symbol specifying the desired file type. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

If RTF is specified, all output will be in RTF format, which can include multiple font styles as well as embedded graphics and can be imported into word processors. If HTML is specified, appropriate text translations will be made automatically to conform to HTML requirements. You can also generate plain ASCII files to output the results of simple queries, to generate input for other parsers or tools, etc.

### Overwrite File
Boolean object indicating whether an existing file should be overwritten. If the argument value is true, an existing file will be overwritten. If the argument value is false, the new contents will be appended to the bottom of any existing file contents. Can be specified either by a variable from the drop-down on the left or a manual selection using the radio buttons on the right.

### Return
Variable to hold the result of this construct, which is a string specifying the full path to the temporary file.
Save File Dialog

<table>
<thead>
<tr>
<th></th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>nil</td>
<td>Save File</td>
</tr>
<tr>
<td>File Name</td>
<td>nil</td>
<td>*.rtf</td>
</tr>
<tr>
<td>Return</td>
<td>nil</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Displays the standard Windows Save File dialog, prompting the user to select a file to open (for writing), and assigns the result to the return variable.

Title
String to use as the dialog window title. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

File Name
String specifying the default file name or filter, such as *.rtf, to appear in the selection field of the standard dialog. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The standard dialog will automatically open on the last file location selected from within. This can be overridden by specifying a variable that contains a full path name.

Return
Variable to hold the result of this construct, which is a string specifying the full path for the selected file, or nil if the user cancels the dialog.
Set Default Input File

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Name:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Sets the default file used by parsing constructs when their *file* parameters are not specified and multiple files are open for reading. Before an input file can be set as the default, it must first be opened.

**File Name**
String naming the input file to be used as the default. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The value of this parameter should be identical to the file name with which the file was opened.
Set Default Output File

<table>
<thead>
<tr>
<th>File Name:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Arguments</td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td></td>
</tr>
</tbody>
</table>

**Description**
Sets the default file used by output constructs when their `file` parameters are not specified and multiple files are open for writing. This file must still be opened before it can be used.

**File Name**
String naming the default output file. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
OUTPUT CONSTRUCTS

Comma-Delimited Output

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable:</td>
<td>nil</td>
</tr>
<tr>
<td>Destination:</td>
<td>nil File</td>
</tr>
<tr>
<td>File:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Outputs the contents of a variable formatted as a comma-delimited field. If the variable being output could contain a comma or other special character, the field is delimited by quotation marks and any embedded quotation marks are doubled (dates, time stamps, strings, and text blocks are output in this manner). Otherwise, the variable is output in standard format.

Variable
Variable whose contents is to be output in comma-delimited format.

Destination
Symbol specifying whether the output should be directed to a file, the on-screen transcript, both a file and the on-screen transcript, or a CORE2net stream.

File
String specifying the name of the destination file to which the variable is to be output.
Diagram File Output

<table>
<thead>
<tr>
<th>Argument</th>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>File:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>File Type:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>Diagram Type:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>Display Mode:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>Diagram Scale:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>Black &amp; White:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>Hierarchy Type:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>Relations:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>Target Classes:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>Create Node Map:</strong></td>
<td><strong>nil</strong></td>
</tr>
<tr>
<td><strong>Return:</strong></td>
<td><strong>nil</strong></td>
</tr>
</tbody>
</table>

**Description**

Outputs any CORE database or schema diagram as a graphics file. A diagram can be output in either WMF, JPG, PNG, BMP, or SVG format.

Optionally, for JPG and PNG formats, assigns to the `return` variable an ordered collection of diagram entity locators each of which corresponds to a hotspot on the output diagram.

**Object**

Variable containing an object of one of the following data types:
- Class
- Element
- Facility

**File**

String naming the *destination* of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**File Type**

Symbol specifying the desired file type. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.
Diagram Type
Symbol specifying the diagram type to output. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Display Mode
Symbol specifying the mode to use when displaying the diagram (with respect to diagram content marked for hiding). Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

The available display modes correspond to the options available on the diagram view within CORE.

- **Standard**: Shows all diagram content. No visible markings indicate any diagram objects that have been marked to hide. The standard view is most appropriate for engineers and analysts as they architect and define the system.
- **Marked**: Displays all diagram content but shows content that will be hidden with dotted lines.
- **Hidden (with label)**: Does not display diagram content marked as hidden. Layouts are compacted such that there is no layout cue that content is missing. The diagram frame is marked in the upper-right corner to indicate that content has been hidden.
- **Hidden**: Does not display diagram content marked as hidden. Layouts are compacted such that there is no layout cue that content is missing. No marking is placed on the diagram frame. This mode is most appropriate for use with customers so that they can focus on the diagram content (rather than thinking about what has been hidden).
- **Use Stored View**: Uses the diagram view settings as specified and saved in CORE.

Diagram Scale
Scale (as a percentage of the original diagram size) at which the diagram is to be output.

Black & White
Boolean indicating whether the diagram should be forced to black and white (as opposed to color). Can be specified either by a variable from the drop-down on the left or a manual selection from the buttons on the right.

Hierarchy Type
Symbol specifying the type of hierarchy diagram or spider diagram to be produced. Only meaningful if `diagram type` is set to “Hierarchy Diagram” or “Spider Diagram” and `object` is an Element. Can be any hierarchy type defined in the CORE project. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Relations
Collection of symbols naming relations to traverse in computing the closure of the specified element `object`. Used if the specified diagram is a custom hierarchy (i.e., `diagram type` is set to “Hierarchy Diagram” and `hierarchy type` is empty). Can be specified either by a variable from the drop-down on the left or manual additions into the collection on the right. The best kind of collection to use is a set.

Target Classes
Variable containing a collection of filter classes. Used if the specified diagram is a custom hierarchy (i.e., `diagram type` is set to “Hierarchy Diagram” and `hierarchy type` is empty). Can be specified either by a variable from the drop-down on the left or manual additions into the collection on the right. Can be of any of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection
Create Node Map
Boolean indicating whether a node map for the diagram should be generated. Can be specified either by a variable from the drop-down on the left or a manual selection from the buttons on the right.

Return
Variable to hold the result of this construct. It is either an ordered collection of diagram entity locators or nil depending on the setting of create node map.
Description

Encodes any CORE database or schema diagram as a graphic in an RTF file.

The diagram is reduced to the scale at which its height is less than or equal to the specified maximum height and its width is less than or equal to the specified maximum width (e.g., to ensure that it fits on a page). Whichever scaling factor is more restrictive is used to reduce both dimensions proportionally. This construct is not useful if the transcript is selected as the destination or if file output is chosen and the specified file was opened in ASCII mode.

Object

Variable containing an object of one of the following data types:
- Class
- Element
- Facility

Destination

Symbol specifying that output be directed either to a file, to an on-screen text transcript, to both a file and the on-screen transcript, or to a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.
COREscript Constructs
Output

File
String naming the destination of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Diagram Type
Symbol specifying the diagram type to output. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Display Mode
Symbol specifying the mode to use when displaying the diagram (with respect to diagram content marked for hiding). Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

The available display modes correspond to the options available on the diagram view within CORE.
- **Standard**
  Shows all diagram content. No visible markings indicate any diagram objects that have been marked to hide. The standard view is most appropriate for engineers and analysts as they architect and define the system.
- **Marked**
  Displays all diagram content but shows content that will be hidden with dotted lines.
- **Hidden (with label)**
  Does not display diagram content marked as hidden. Layouts are compacted such that there is no layout cue that content is missing. The diagram frame is marked in the upper-right corner to indicate that content has been hidden.
- **Hidden**
  Does not display diagram content marked as hidden. Layouts are compacted such that there is no layout cue that content is missing. No marking is placed on the diagram frame. This mode is most appropriate for use with customers so that they can focus on the diagram content (rather than thinking about what has been hidden).
- **Use Stored View**
  Uses the diagram view settings as specified and saved in CORE.

Width
Float specifying the maximum width of the diagram (used in conjunction with the height to determine diagram scale). Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Height
Float specifying the maximum width of the diagram (used in conjunction with the width to determine diagram scale). Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Black & White
Boolean indicating whether the diagram should be forced to black and white (as opposed to color). Can be specified either by a variable from the drop-down on the left or a manual selection from the buttons on the right.

Hierarchy Type
Symbol specifying the type of hierarchy diagram or spider diagram to be produced. Only meaningful if diagram type is set to “Hierarchy Diagram” or “Spider Diagram” and object is an element. Can be any hierarchy type defined in the CORE project. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Relations
Collection of symbols naming relations to traverse in computing the closure of the specified element object. Used if the specified diagram is a custom hierarchy (i.e., diagram type is set to “Hierarchy Diagram” and hierarchy type is empty.) Can be specified either by a variable from the drop-down on the left or manual additions into the collection on the right. The best kind of collection to use is a set.
Target Classes
Variable containing a collection of filter classes. Used if the specified diagram is a custom hierarchy (i.e., `diagram type` is set to "Hierarchy Diagram" and `hierarchy type` is empty). Can be specified either by a variable from the drop-down on the left or manual additions into the collection on the right. Can be of any of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection
Element Output

Description
Represents an element object from the CORE database in the report output. The amount and format of the information included from the element is determined by its representation.

Object
Variable containing an element to be referenced in or from which information is to be extracted for report output.

Destination
Symbol specifying that output be directed to either a file, an on-screen text transcript, both a file and the on-screen transcript, or a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

File
String naming the destination of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Representation
Ordered collection of representation fields, each of which can be the alias of the element’s class, the name of the element itself, the value of an attribute defined for the class, the targets of a relation, or hard-coded text.

Class Name
Outputs the alias of the element’s class, if defined, or the class name if the alias is undefined.

Element Name
Outputs the name of the element.
Attribute

Outputs the attribute value for the specified attribute with boilerplate text before and after the value. For each Attribute field, the desired attribute must be selected from the drop-down list.

If the attribute value is nil, nothing is output. Otherwise, the before text is output first, followed by the attribute value, and finally the after text is output. If boilerplate text is desired regardless of whether or not a value exists, the boilerplate text should be specified in a Text representation field instead of an Attribute representation field.

Targets

Outputs the targets for the specified relation with boilerplate text before and after the target list as well as text to separate individual targets in the list. For each Targets field, the desired relation and sort block must be selected from the drop-down lists. The sort block specifies not only in what order to output the targets but also what form the representation should take.

If no targets exist for the specified relation, nothing is output. Otherwise, the before text is output first, followed by the list of targets separated by the between text, and finally the after text is output. If boilerplate text is desired regardless of whether or not targets exist, the boilerplate text should be specified in a Text representation field instead of a Targets representation field.

Text

Outputs the boilerplate text specified in the text field.
Include External Boilerplate

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include File:</td>
<td>nil</td>
</tr>
<tr>
<td>Destination:</td>
<td>nil</td>
</tr>
<tr>
<td>File:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Imports an external file into a report. This is only meaningful if the specified destination is a file and the file was opened in RTF mode. Any file type is supported that can be read and understood by the word processor into which the RTF file will ultimately be imported.

Include File
String naming the file to be imported. Can be specified either by a variable from the drop-down on the left or by browsing using the button on the right. The file browse button opens a standard Windows file dialog prompting the user to select a file for importation.

Destination
Symbol specifying that output be directed to either a file, an on-screen text transcript, both a file and the on-screen transcript, or a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

File
String naming the destination of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
COREscript Constructs
Output

Include External Graphic

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include File:</td>
<td>nil</td>
</tr>
<tr>
<td>Destination:</td>
<td>nil</td>
</tr>
<tr>
<td>File:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Imports an external graphics *file* into a report. This is only meaningful if the specified *destination* is a file and the file was opened in RTF mode. Any picture type is supported that can be read and understood by the word processor into which the RTF file will ultimately be imported.

**Include File**

String naming the graphics file to be imported. Can be specified either by a variable from the drop-down on the left or by browsing using the button on the right. The file browse button opens a standard Windows file dialog prompting the user to select a file for importation.

**Destination**

A symbol specifying that output be directed to either a *file*, an on-screen text transcript, both a file and the on-screen transcript, or a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**File**

String naming the *destination* of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
Include Style Sheet

**Description**
Places a style sheet in a report file, on the report transcript, or both. This is only meaningful if the destination is a file, and the file is opened in RTF mode. Only one style sheet can be included in a script.

**Style Sheet**
A block of text containing style commands in the RTF language to parse and output. The easiest way to create this is to define your desired styles in a word processor, save the document in RTF format, and use the Extract button to have CORE copy the style header from the RTF file into this field. Word processors can also pull styles from multiple documents into a single document.

**Ignore DB Formatting**
Boolean indicating whether any styles associated with element text attributes output by the script should be removed upon output. Can be specified either by a variable from the drop-down on the left or manual selection from the buttons on the right.

**Destination**
Symbol specifying that output be directed to either a file, an on-screen text transcript, both a file and the on-screen transcript, or a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**File**
String naming the destination of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
## RDT Output

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Destination:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>File:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Include Attributes:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Include Relationships:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Include Structure:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Include Folders:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Include Access Control:</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Include Stored Views:</strong></td>
<td>nil</td>
</tr>
</tbody>
</table>

### Description
Outputs information from an element `object` in the CORE database in .rdt format that can be imported into CORE. The amount of information included from the element is determined by the include toggles.

### Object
Variable containing an element from which information is to be extracted for output.

### Destination
Symbol specifying that output be directed to either a file, an on-screen text transcript, both a file and the on-screen transcript, or a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

### File
String naming the `destination` of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

### Include Toggles
A series of toggles indicating whether or not the element attributes, relationships, structure, class folders, access control information, and stored view descriptions are to be included in the output. Can be specified either by a variable from the drop-down on the left or a manual selection from the Yes or No buttons on the right. If a variable is used, it must contain a Boolean value. The value of `true` corresponds to the button choice of Yes. The value of `false` corresponds to the button choice of No.
**Set Style**

<table>
<thead>
<tr>
<th>Style</th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination</td>
<td>nil</td>
<td>File</td>
</tr>
<tr>
<td>File</td>
<td>nil</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Sets the formatting *style* for subsequent output to a *file*. This is only meaningful if the file is opened in RTF mode. The *style* must previously have been included in the report.

**Style**

String naming a style. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

To populate the list on the right use the Load Style Sheet command on the Script menu to select the CORE report that includes the style sheet to be used. The styles will then be available in the drop down.

**Destination**

A symbol specifying that output be directed to either a *file*, an on-screen text transcript, both a file and the on-screen transcript, or a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**File**

String naming the *destination* of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
Table Output

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>Destination:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>File:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>Table Definition:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>Width:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>Show Banded Output:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>Merge Non-Repeat Cells:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>Ignore Sort Block:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>Ignore Filter:</td>
<td>nil ▼</td>
</tr>
</tbody>
</table>

Description

Builds a table using a collection of elements and the specified table definition, properly encoding the resulting table in an RTF, HTML, or text file. This construct reuses the same table definitions used in the Import from CSV and Export to CSV/Word commands for easy inclusion of formatted tables in reports.

Note: This construct is not available in CORE2net.

Collection

Name of a variable containing the collection over which to iterate, which can be of one of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set

If the collection is a dictionary, this construct iterates over the values stored in the dictionary, not the keys whereby the values are accessed.

The values contained in the collection must be of type Element.

Destination

Symbol specifying that output be directed either to a file, to an on-screen text transcript, to both a file and the on-screen transcript, or to a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

File

String naming the destination of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Table Definition

Table definition of a table defined in the CORE project. This specifies the columns and formatting of the table as well as any filtering or sorting applied to the collection. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.
**Width**
Float specifying the width of the table. This value is only used in formatting RTF tables where it sets the width of the table and the individual columns are then sized based upon the relative weights in the table definition. (HTML column widths are set based upon the weights in the table definition, and columns in text files are unformatted.) Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Show Banded Output**
Boolean indicating whether the rows of the table should be shaded to reflect alternating bands. This value is only used in formatting RTF tables. (Banding of HTML rows is controlled via CSS, and text files do not include style information.) Can be specified either by a variable from the drop-down on the left or a manual selection from the buttons on the right.

**Merge Non-Repeat Cells**
Boolean indicating whether child rows with non-repeated values should be merged. When generating tables, it is quite common to generate multiple rows based upon a single element. In this situation, if this setting is TRUE the cells in the corresponding column will be merged. (See Export to Word in the CORE help for more information). This value is only used in formatting RTF tables. Can be specified either by a variable from the drop-down on the left or a manual selection from the buttons on the right.

**Ignore Sort Block**
Boolean indicating whether the sort block specified as part of the table definition should be ignored. If TRUE, the order of the collection is preserved. If FALSE, the collection is sorted based upon the table definition sort block. Can be specified either by a variable from the drop-down on the left or a manual selection from the buttons on the right.

**Ignore Filter**
Boolean indicating whether the filter specified as part of the table definition should be ignored. If TRUE, the contents of the collection will not be filtered, and all elements in the collection will be included in the table. If FALSE, the collection is filtered based upon the table definition filter. Can be specified either by a variable from the drop-down on the left or a manual selection from the buttons on the right.
## COREscript Constructs
### Output

### Tag Output

<table>
<thead>
<tr>
<th>Tag:</th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Destination:</th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td></td>
<td>File</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>File:</th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Description

Places the *tag* text in the script output. Like the Text Output construct except that no translation is performed on the *tag*. This allows the direct output of RTF and HTML commands.

### Tag

A text string. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

### Destination

A symbol specifying that output be directed to either a *file*, an on-screen text transcript, both a file and the on-screen transcript, or a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

### File

String naming the *destination* of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
## Text Output

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Text:</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Description

Places fixed *text* in the output.

### Text

A text string to output. Both carriage return-line feeds and tab spacing are allowed in this field.

### Destination

A symbol specifying that output be directed to either a *file*, an on-screen text transcript, both a file and the on-screen transcript, or a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

### File

String naming the *destination* of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
Variable Output

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable:</td>
<td>nil</td>
</tr>
<tr>
<td>Text Before:</td>
<td>nil</td>
</tr>
<tr>
<td>Text After:</td>
<td>nil</td>
</tr>
<tr>
<td>Destination:</td>
<td>nil</td>
</tr>
<tr>
<td>File:</td>
<td>File</td>
</tr>
</tbody>
</table>

**Description**

Places a string representation of the object contained in the specified `variable` in a report output with boilerplate text before and after the string.

If the variable is `nil`, nothing is output. Otherwise the `text before` is output first, followed by the `variable` value, and finally the `text after` is output. If boilerplate text is desired regardless of whether or not a value exists, the boilerplate text should be specified in a Text Output Construct rather than in a Variable Output Construct.

**Variable**

Variable whose value is to be output.

**Text Before**

Text to be output before outputting the `variable` value if it exists.

**Text After**

Text to be output after outputting the `variable` value if it exists.

**Destination**

A symbol specifying that output be directed to either a `file`, an on-screen text transcript, both a file and the on-screen transcript, or a CORE2net stream. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**File**

String naming the `destination` of file output. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
# PARSING CONSTRUCTS

**At End**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Assigns to the *return* variable a Boolean value indicating whether or not the current position pointer in the specified *file* is at the end of the *file*. An error results if the *file* has not previously been opened for input.

**File**

String naming the input file to be tested. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The value of this parameter should be identical to the *file name* with which the file was opened. If this parameter is not specified, the default input file is tested.

**Return**

Variable to hold the result of this construct.
COREscript Constructs
Parsing

Get Position

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the *return* variable the integer index of the current position pointer in the input stream specified by the *file* parameter. An error results if the *file* has not previously been opened for input. This construct can be used to temporarily record a point in the *file* in order to return to it later using the Set Position construct.

**File**
A string naming the input file to be queried. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The value of this parameter should be identical to the *file name* with which the file was opened. If this parameter is not specified, the default input file is queried.

**Return**
Variable to hold the result of this construct.
COREscript Constructs
Parsing

Next

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File:</td>
<td>nil</td>
</tr>
<tr>
<td>Access:</td>
<td>Character</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Parses a string from the specified file, beginning at the current position in the file, advances the current position pointer past what has been read, and assigns the string to the return variable. An error results if the file has not previously been opened for input.

File
A string naming the input file to be read. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The value of this parameter should be identical to the file name with which the file was opened. If this parameter is not specified, the default input file is read.

Access
Determines how much is read from the specified file. If this parameter is set to Character, the next byte is read and converted to a string. An error results if the current position pointer in the file is at the end of the file.

If this parameter is set to Word and the current position in the file is the end of the file, then this construct returns nil. Otherwise, it skips forward from the current position to the first alphabetic or numeric character it encounters or to the end of the file. It then reads any contiguous sequence of letters and digits beginning at that point, stopping, if necessary, at the end of the file. This construct may return an empty string if there are no letters or digits remaining in the file.

If this parameter is set to Line, then this construct reads all characters up to the next line feed, the next carriage return and line feed combination, or the end of the file. If the current position in the file is at the end of the file, then this construct returns an empty string.

Return
Variable to hold the result of this construct.
COREscript Constructs
Parsing

**Peek**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File:</td>
<td>nil</td>
</tr>
<tr>
<td>Access:</td>
<td>☑ Character □ Word □ Line</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Parses a string from the specified *file*, beginning at the current position in the *file*, and assigns the string to the *return* variable. An error results if the *file* has not previously been opened for input. This construct differs from the Next construct in that the current position pointer in the *file* is not advanced, such that the next parsing construct to be invoked begins at the same point in the input stream.

**File**

A string naming the input file to be read. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The value of this parameter should be identical to the *file name* with which the file was opened. If this parameter is not specified, the default input file is read.

**Access**

Determines how much is read from the specified *file*. If this parameter is set to *Character*, the next byte is read and converted to a string. Unlike the Next construct, an error does not result if the current position pointer in the *file* is at the end of the *file*. Instead, this construct returns *nil*.

If this parameter is set to *Word* and the current position in the *file* is the end of the *file*, then this construct returns *nil*. Otherwise, it skips forward from the current position to the first alphabetic or numeric character it encounters or to the end of the file. It then reads any contiguous sequence of letters and digits beginning at that point, stopping, if necessary, at the end of the *file*. This construct may return an empty string if there are no letters or digits remaining in the *file*.

If this parameter is set to *Line*, then this construct reads all characters up to the next line feed, the next carriage return and line feed combination, or the end of the *file*. Unlike the Next construct, if the current position in the *file* is at the end of the *file*, then this construct returns *nil*.

**Return**

Variable to hold the result of this construct.
Set Position

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File:</td>
<td>nil</td>
</tr>
<tr>
<td>Position:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Moves the current position pointer in the specified file to the specified position. An error results if the file has not previously been opened for input.

**File**

A string naming the input file which position pointer is to be moved. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The value of this parameter should be identical to the file name with which the file was opened. If this parameter is not specified, the position pointer of the default input file is set.

**Position**

An integer specifying the index of the position in the specified file to which its position pointer is to be set. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Note that file positions are zero-based – that is, the beginning of a file is index 0, and the position at the index equal to the size of the file (where the At End construct returns true) is after the last byte of the file. An error results if the specified position is negative or greater than the size of the file.

**Return**

Variable to hold the result of this construct, which is an integer specifying the previous position pointer in the specified file, the position at the time this construct was invoked.
COREscript Constructs
Parsing

Size

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the return variable an integer, which is the number of bytes in the specified file. An error results if the file has not previously been opened for input.

File
A string naming the input file to be queried. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The value of this parameter should be identical to the file name with which the file was opened. If this parameter is not specified, the default input file is queried.

Return
Variable to hold the result of this construct.
COREscript Constructs
Parsing

Skip

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>File:</td>
<td>nil</td>
</tr>
<tr>
<td>Skip:</td>
<td>Over Separators</td>
</tr>
<tr>
<td></td>
<td>To First Alphanumeric</td>
</tr>
<tr>
<td></td>
<td>To Character</td>
</tr>
<tr>
<td>Character:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Advances the current position pointer in the specified file to the specified place in its input stream. Note that the position might not have to change in order for the conditions implied by the skip parameter to be met (i.e., the input stream might already be positioned at the desired type of object). An error results if the file has not previously been opened for input.

File
A string naming the input file which position pointer is to be moved. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The value of this parameter should be identical to the file name with which the file was opened. If this parameter is not specified, the position pointer of the default input file is set.

Skip
Specifies how far the current position pointer advances through the specified file by informing the report writer what kind of object to skip over or to skip to in the input stream. If this parameter is set to Over Separators, then the parser skips up to the first non-white space character or up to the end of the file. White space is defined as spaces, tabs, carriage returns, line feeds, and form feeds.

If this parameter is set to To First Alphanumeric, then the parser skips all characters until it encounters the first letter or digit after the point in the input stream at which this construct began or until it reaches the end of the file.

If this parameter is set To Character, then the report author can select a variable containing a printable character or can manually type a character in the field corresponding to this choice, and, when this construct is invoked, the parser will advance to the next occurrence of the specified character in the input stream or to the end of the file. Or the report author can choose a non-printable character from the drop-down list corresponding to this choice, and, when this construct is invoked, the parser will advance to the next occurrence of the specified character in the input stream or to the end of the file.

Character
If a variable is specified in this field and the To Character radio button is selected, then the parser attempts to match in the input stream the value of this variable instead of any character that has been manually entered in the field next to the radio button.

Return
Variable to hold the result of this construct, which is an integer specifying the previous position pointer in the specified file, the position at the time this construct was invoked.
### Description
Advances the current position pointer in the specified file to the specified place in its input stream and assigns to the return variable the string of all characters (if any) over which the pointer was advanced. Note that the position might not have to change in order for the conditions implied by the up to parameter to be met (i.e., the input stream might already be positioned at the desired type of object). An error results if the file has not previously been opened for input.

### File
A string naming the input file to be read. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right. The value of this parameter should be identical to the file name with which the file was opened. If this parameter is not specified, the default input file is read.

### Character
If a variable is specified in this field, then the parser reads from the input stream until it encounters the value of this variable. If a special character is required (such as a carriage return, line feed, tab, or space) it can be specified using the drop-down list in the argument pane.

### Return
Variable to hold the result of this construct.

---

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Character</strong></td>
<td>nil</td>
</tr>
<tr>
<td><strong>Return</strong></td>
<td>nil</td>
</tr>
</tbody>
</table>
**PROJECT QUERY CONSTRUCTS**

**Project Property**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Property:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

The Project Property Query construct assigns to the `return` variable the value of the specified `property` for the given project `object`.

**Object**

Project in the CORE database for which the property value is desired. Note that this argument is a project object itself, not the name of a project.

**Property**

Symbol naming the property of which the value is desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. The available properties are the following:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>basePath</code></td>
<td>a string that is the full path set in the Base Path property for the project. Returns nil if this path has not been set.</td>
</tr>
<tr>
<td><code>completenessCheckerName</code></td>
<td>a descriptive string indicating the completeness check level currently in use for the project.</td>
</tr>
<tr>
<td><code>creationStamp</code></td>
<td>the time stamp (combination of date and time) at which the project was created.</td>
</tr>
<tr>
<td><code>creator</code></td>
<td>a symbol that is the name of the user that created the project.</td>
</tr>
<tr>
<td><code>customerAddress</code></td>
<td>formatted text that is the contents of the project Customer Address property. Returns nil if this property is empty.</td>
</tr>
<tr>
<td><code>customerName</code></td>
<td>a string that is the contents of the project Customer Name property. Returns nil if this property is empty.</td>
</tr>
<tr>
<td><code>description</code></td>
<td>formatted text that is the contents of the project Description property. Returns nil if this property is empty.</td>
</tr>
<tr>
<td><code>guidance</code></td>
<td>formatted text that is the contents of the project Guidance property. Returns nil if this property is empty.</td>
</tr>
<tr>
<td><code>lastAccessedStamp</code></td>
<td>the time stamp (combination of date and time) at which the project was last read from.</td>
</tr>
<tr>
<td><code>modificationStamp</code></td>
<td>the time stamp (combination of date and time) at which the project property was last modified.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>a symbol that is the name of the project</td>
</tr>
<tr>
<td><code>organizationAddress</code></td>
<td>formatted text that is the contents of the project Organization Address property. Returns nil if this property is empty.</td>
</tr>
<tr>
<td><code>organizationName</code></td>
<td>a string that is the contents of the project Organization Name property. Returns nil if this property is empty.</td>
</tr>
<tr>
<td><code>schemaBaselineName</code></td>
<td>a string that is the name of the schema selected as the base schema when the project was created.</td>
</tr>
<tr>
<td><code>schemaVersion</code></td>
<td>a hierarchical number that is the version (e.g., 8.0, 9.0) of the schema being used in the project.</td>
</tr>
</tbody>
</table>
COREscript Constructs
Project Query

Return
Variable to hold the result of this construct.
Project Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

The Project Query construct assigns to the *return* variable the project object specified by its *name*.

**Name**

Symbol naming the desired project. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**

Variable to hold the result of this construct, which is a project object in the CORE database.
Project Registry Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Key:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Returns the value associated with the specified *key* in the project registry. If the value has not been set, returns nil.

**Object**

Project in the CORE database from which the registry value is to be retrieved. Note that this argument is a project object, not the name of a project.

**Key**

Symbol specifying the name of the registry entry to be accessed.

**Return**

Variable to hold the result of this construct.
COREscript Constructs
Project Query

Projects

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
The Projects construct returns a collection of all projects defined in the CORE database.

Return
Variable to hold the result of this construct, which is a set of projects. Note that the objects in the set are project objects themselves, not the names of projects.
Remove Project Registry Key

**Description**
Removes the entry associated with the specified key from the project registry. Returns an error if the user does not have the project administrator permissions necessary to remove a registry entry.

**Object**
Project in the CORE database from which registry value is to be removed. Note that this argument is a project object, not the name of a project.

**Key**
Symbol specifying the name of the registry entry to be accessed.

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Key:</td>
<td>nil</td>
</tr>
</tbody>
</table>
### Set Project

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Sets the project for subsequent database and schema queries. If not explicitly set using a `Set Project` construct, project is defaulted to the project associated with the window from which a script is run.

**Project**

Project in the CORE database from which subsequent retrievals are to be made. Note that this argument is a project object, not the name of a project.

**Return**

Project in the CORE database that is the previously set project. Note that this argument is a project object, not the name of a project.
Set Project Registry Value

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>rail</td>
</tr>
<tr>
<td>Key:</td>
<td>rail</td>
</tr>
<tr>
<td>Value:</td>
<td>rail</td>
</tr>
</tbody>
</table>

**Description**
Sets the specified value at the specified key in the project registry. Returns an error if the user does not have write permission to the project.

**Object**
Project in the CORE database for which the registry value is to be set. Note that this argument is a project object, not the name of a project.

**Key**
Symbol specifying the name of the registry entry to be set.

**Value**
Variable to hold the result of this construct.
PROMPTING CONSTRUCTS

Composite Prompt

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>nil</td>
</tr>
<tr>
<td>Message</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description

Composite prompt allows the report author to replace a series of individual prompt windows with a single, combined prompt. The Composite Prompt construct acts as a block and combines all of the enclosed prompts into a single dialog displayed to the user during report execution. Note that only prompt constructs can be inserted within a Composite Prompt block.

Arguments and parameters for individual prompting constructs within the Composite Prompt block operate in exactly the same manner as when a prompt is used separately. The only difference is that the individual prompts are combined into a form-type dialog.

Title

String to use as the dialog window title. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Message

String to use as the notification message displayed at the top of the dialog. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
### Drop-Down Selection Prompt

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>Title:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>Message:</td>
<td>nil ▼</td>
</tr>
<tr>
<td>Return:</td>
<td>nil ▼</td>
</tr>
</tbody>
</table>

#### Description
Selection prompts allow the report to prompt the user to select one or more items from a list. The Drop-Down Selection Prompt construct prompts the user to select one string from a drop-down list representing a collection of strings and assigns the selected string to the `return` variable. CORE automatically tracks user selections and uses the last selection as the initial / default selection when the report is run again. Note that if the Cancel button is pressed, `nil` is returned.

#### Collection
Collection of strings from which to select. Can be of one of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection
If the collection is a dictionary, the strings should be the values stored in the dictionary, not the keys whereby other values are accessed.

#### Title
String to use as the dialog window title. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

#### Message
String to use as the notification message displayed in the dialog. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

#### Return
Variable to hold the result of this construct, which is a string or `nil`. 
Element Selection Prompt

**Arguments**

<table>
<thead>
<tr>
<th>Class:</th>
<th>nil</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>nil</td>
</tr>
<tr>
<td>Class Label:</td>
<td>nil</td>
</tr>
<tr>
<td>Candidate Label:</td>
<td>nil</td>
</tr>
<tr>
<td>Default Class:</td>
<td>nil</td>
</tr>
<tr>
<td>Default Element:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Parameters**

The Element Selection Prompt construct displays an action dialog allowing the user to select one element from any class in the collection of classes and assigning the selection to the return variable. If no element is selected or if the Cancel button is pressed, then this construct returns nil.

If a default class or element is specified, these will be highlighted when the dialog is presented during execution. If no defaults are specified, CORE automatically tracks user selections and uses the last selections as the initial / default selection when the report is run again.

**Classes**

Collection of valid classes from which the user may select an element. Can be of any of the following data types:

- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection

**Title**

String to be used as the dialog window title. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Class Label**

String to be used as the label for the class pane. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Candidate Label**

String to be used as the label for the pane listing all elements in the selected class. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Default Class**

A class object to be used as the default class selection in the prompt. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.
COREscript Constructs
Prompting

**Default Element**
Symbol naming the default element to be selected in the dialog. Can be specified either by a variable from the drop-down on the left or a manual selection using an element selection window opened by pressing the Select Element button.

**Return**
Variable to hold the result of this construct, which is an element or *nil*. 
Message Prompt

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>nil</td>
</tr>
<tr>
<td>Message:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
The Message Prompt construct displays a notification dialog for the user. The user must close the dialog before proceeding.

**Title**
String to use as the dialog window title. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Message**
String to use as the notification message displayed in the dialog. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
Multiple Selection Prompt

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection: nil</td>
<td>Select</td>
</tr>
<tr>
<td>Title: nil</td>
<td>Objects</td>
</tr>
<tr>
<td>Object Label: nil</td>
<td>Selections</td>
</tr>
<tr>
<td>Selection Label: nil</td>
<td></td>
</tr>
<tr>
<td>Sort Selections: nil</td>
<td>Yes</td>
</tr>
<tr>
<td>Return: nil</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Selection Prompts allow the report to prompt the user to select one or more items from a list. The Multiple Selection Prompt construct prompts the user to select one or more objects from a list and assigns the collection of selected objects to the `return` variable. Each object is represented as a string in the list presented to the user. Note that if the Cancel button is pressed, `nil` is returned. If the OK button is pressed without selecting an item from the list, an empty collection is returned.

CORE automatically tracks user selections and uses the last selections as the initial / default selection when the report is run again.

**Collection**

Collection of objects from which to select. Can be one of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection

If the collection is a dictionary, the user-selectable objects should be the values stored in the dictionary, not the keys whereby other values are accessed.

**Title**

String to be used as the dialog window title. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Object Label**

Indicates the string with which to label the candidate list (the list from which selections are being made). Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Selection Label**

Indicates the string with which to label the list of selected items. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Sort Selections**

Boolean object indicating whether or not the selections should be. If the argument value is `true`, this construct returns the user selections in a sorted collection using a simple comparison on the objects in the collection as the sort criterion (but each type of object handles the comparison in its own way). If the argument value is `false`, the selections are returned in the `collection` in the order selected by the user. Can be specified either by a variable from the drop-down on the left or a manual selection from the buttons on the right.
Return
Variable to hold the selected subset of the collection described above. It may be nil or a sorted collection.
Project Explorer Prompt

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>Select Project Segment</td>
</tr>
<tr>
<td>Message:</td>
<td>Select the desired segment of the project.</td>
</tr>
<tr>
<td>Include Project:</td>
<td>☐ Yes ☐ No</td>
</tr>
<tr>
<td>Include Packages:</td>
<td>☐ Yes ☐ No</td>
</tr>
<tr>
<td>Include Folders:</td>
<td>☐ Yes ☐ No</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description

Project Explorer Prompts display an expandable tree view of the database allowing the user to select a single item. Depending upon the options selected, the user can select from packages, folders, or the entire database. The selected object is assigned to the return variable. Note that if the Cancel button is pressed, nil is returned.

CORE automatically tracks user selections and uses the last selection as the initial / default selection when the report is run again.

Title

String to use as the dialog window title. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Message

String to use as the notification message displayed in the dialog. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Include Project

Boolean indicating whether or not the project should be included in the tree view as a possible selection. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Include Packages

Boolean indicating whether or not packages should be included in the tree view as a possible selection. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Include Folder

Boolean indicating whether or not folders should be included in the tree view as a possible selection. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Return

Variable to hold the selected object or nil.
Prompt Separator

| When executed as part of a composite prompt, this construct displays a horizontal separator between the prompt fields. |
| There are no parameters to set for a separator construct. |

Description

When used within a Composite Prompt block, the Prompt Separator draws a horizontal line separating one set of fields from the next set of fields. Using this construct helps visually group and separate items on the composite prompt.

When used outside of a Composite Prompt block, this construct has no effect.
COREscript Constructs
Prompting

Report Settings Prompt

The Report Settings Prompt construct was removed from CORE in version 7. The “prepared by” and “prepared for” information can be accessed via the Report Settings Access Construct but the prompt is no longer available.

This construct remains as a placeholder allowing legacy reports to run without error.

Description

Beginning with CORE 7, the values previously accessed as report settings have been migrated to project properties. The “prepared by” fields are now the organization name / address properties. The “prepared for” fields are now the customer name / address properties.

Though the constructs are ignored during execution, they remain in the construct language as placeholders allowing existing reports that use these constructs to execute without error.
**Single Selection Prompt**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection:</td>
<td>nil</td>
</tr>
<tr>
<td>Title:</td>
<td>nil</td>
</tr>
<tr>
<td>Object Label:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Selection Prompts allow the report to prompt the user to select one or more items from a list. The Single Selection Prompt construct prompts the user to select a single object from a list and assigns the selection to the `return` variable. Each object is represented as a string in the list presented to the user. Note that if the Cancel button is pressed, `nil` is returned.

CORE automatically tracks user selections and uses the last selection as the initial / default selection when the report is run again.

**Collection**

Collection of objects from which to select. Can be of one of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection

If the collection is a dictionary, the user-selectable objects should be the values stored in the dictionary, not the keys whereby other values are accessed.

**Title**

String to use as the dialog window title. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Object Label**

Indicates the string with which to label the pane displaying the currently selected object, if any. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Return**

Variable to hold the selected object or `nil`.
**Value Prompt**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title:</td>
<td>nil</td>
</tr>
<tr>
<td>Message:</td>
<td>nil</td>
</tr>
<tr>
<td>Default:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

The Value Prompt construct prompts the user to enter a string before continuing and assigns it to the *return* variable. Note that if the Cancel button is pressed, an empty string is returned.

If a default value is specified, this will be shown as the initial / default value when the dialog is presented during execution. If no default is specified, CORE automatically tracks user input and uses the input as the initial / default value when the report is run again.

**Title**

String to be used as the dialog window title. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Message**

String to be used as the notification message displayed in the dialog. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Default**

Initial string to be displayed and selected in the entry field when the prompt is opened. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Return**

Variable to hold the string entered by the user.
YesOrNo Prompt

**Description**
The YesOrNo Prompt construct asks the user a yes/no question and assigns to the `return` variable a Boolean `true` if the user clicks on the Yes button or a Boolean `false` if the user clicks on the No button.

Because a default value (yes or no) is always specified in this construct, this value will be shown as the initial / default selection each time the report is run.

**Title**
String to be used as the dialog window title. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Message**
String to be used as the notification message displayed in the dialog. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Default**
A Boolean indicating which button is the default button. If the object is `true`, then the default is the Yes button. If the object is `false`, then the default is the No button. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Return**
Variable to hold the result of this construct, which is a Boolean.
SCHEMA QUERY CONSTRUCTS

Attribute Definition Property

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Property:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the *return* variable the value of the specified *property* for the given attribute definition.

**Object**
Attribute definition from the CORE schema for which the property value is desired.

**Property**
Symbol naming the property which value is desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**
Variable to hold the result of this construct.
Class Attributes

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the `return` variable a collection of all attribute definitions for the specified `class`. These definitions can then be queried for their property values (e.g., name, type) as necessary.

**Class**
A class defined in the CORE schema for which the attribute definitions are desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. Note that this argument is a class object itself, not just the name of a class.

**Return**
Variable to hold the result of this construct, which is a sorted collection of attribute definitions ordered alphabetically by display name.
**COREscript Constructs**

**Schema Query**

### Class Parameters

<table>
<thead>
<tr>
<th>Class:</th>
<th>nil</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
</table>

**Description**

Assigns to the `return` variable a collection of all parameter definitions for the specified `class`. These definitions can then be queried for their property values (e.g., name, type) as necessary.

**Class**

A class defined in the CORE schema for which the attribute definitions are desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. Note that this argument is a class object itself, not just the name of a class.

**Return**

Variable to hold the result of this construct, which is a sorted collection of parameter definitions ordered alphabetically by display name.
COREscript Constructs
Schema Query

Class Property

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Property:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the *return* variable the value of the specified *property* for the given class *object*.

**Object**
Class in the CORE schema for which the property value is desired. Note that this argument is a class object itself, not just the name of a class.

**Property**
Symbol naming the property which value is desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**
Variable to hold the result of this construct.
Class Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the *return* variable the class object specified by its *name*.

**Name**
Symbol naming the desired class. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**
Variable to hold the result of this construct, which is a class object in the CORE schema.
Class Relations

<table>
<thead>
<tr>
<th>Class</th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description
Assigns to the return variable a collection of all relations from the specified class. These definitions can then be queried for their property values (e.g., name, complement) as necessary.

Class
A class defined in the CORE schema for which the relations are desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. Note that this argument is a class object itself, not just the name of a class.

Return
Variable to hold the result of this construct, which is a sorted collection of relations ordered alphabetically by name.
**COREscript Constructs**

**Schema Query**

### Class Target Classes

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class:</td>
<td>nil</td>
</tr>
<tr>
<td>Relation:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Assigns to the *return* variable a collection of all classes defined in the CORE schema as targets of the specified *relation* from the specified *class*. Relationships defined by the *relation* can be created in the CORE database with elements of the specified *class* as the source and elements of the target classes as targets, subject to the limitations imposed by the user in the schema.

**Class**

Class defined in the CORE schema for which the target classes of the specified *relation* are desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. Note that this argument is a class object itself, not just the name of a class.

**Relation**

Relation defined in the CORE schema for which the target classes are desired. Should be a valid relation from the specified *class*. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. Note that this argument is a relation object itself, not just the name of a relation.

**Return**

Variable to hold the result of this construct, which is a sorted collection of class objects ordered alphabetically by display name.
Database Classes

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the `return` variable a collection of all concrete (i.e., non-abstract) classes in the schema.

Return
Variable to hold the result of this construct, which is an ordered collection of classes ordered alphabetically by display name. Note that the objects in the collection are class objects themselves, not just the names of classes.
Facilities

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the *return* variable a collection of all facilities defined in the CORE schema.

**Return**
Variable to hold the result of this construct, which is a sorted collection of facilities ordered alphabetically by name. Note that the objects in the collection are facility objects themselves, not just the names of facilities.
Facility Database Classes

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facility:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the return variable a collection of all concrete (i.e., non-abstract) classes defined in the CORE schema as belonging to the specified facility.

Facility
Facility in the CORE schema to be queried. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. Note that this argument is a facility object itself and not just the name of a facility.

Return
Variable to hold the result of this construct, which is an ordered collection of classes ordered alphabetically by display name. Note that the objects in the collection are class objects themselves, not just the names of classes.
Facility Property

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object: nil</td>
<td>nil</td>
</tr>
<tr>
<td>Property: nil</td>
<td>nil</td>
</tr>
<tr>
<td>Return: nil</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Assigns to the *return* variable the value of the specified *property* for the given facility *object*.

**Object**

Facility in the CORE schema for which the property value is desired. Note that this argument is a facility object, not just the name of a facility.

**Property**

Symbol naming the property which value is desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**

Variable to hold the result of this construct.
Facility Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the `return` variable the facility object specified by its `name`.

**Name**
Symbol naming the desired facility. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**
Variable to hold the result of this construct, which is a facility object in the CORE schema.
Facility Schema Classes

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Facility:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Assigns to the *return* variable a collection of all abstract and concrete classes used to identify the classes in the CORE schema that belong to the specified *facility*.

**Facility**

Facility in the CORE schema to be queried. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. Note that this argument is a facility object itself and not just the name of a facility.

**Return**

Variable to hold the result of this construct, which is a sorted collection of classes ordered alphabetically by display name. Note that the objects in the collection are class objects themselves, not just the names of classes.
Parameter Definition Property

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Property:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the *return* variable the value of the specified *property* for the given parameter definition.

**Object**
Parameter definition from the CORE schema for which the property value is desired.

**Property**
Symbol naming the property which value is desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Return**
Variable to hold the result of this construct.
COREscript Constructs
Schema Query

Relation Attributes

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the return variable a collection of all attribute definitions for the specified relation. These definitions can then be queried for their property values (e.g., name, type) as necessary.

Relation
A relation defined in the CORE schema for which the attribute definitions are desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. Note that this argument is a relation object itself, not just the name of a class.

Return
Variable to hold the result of this construct, which is a sorted collection of attribute definitions ordered alphabetically by display name.
Relation Property

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Property:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description

Assigns to the return variable the value of the specified property for the given relation object.

Object
Relation in the CORE schema for which the property value is desired. Note that this argument is a relation object itself, not just the name of a relation.

Property
Symbol naming the property which value is desired. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Return
Variable to hold the result of this construct.
COREscript Constructs
Schema Query

Relation Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the return variable the relation object specified by its name.

Name
Symbol naming the desired relation. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Return
Variable to hold the result of this construct, which is a relation object in the CORE schema.
Relations

<table>
<thead>
<tr>
<th>Return:</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the *return* variable a collection of all relations in the schema.

Return
Variable to hold the result of this construct, which a sorted collection of relations ordered alphabetically by name. Note that the objects in the collection are relation objects themselves, not just the names of relations.
Schema Classes

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the `return` variable a collection of all classes in the schema.

**Return**
Variable to hold the result of this construct, which a sorted collection of classes ordered alphabetically by name. Note that the objects in the collection are class objects themselves, not just the names of classes.
SIMULATION CONSTRUCTS

Resource Level Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the `return` variable the currently available amount of the specified resource if the resource is used in the scope of the current model being simulated. If the resource is defined but not utilized by the model, the construct returns the initial value of the resource. If the resource is undefined, the construct returns zero.

**Object**
Variable containing the element in the Resource class from the CORE database to query.

**Return**
Variable to hold the result of this construct, which is either a float or integer depending on the Amount Type attribute of the Resource element.

Note: Only available to Simulation scripts.
COREscript Constructs
Simulation

Script Context Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the `return` variable the context in which the script is executing (i.e., the Function element in the case of a Begin Logic, Duration, End Logic, Exit Logic, or Timeout attribute; the Resource element in the case of a Maximum Amount attribute; and the relationship object in the case of an Amount attribute).

**Return**
Variable to hold the result of this construct, which is either a Function, Resource, or relationship object.

Note: Only available to Simulation scripts.
Simulator Property Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
<td></td>
</tr>
<tr>
<td>Return</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**
Assigns to the *return* variable the selected simulator *property* during simulation execution.

**Property**
The simulator property to be returned. This field is a drop-down list with the following selections:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>areLinksConstrained</code></td>
<td>a Boolean that is true if the Link constraints are being applied, false otherwise.</td>
</tr>
<tr>
<td><code>executionContext</code></td>
<td>the simulation model context providing access to the actual execution instances of functions in execution and the items being exchanged.</td>
</tr>
<tr>
<td><code>executionPath</code></td>
<td>a collection from most recent to most distant showing the functional decomposition that has resulted in this specific execution instance of the function.</td>
</tr>
<tr>
<td><code>time</code></td>
<td>the current simulation clock time</td>
</tr>
<tr>
<td><code>timeLimit</code></td>
<td>user set simulation clock time at which the simulation will automatically be terminated if it has not already completed. Returns <em>nil</em> if it has not been set.</td>
</tr>
</tbody>
</table>

**Return**
Variable to hold the result of this construct.

Note: Only available to Simulation scripts.
COREscript Constructs
Simulation

THIS PAGE INTENTIONALLY BLANK
UTILITIES CONSTRUCTS

Available Diagrams Query

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diagram Query:</td>
<td>allAvailableViews</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Returns an `OrderedCollection` of diagram types (represented as symbols) sorted in alphabetical order. Using the construct insulates the script against changes between versions and editions of CORE, providing a standard mechanism to access the various collections of diagrams.

**Diagram Query**

Desired collection of available views to return (`allAvailableViews`, `behavioralViews`, `physicalInterfaceViews`, or `physicalViews`). Can be specified by selection from the drop-down on the right.

**Return**

Variable to hold the result of this construct.
Beep

*(No settings required)*

**Description**

Beeps the system speaker.
Breakpoint

Table:

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Line Fields:</td>
<td>nil</td>
</tr>
<tr>
<td>Multiple Line Fields:</td>
<td>nil</td>
</tr>
<tr>
<td>List Panes:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description

Stops script execution and opens a debugger on the current script.

Single Line Fields

An integer specifying the number of entry fields to display values in the debugger (good for dates, strings, integers, elements, etc.).

Multiple Line Fields

An integer specifying the number of formatted text fields to display large blocks of information (good for text fields such as descriptions).

List Panes

An integer specifying the number of list panes to display collections (intended for collections such as arrays, ordered collections, and dictionaries).

Although any variable can be displayed in any pane, it is most valuable to use the pane best suited for that variable.
Cancel Report

*(No settings required)*

**Description**

Aborts the execution of the script. This is the same as if the Cancel button on the progress indicator is pressed.

Note: Not available to Simulation scripts.
Change Indicator

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Allows the report to change the message currently displayed in the progress indicator. This is a useful technique to keep the individual running a report informed of the status. Note that the prior message is assigned to the *return* variable.

**Message**

String to be displayed. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

**Return**

Variable to hold the result of this construct, which is the previously displayed string.

Note: Not available to Simulation scripts.
**Color Query**

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color:</td>
<td></td>
</tr>
<tr>
<td>Return:</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Assigns to the return variable the selected color specification. Using this construct, utility scripts can be written to assign element colors based upon a specific conditions (e.g., set the fill color for open issues to bright green, assign the fill color for all functions allocated to human Components to lavender).

**Color**

Color to be selected. Can be specified by selection from the drop-down on the right.

**Return**

Variable to hold the result of this construct.
### Comment

Comment allows the script author to add internal comments to document the script.

### Description

Comment allows the script author to add internal comments to document the script.

### Comment

An explanation of what is accomplished by following constructs, assumptions made, etc.
Expire Output

<table>
<thead>
<tr>
<th>Expiration Stamp</th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Description

Expire Output is used in a CORE2net script to tell a browser when to 'expire' the output page. Expire Output sets the ‘Expires’ flag within the HTTP header of the output document.

Expiration Stamp

A time stamp that ensures the latest database updates are shown in CORE2net output.
Open Transcript

*(No settings required)*

**Description**

Opens the report transcript. Note that this happens automatically if “Transcript” or “Both [File and Transcript]” is specified as the destination of an output construct or if an error is encountered.
Send Mail Message

**Arguments**

- **To:**
  - nil
  - Add
  - Remove

- **cc:**
  - nil
  - Add
  - Remove

- **bcc:**
  - nil
  - Add
  - Remove

- **Subject:**
  - nil
  - Add
  - Remove

- **Priority:**
  - nil
  - Normal

- **Body:**
  - nil
  - Attachments:
    - nil
    - Add
    - Remove

**Parameters**

**Description**

Allows a script to send an e-mail message. This could be used in a script to notify users of any database modifications after a specific date and time.

Requires that MAPI32.dll be enabled. If Microsoft mail options have been installed and configured, this .dll should be enabled. If CORE does not allow you to utilize the send feature contact your network system administrator to verify that this .dll is enabled.

**Addresses**

Ordered Collections of the To, cc, and bcc distribution addresses. Each collection can be specified by a variable from the drop-down on the left or a manual entry using the buttons on the right.

**Subject**

String containing the subject line of the message. Can be specified either by a variable from the drop-down on the left or a manual entry of the text on the right.

**Priority**

Symbol designating the delivery priority of the message. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

**Body**
String containing the text of the message. Can be specified either by a variable from the drop-down on the left or a manual entry of the text on the right.

**Attachments**
Ordered collection of full path names for files to be attached to the message. Can be specified either by a variable from the drop-down on the left or a manual identification of each file on the right.
**System Property Query**

<table>
<thead>
<tr>
<th>Property</th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Assigns to the *return* variable the value of the *property*.

**Property**

The system property to be returned. This field is a drop-down list with the following selections:

- **allReportScripts**: a dictionary of all reports scripts available to the CORE project. Each value is a string that is the full path for the scripts with a key that is a string containing the name of the script.
- **baselineSchemaName**: the name of the schema selected as the base schema when the current project was created.
- **commonScriptDirectory**: a string that is the full path for the directory of Common scripts. The value is context sensitive to whether it is executed from a report or CORE2net script.
- **commonScriptURI**: a string that is the Uniform Resource Identifier (URI) for the directory of Common CORE2net scripts.
- **currentScriptDirectory**: a string that is the full path for the directory of the script that is being executed.
- **date**: the current date.
- **mainScriptDirectory**: a string that is the full path for the directory of Main scripts. The value is context sensitive to whether it is executed from a report or CORE2net script.
- **mainScriptURI**: a string that is the Uniform Resource Identifier (URI) for the directory of Main CORE2net scripts.
- **outputDirectory**: the full path of the Output directory as set in User Preferences.
- **project**: the current project.
- **repositoryName**: a string that is the full path for the current repository file.
- **schemaVersion**: a hierarchical number which is the version (e.g., either 4.0 or 5.0) of the schema being used in the project.
- **selectedElement**: the element selection in the current window.
- **selectedElements**: the set of element selected in the current window.
- **selectedFolder**: the folder selected in the current window.
- **selectedPackage**: the package selected in the current window.
- **time**: the current time.
- **timeStamp**: the current time stamp (combination of the date and time).
- **user**: the current system user.
- **userAgent**: a user agent object that represents the browser in use in a CORE2net session. Note that this object responds to name (returns a string), isInternetExplorer (returns a Boolean), and isNetscape (returns a Boolean).
- **userScriptDirectory**: a string that is the full path for the directory of User scripts. The value is context sensitive to whether it is executed from a report or CORE2net script.
COREscript Constructs
Utilities

| userScriptURI | a string that is the Uniform Resource Identifier (URI) for the directory of User CORE2net scripts. |

Return
Variable to hold the specified system property. Unless otherwise stated, a system property is returned as an object.
VARIABLE MANIPULATION CONSTRUCTS

Database Object Sort

Description
Assigns to the return variable a new ordered collection containing all the same objects as are in the collection to be sorted. The objects in the new collection are ordered so that when substituted for the iteration variable in the specified expression, the value that results for each one is less than or equal to the value that results when the object in the next higher ordinal position within the new collection is substituted for the iteration variable in the specified expression. Note that the original collection is not modified.

Collection
Variable containing a collection of objects to be sorted. Can be of one of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection
- String
- Symbol

If the collection is a dictionary, the objects in the returned collection and the objects that are assigned to the iteration variable as described above are the values stored in the dictionary, not the keys whereby the values are accessed. If the collection is a string or symbol, this construct returns a collection of its characters in a new order.

Ascending
Boolean object indicating whether the collection should be sorted in ascending or descending order. If the argument value is false, this construct returns the objects in the collection in descending order. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Iteration Variable
String specifying the name of a variable referenced in the expression. This variable is declared within the scope of this construct but will not be accessible after this construct terminates.

Expression
Indicates the criterion to be used in sorting. An expression consists of operators in the CORE expression language, variables defined using the Edit Variables command under the Properties menu of the Report Editor, constants that can be coded in some data types, and the iteration variable described above.
COREscript Constructs
Variable Manipulation

Validate
Validates the syntax of the expression to ensure that neither operators nor operands appear in unexpected locations, all operators are recognized, and all variables have been defined. No type or bounds checking can be performed.

Return
Variable to hold the result of this construct, which is an ordered collection.
Element Sort

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection:</td>
<td>nil</td>
</tr>
<tr>
<td>Sort Block:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the return variable a new sorted collection containing all the same elements as are in the collection to be sorted. Elements in the new collection are ordered based upon the specified sort block. This construct differs from the Database Object Sort Construct in that the Element Sort Construct uses a pre-defined CORE sort block to sort the elements whereas the Database Object Sort Construct sorts the elements based upon an expression entered as a construct parameter.

Collection
Variable containing the collection of elements to be sorted. This collection can contain either elements from a single class or elements from multiple classes.

Sort Block
Sort block definition of a sort block defined in the CORE project. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right. This sort criteria continues to govern the behavior of the sorted collection that results from this construct.

Return
Variable to hold the sorted collection of elements.
Element Sort Blocks

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the *return* variable an alphabetically sorted collection containing all sort block definitions in the CORE project. This list contains both single class sort blocks and multi-class sort blocks. Therefore, it is very important that a standard naming convention be used when defining sort blocks in CORE. For example, if the names of all multi-class sort blocks have “by class” appended (e.g., Numeric by class), it is very easy to distinguish single class sort blocks from multi-class sort blocks.

Return
The variable to hold the sorted collection of all sort block definitions.
COREscript Constructs
Variable Manipulation

Filter

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection:</td>
<td>nil</td>
</tr>
<tr>
<td>Iteration Variable:</td>
<td></td>
</tr>
<tr>
<td>Expression</td>
<td></td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

Description
Assigns to the `return` variable a new collection of the same data type as the `collection` to be filtered (except that if the `collection` is a symbol, then a string is returned). Each object in the `collection` to be sorted is assigned, in turn, to the `iteration variable`, and, if the specified `expression` then evaluates to a Boolean `true`, the object is added to the new collection.

Collection
Variable containing the collection of objects to be filtered. Can be of one of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection
- String
- Symbol

Iteration Variable
String specifying the name of a variable referenced in the `expression`. This variable is declared within the scope of this construct but will not be accessible after this construct terminates.

If the `collection` to be filtered is a dictionary, then only the values stored in the dictionary will be assigned to the iteration variable, not the keys whereby the values are accessed. If the `collection` is a string or symbol, individual characters are assigned to the iteration variable.

Expression
Indicates the criterion to be used in filtering. An expression consists of operators in the CORE expression language, variables defined using the Edit Variables command under the Properties menu of the Script Editor, constants that can be coded in some data types, and the `iteration variable` described above. This expression must evaluate to a Boolean.

Validate
Validates the syntax of the `expression` to ensure that neither operators nor operands appear in unexpected locations, all operators are recognized, and all variables have been defined. No type or bounds checking can be performed.
Return

Variable to hold the result of this construct. It is of the same data type as the collection to be filtered (except that if the collection is a symbol, then a string is returned) and contains a subset of the objects therein. If the collections are ordered or sorted, the objects they contain maintain their relative positions.

If the collection is a sorted collection, then the same sort criterion will be maintained by the returned collection. If the collection is a dictionary, each filtered value will be associated with the same key.
COREscript Constructs
Variable Manipulation

Random Number

<table>
<thead>
<tr>
<th>Argument/Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribution:</td>
<td>A symbol naming the distribution. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.</td>
</tr>
<tr>
<td>Mean:</td>
<td>The first parameter (mean, N, lower bound, …) required for the specified distribution. The data type is dependent upon the distribution specified. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.</td>
</tr>
<tr>
<td>Standard Deviation:</td>
<td>If required, the second parameter (standard deviation, beta, …) for the specified distribution. The data type is dependent upon the distribution specified. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.</td>
</tr>
<tr>
<td>Stream:</td>
<td>An integer indicating which of the 99 available random number streams to supply the seed value for the specified distribution. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.</td>
</tr>
<tr>
<td>Return:</td>
<td>Variable to hold the result of this construct, which is a number.</td>
</tr>
</tbody>
</table>

Description

Assigns to the return variable a random number drawn from the specified random distribution using the provided parameters. This construct provides access to the 19 random number distributions supported by CORE and COREsim. Access to random numbers is particularly useful when authoring COREsim scripts to randomly select exit paths, induce behavior, etc.

When a distribution is selected from the drop-down list, the labels and fields of this construct transform to represent the specific parameters and data types for the selected distribution. For example, the Normal distribution requires a mean and a standard deviation, both of type float. The ChiSquared distribution requires only a single parameter (N) of type integer. A discussion of the random number distributions can be found in Appendix A of the COREsim User Guide.
### Report Section

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>nil</td>
</tr>
<tr>
<td>Number:</td>
<td>nil</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

#### Description

Assigns to the *return* variable a new object of the report section data type. One possible use of this construct is as follows:

This construct is executed multiple times at the beginning of a report script to create different report sections, and each resultant object is added to a collection. A Multiple Selection Prompt is then opened on the collection, allowing the user to choose a subset of the report sections. An iterate construct iterates over the selected report sections and builds a dictionary from them using a simple data type such as a symbol for keys. The script language statements corresponding to each section of the report can be placed in "if-then" constructs, using the presence of the associate key in the dictionary as the condition for execution.

#### Name

A string naming the new report section. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

#### Number

A hierarchical number according to which report sections can be sorted. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

#### Return

Variable to hold the result of this construct, which is a report section.
Report Setting

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting:</td>
<td>Prepared By Name</td>
</tr>
<tr>
<td>Return:</td>
<td>nil</td>
</tr>
</tbody>
</table>

**Description**

Assigns to the `return` variable the value of the selected `setting` property from the report settings dialog (e.g., Prepared By Name).

**Setting**

Name of the property which value is to be returned. It is one of the following strings: “Prepared By Address,” “Prepared By Name,” “Prepared For Address,” or “Prepared For Name.”

**Return**

Variable to hold the result of this construct.

Note: Not available to Simulation scripts.
COREscript Constructs
Variable Manipulation

Sort

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection: nil</td>
<td></td>
</tr>
<tr>
<td>Ascending: nil</td>
<td></td>
</tr>
<tr>
<td>Return: nil</td>
<td></td>
</tr>
</tbody>
</table>

Description
Assigns to the return variable a new, sorted ordered collection containing all the same objects as are in the collection to be sorted. Note that the original collection is not modified.

Collection
Variable containing a collection of objects to be sorted. Can be of one of the following data types:
- Array
- Bag
- Dictionary
- Ordered Collection
- Set
- Sorted Collection
- String
- Symbol

If the collection is a dictionary, the objects in the returned collection are the values stored in the dictionary, not the keys whereby the values are accessed. If the collection is a string or symbol, this construct returns a collection of its characters in a new order.

Ascending
Boolean object indicating whether the collection should be sorted in ascending or descending order. If the argument value is false, this construct returns the objects in the collection in descending order. Can be specified either by a variable from the drop-down on the left or a manual selection from the drop-down on the right.

Return
Variable to hold the result of this construct, which is an ordered collection.
Table Definitions

<table>
<thead>
<tr>
<th>Return:</th>
<th>Arguments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>nil</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Description**
Assigns to the `return` variable an alphabetically sorted collection containing all table definitions in the CORE project. This collection can be used in conjunction with the Table Output construct to quickly include a table in a report.

**Return**
The variable to hold the sorted collection of all table definitions.
Variable Assignment

Description
Assigns an object to the *return* variable. The object assigned can be a new and empty collection of one of the available data types or it can be the result of a CORE expression.

Expression
An expression consisting of operators in the CORE expression language, variables defined using the Edit Variables command under the Properties menu of the Script Editor, and constants that can be coded in some data types.

Validate
Validates the syntax of the expression to ensure that neither operators nor operands appear in unexpected locations, all operators are recognized, and all variables have been defined. No type or bounds checking can be performed.

Data Type
This construct can create new objects of the following types:
- Array
- Dictionary
- Ordered Collection
- Set
- Table

Arrays and tables have fixed dimensions that are also defined by this construct.

Size
The number of indices allocated to the returned array if the Array *data type* is selected. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Rows
The number of rows allocated in the returned table if the Table *data type* is selected. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.
COREscript Constructs
Variable Manipulation

Columns
The number of columns allocated in the returned table if the Table data type is selected. Can be specified either by a variable from the drop-down on the left or manual entry in the field on the right.

Return
Variable to hold the result of this construct.
CONSTRUCT INDEX

Add Element To Folder .................................. 27
At End.................................................. 71
Attribute Definition Property .................. 105
Available Diagrams Query .................. 129
Beep.................................................. 130
Block Construct.................................. 1
Breakpoint ........................................ 131
Cancel Report .................................... 132
Change Indicator .................................. 133
Children Query .................................... 13
Class Attributes .................................. 106
Class Parameters .................................. 107
Class Property .................................... 108
Class Query......................................... 109
Class Relations .................................... 110
Class Target Classes ................................ 111
Close File ......................................... 45
Color Query ........................................ 134
Comma-Delimited Output .................. 53
Comment............................................. 135
Composite Selection Prompt .................. 91
Database Classes .................................. 112
Database Object Sort ......................... 143
Delete Object ....................................... 28
Diagnostic Results Query .................... 14
Diagram File Output ......................... 54
Diagram Output .................................. 57
Disabled Code Construct ...................... 2
Drop-Down Selection Prompt .................. 92
Element Attribute Query ..................... 15
Element Output .................................... 60
Element Selection Prompt ..................... 93
Element Sort ....................................... 145
Element Sort Blocks ......................... 146
Elements Defined Parameters Query ......... 16
Elements Parameter Units Query ............. 17
Elements Parameter Value Query ............. 18
Elements Query .................................... 19
ExpireOutputs ..................................... 136
Facilities .......................................... 113
Facility Database Classes ..................... 114
Facility Property .................................. 115
Facility Query ..................................... 116
Facility Schema Classes ....................... 117
Filter ............................................. 147
Get Position ....................................... 72
Hierarchy Iterate Construct .................. 3
If Then Construct.................................. 5
If Then Else Construct ......................... 6
Include External Boilerplate .......... 63
Include External Graphic .................. 64
Include Script ..................................... 7
Include Style Sheet ......................... 65
Iterate Construct .................................. 8
Lock Object ........................................ 29
Message Prompt .................................... 95
Move Element To Folder ....................... 30
Multiple Selection Prompt .................... 96
Named Element Query ......................... 20
New Element ...................................... 31
New Folder ........................................ 32
New Relationship .................................. 33
Next.................................................. 73
Open File Dialog .................................. 46
Open Input File .................................... 47
Open Output File .................................. 48
Open Temporary Output File .................. 49
Open Transcript ................................... 137
Parameter Definition Property ............... 118
Parents Query ..................................... 21
Peek.................................................. 74
Project Explorer Prompt ....................... 98
Project Property ................................... 81
Project Query ....................................... 83
Project Registry Query ......................... 84
Projects ............................................ 85
Prompt Separator .................................. 99
Purge Attribute Versions ..................... 34
Random Number ................................... 149
RDToolkit Output .................................. 66
Relation Attributes ................................ 119
Relation Property .................................. 120
Relation Query ..................................... 121
Relations ........................................... 122
Relationship Attribute Query ............... 22
Relationships Query ............................... 23
Remove Element From Folder ............... 35
Remove Project Registry Key ............... 86
Rename Element .................................... 36
Rename Folder ..................................... 37
Renumber Element ................................ 38
Repeat Until Construct ......................... 9
Report Section .................................... 150
Report Setting ..................................... 151
Report Settings Prompt ....................... 100
Resource Level Query ............................ 125
Return ............................................. 10
Root Folder Query .................................. 24
Save File Dialog ................................. 50
Schema Classes .................................... 123
Script Context Query ......................... 126
Send Mail Message ......................... 138
<table>
<thead>
<tr>
<th>COREscript Constructs</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set Default Input File</td>
<td>51</td>
</tr>
<tr>
<td>Set Default Output File</td>
<td>52</td>
</tr>
<tr>
<td>Set Element Attribute</td>
<td>39</td>
</tr>
<tr>
<td>Set Element Diagnostic Results</td>
<td>40</td>
</tr>
<tr>
<td>Set Element Parameter Units</td>
<td>41, 42</td>
</tr>
<tr>
<td>Set Position</td>
<td>75</td>
</tr>
<tr>
<td>Set Project</td>
<td>87</td>
</tr>
<tr>
<td>Set Project Registry Value</td>
<td>88</td>
</tr>
<tr>
<td>Set Style</td>
<td>67</td>
</tr>
<tr>
<td>Simulator Property Query</td>
<td>127</td>
</tr>
<tr>
<td>Single Selection Prompt</td>
<td>101</td>
</tr>
<tr>
<td>Size</td>
<td>76</td>
</tr>
<tr>
<td>Skip</td>
<td>77</td>
</tr>
<tr>
<td>Sort</td>
<td>152</td>
</tr>
<tr>
<td>System Property Query</td>
<td>140</td>
</tr>
<tr>
<td>Tag Output</td>
<td>68</td>
</tr>
<tr>
<td>Targets Query</td>
<td>25</td>
</tr>
<tr>
<td>Text Output</td>
<td>69</td>
</tr>
<tr>
<td>Unlock Object</td>
<td>44</td>
</tr>
<tr>
<td>Up To</td>
<td>78</td>
</tr>
<tr>
<td>Update Block Construct</td>
<td>11</td>
</tr>
<tr>
<td>Value Prompt</td>
<td>102</td>
</tr>
<tr>
<td>Variable Assignment</td>
<td>153</td>
</tr>
<tr>
<td>Variable Output</td>
<td>70</td>
</tr>
<tr>
<td>While Do Construct</td>
<td>12</td>
</tr>
<tr>
<td>YesOrNo Prompt</td>
<td>103</td>
</tr>
</tbody>
</table>