

globus rsl Reference Manual

9.1

Generated by Doxygen 1.4.7

Thu Nov 21 15:18:29 2013

Contents

1	globus rsl Main Page	1
2	globus rsl Module Index	1
3	globus rsl Module Documentation	2

1 globus rsl Main Page

The Globus RSL library is provides the following functionality:

- **RSL Predicates** (p. 2)
- **RSL Constructors** (p. 6)
- **RSL Memory Management** (p. 9)
- **RSL Accessor Functions** (p. 12)
- **RSL Value Accessors** (p. 17)
- **RSL Display** (p. 20)
- **RSL Parsing** (p. 22)
- **List Functions** (p. 17)

2 globus rsl Module Index

2.1 globus rsl Modules

Here is a list of all modules:

RSL Predicates	2
RSL Constructors	6
RSL Memory Management	9
RSL Accessor Functions	12
List Functions	17
RSL Value Accessors	17
RSL Display	20
RSL Helper Functions	21
RSL Parsing	22

3 globus_rsl Module Documentation

3.1 RSL Predicates

The functions in this group return boolean values indicating whether an RSL syntax tree is of a particular type.

Functions

- `int globus_rsl_is_relation (globus_rsl_t *ast)`
- `int globus_rsl_is_boolean (globus_rsl_t *ast)`
- `int globus_rsl_is_relation_eq (globus_rsl_t *ast)`
- `int globus_rsl_is_relation_lessthan (globus_rsl_t *ast)`
- `int globus_rsl_is_relation_attribute_equal (globus_rsl_t *ast, char *attribute)`
- `int globus_rsl_is_boolean_and (globus_rsl_t *ast)`
- `int globus_rsl_is_boolean_or (globus_rsl_t *ast)`
- `int globus_rsl_is_boolean_multi (globus_rsl_t *ast)`
- `int globus_rsl_value_is_literal (globus_rsl_value_t *ast)`
- `int globus_rsl_value_is_sequence (globus_rsl_value_t *ast)`
- `int globus_rsl_value_is_variable (globus_rsl_value_t *ast)`
- `int globus_rsl_value_is_concatenation (globus_rsl_value_t *ast)`

3.1.1 Detailed Description

The functions in this group return boolean values indicating whether an RSL syntax tree is of a particular type.

3.1.2 Function Documentation

3.1.2.1 `int globus_rsl_is_relation (globus_rsl_t *ast)`

RSL relation test.

The `globus_rsl_is_relation()` (p. 2) function tests whether the the RSL pointed to by the *ast* parameter is a relation. The RSL syntax supports the following relation operations:

`=` Equal

`!=` Not Equal

`>` Greater Than

`>=` Greater Than or Equal

`<` Less Than

`<=` Less Than or Equal

`<=` Less Than or Equal

Some examples of RSL relations are

```
"queue" = "debug"
"queue" != "slow"
"min_memory" > "1000"
"max_wall_time" >= "60"
"count" < "10"
"host_count" <= "5"
```

GRAM only supports equality relations.

Parameters:

ast Pointer to an RSL parse tree structure.

Returns:

The **globus_rsl_is_relation()** (p. 2) function returns GLOBUS_TRUE if the RSL parse tree pointed to by *ast* is a relation; otherwise, it returns GLOBUS_FALSE.

3.1.2.2 int globus_rsl_is_boolean (globus_rsl_t * *ast*)

RSL boolean test.

The **globus_rsl_is_boolean()** (p. 3) function tests whether the the RSL pointed to by the *ast* parameter is a boolean composition of other RSL parse trees. The syntactically understood boolean compositions are "&" (conjunction), "|" (disjunction), and "+" (multi-request). Some bexamples of RSL booleans are

```
& ( "queue" = "debug" ) ( "max_time" = "10000" )  
| ( "count" = "1" ) ( "count" = "10" )  
+ ( & ( "executable" = "1.exe" ) ) ( & ( "executable" = "2.exe" ) )
```

Parameters:

ast Pointer to an RSL parse tree structure.

Returns:

The **globus_rsl_is_boolean()** (p. 3) function returns GLOBUS_TRUE if the RSL parse tree pointed to by *ast* is a boolean composition; otherwise, it returns GLOBUS_FALSE.

3.1.2.3 int globus_rsl_is_relation_eq (globus_rsl_t * *ast*)

RSL equality operation test.

The **globus_rsl_is_relation_eq()** (p. 3) function tests whether the the RSL pointed to by the *ast* parameter is an equality relation. An example of an equality relation is

```
"queue" = "debug"
```

Parameters:

ast Pointer to an RSL parse tree structure.

Returns:

The **globus_rsl_is_relation_eq()** (p. 3) function returns GLOBUS_TRUE if the RSL parse tree pointed to by *ast* is an equality relation; otherwise, it returns GLOBUS_FALSE.

3.1.2.4 int globus_rsl_is_relation_less than (globus_rsl_t * *ast*)

RSL less than operation test.

The **globus_rsl_is_relation_less than()** (p. 3) function tests whether the the RSL pointed to by the *ast* parameter is a less-than relation. An example of a less-than relation is

```
"count" = "10"
```

Parameters:

ast Pointer to an RSL parse tree structure.

Returns:

The **globus_rsl_is_relation_lessthan()** (p. 3) function returns GLOBUS_TRUE if the RSL parse tree pointed to by *ast* is a less-than relation; otherwise, it returns GLOBUS_FALSE.

3.1.2.5 int globus_rsl_is_relation_attribute_equal (globus_rsl_t * *ast*, char * *attribute*)

RSL attribute name test.

The **globus_rsl_is_relation_attribute_equal()** (p. 4) function tests whether the the RSL pointed to by the *ast* parameter is a relation with the attribute name which matches the string pointed to by the *attribute* parameter. This attribute name comparison is case-insensitive.

Parameters:

ast Pointer to an RSL parse tree structure.

attribute Name of the attribute to test

Returns:

The **globus_rsl_is_relation_attribute_equal()** (p. 4) function returns GLOBUS_TRUE if the RSL parse tree pointed to by *ast* is a relation and its attribute name matches the *attribute* parameter; otherwise, it returns GLOBUS_FALSE.

3.1.2.6 int globus_rsl_is_boolean_and (globus_rsl_t * *ast*)

RSL boolean and test.

The **globus_rsl_is_boolean_and()** (p. 4) function tests whether the the RSL pointed to by the *ast* parameter is a boolean "and" composition of RSL trees. An example of a boolean and relation is

```
& ( "queue" = "debug" ) ( "executable" = "a.out" )
```

Parameters:

ast Pointer to an RSL parse tree structure.

Returns:

The **globus_rsl_is_boolean_and()** (p. 4) function returns GLOBUS_TRUE if the RSL parse tree pointed to by *ast* is a boolean and of RSL parse trees; otherwise, it returns GLOBUS_FALSE.

3.1.2.7 int globus_rsl_is_boolean_or (globus_rsl_t * *ast*)

RSL boolean or test.

The **globus_rsl_is_boolean_or()** (p. 4) function tests whether the the RSL pointed to by the *ast* parameter is a boolean "or" composition of RSL trees. An example of a boolean or relation is

```
| ( "count" = "2" ) ( "count" = "4" )
```

Parameters:

ast Pointer to an RSL parse tree structure.

Returns:

The **globus_rsl_is_boolean_or()** (p. 4) function returns GLOBUS_TRUE if the RSL parse tree pointed to by *ast* is a boolean and of RSL parse trees; otherwise, it returns GLOBUS_FALSE.

3.1.2.8 int globus_rsl_is_boolean_multi (globus_rsl_t * ast)

RSL boolean multi test.

The **globus_rsl_is_boolean_multi()** (p. 5) function tests whether the the RSL pointed to by the *ast* parameter is a boolean "multi-request" composition of RSL trees. An example of a boolean multi-request relation is

```
+ ( &( "executable" = "exe.1") ( "count" = "2" ) )
  ( &( "executable" = " exe.2") ( "count" = "2" ) )
```

Parameters:

ast Pointer to an RSL parse tree structure.

Returns:

The **globus_rsl_is_boolean_multi()** (p. 5) function returns GLOBUS_TRUE if the RSL parse tree pointed to by *ast* is a boolean multi-request of RSL parse trees; otherwise, it returns GLOBUS_FALSE.

3.1.2.9 int globus_rsl_value_is_literal (globus_rsl_value_t * ast)

RSL literal string test.

The **globus_rsl_value_is_literal()** (p. 5) function tests whether the the RSL value pointed to by the *ast* parameter is a literal string value. An example of a literal string is

```
"count"
```

Parameters:

ast Pointer to an RSL value structure.

Returns:

The **globus_rsl_value_is_literal()** (p. 5) function returns GLOBUS_TRUE if the RSL value pointed to by *ast* is a literal string value; otherwise, it returns GLOBUS_FALSE.

3.1.2.10 int globus_rsl_value_is_sequence (globus_rsl_value_t * ast)

RSL value sequence test.

The **globus_rsl_value_is_sequence()** (p. 5) function tests whether the the RSL value pointed to by the *ast* parameter is a sequence of RSL values. An example of a sequence of values is

```
"1" "2" "3"
```

Parameters:

ast Pointer to an RSL value structure.

Returns:

The **globus_rsl_value_is_sequence()** (p. 5) function returns GLOBUS_TRUE if the RSL value pointed to by *ast* is a value sequence; otherwise, it returns GLOBUS_FALSE.

3.1.2.11 int globus_rsl_value_is_variable (globus_rsl_value_t * ast)

RSL value variable test.

The **globus_rsl_value_is_variable()** (p. 6) function tests whether the the RSL value pointed to by the *ast* parameter is a variable reference. RSL values. An example of a variable reference is

```
$ ( "GLOBUSRUN_GASS_URL" )
```

Parameters:

ast Pointer to an RSL value structure.

Returns:

The **globus_rsl_value_is_sequence()** (p. 5) function returns GLOBUS_TRUE if the RSL value pointed to by *ast* is a value sequence; otherwise, it returns GLOBUS_FALSE.

3.1.2.12 int globus_rsl_value_is_concatenation (globus_rsl_value_t * ast)

RSL value concatenation test.

The **globus_rsl_value_is_concatenation()** (p. 6) function tests whether the the RSL value pointed to by the *ast* parameter is a concatenation of RSL values. An example of an RSL value concatenation is

```
$ ( "GLOBUSRUN_GASS_URL" ) # "/input"
```

Parameters:

ast Pointer to an RSL value structure.

Returns:

The **globus_rsl_value_is_concatenation()** (p. 6) function returns GLOBUS_TRUE if the RSL value pointed to by *ast* is a value concatenation; otherwise, it returns GLOBUS_FALSE.

3.2 RSL Constructors

Functions

- globus_rsl_t * globus_rsl_make_boolean (int operator, globus_list_t *children)
- globus_rsl_t * globus_rsl_make_relation (int operator, char *attributename, globus_rsl_value_t *value_sequence)
- globus_rsl_value_t * globus_rsl_value_make_literal (char *string)
- globus_rsl_value_t * globus_rsl_value_make_sequence (globus_list_t *value_list)
- globus_rsl_value_t * globus_rsl_value_make_variable (globus_rsl_value_t *sequence)
- globus_rsl_value_t * globus_rsl_value_make_concatenation (globus_rsl_value_t *left_value, globus_rsl_value_t *right_value)

3.2.1 Function Documentation

3.2.1.1 globus_rsl_t* globus_rsl_make_boolean (int operator, globus_list_t * children)

RSL boolean constructor.

The **globus_rsl_make_boolean()** (p. 6) function creates a boolean composition of the RSL nodes in the list pointed to by *children*. The new RSL node which is returned contains a reference to the list, not a copy.

Parameters:

operator The boolean RSL operator to use to join the RSL parse tree list pointed to by the *children* parameter. This value must be one of GLOBUS_RSL_AND, GLOBUS_RSL_OR, GLOBUS_RSL_MULTIREQ in order to create a valid RSL tree.

children Pointer to a list of RSL syntax trees to combine with the boolean operation described by the *operator* parameter.

Returns:

The **globus_rsl_make_boolean()** (p. 6) function returns a new RSL parse tree node that contains a shallow reference to the list of values pointed to by the *children* parameter joined by the operator value in the *operator* parameter. If an error occurs, **globus_rsl_make_boolean()** (p. 6) returns NULL.

3.2.1.2 globus_rsl_t* globus_rsl_make_relation (int operator, char * attributename, globus_rsl_value_t * value_sequence)

RSL relation constructor.

The **globus_rsl_make_relation()** (p. 7) function creates a relation between the attribute named by the *attributename* parameter and the values pointed to by the *value_sequence* list. The new RSL relation node which is returned contains a reference to the *attributename* and *value_sequence* parameters, not a copy.

Parameters:

operator The RSL operator to use to relate the RSL attribute name pointed to by the *attributename* parameter and the values pointed to by the *value_sequence* parameter. This value must be one of GLOBUS_RSL_EQ, GLOBUS_RSL_NEQ, GLOBUS_RSL_GT, GLOBUS_RSL_GTEQ, GLOBUS_RSL_LT, or GLOBUS_RSL_LTEQ in order to create a valid RSL node.

attributename Pointer to a string naming the attribute of the new RSL relation.

value_sequence Pointer to a sequence of RSL values to use in the new RSL relation.

Returns:

The **globus_rsl_make_relation()** (p. 7) function returns a new RSL parse tree node that contains a shallow reference to the attribute name pointed to by the *attributename* parameter and the RSL value sequence pointed to by the *value_sequence* parameter. If an error occurs, **globus_rsl_make_relation()** (p. 7) returns NULL.

3.2.1.3 globus_rsl_value_t* globus_rsl_value_make_literal (char * string)

RSL literal constructor.

The **globus_rsl_value_make_literal()** (p. 7) function creates a string literal RSL value node containing the value pointed to by the *string* parameter. The new RSL value node which is returned contains a reference to the *string* parameter, not a copy.

Parameters:

string The literal string to be used in the new value.

Returns:

The **globus_rsl_value_make_literal()** (p. 7) function returns a new RSL value node that contains a shallow reference to the string pointed to by the *string* parameter. If an error occurs, **globus_rsl_value_make_literal()** (p. 7) returns NULL.

3.2.1.4 **globus_rsl_value_t* globus_rsl_value_make_sequence (globus_list_t * *value_list*)**

RSL value sequence constructor.

The **globus_rsl_value_make_sequence()** (p. 8) function creates a value sequence RSL node referring to the values pointed to by the *value_list* parameter. The new node returned by this function contains a reference to the *value_list* parameter, not a copy.

Parameters:

value_list A pointer to a list of globus_rsl_value_t pointers.

Returns:

The **globus_rsl_value_make_sequence()** (p. 8) function returns a new RSL value node that contains a shallow reference to the list pointed to by the *value_list* parameter. If an error occurs, **globus_rsl_value_make_sequence()** (p. 8) returns NULL.

3.2.1.5 **globus_rsl_value_t* globus_rsl_value_make_variable (globus_rsl_value_t * *sequence*)**

RSL variable reference constructor.

The **globus_rsl_value_make_variable()** (p. 8) function creates a variable reference RSL node referring to the variable name contained in the value pointed to by *sequence* parameter. The new node returned by this function contains a reference to the *sequence* parameter, not a copy.

Parameters:

sequence A pointer to a RSL value sequence.

Returns:

The **globus_rsl_value_make_variable()** (p. 8) function returns a new RSL value node that contains a shallow reference to the value sequence pointed to by the *sequence* parameter. If an error occurs, **globus_rsl_value_make_variable()** (p. 8) returns NULL.

3.2.1.6 **globus_rsl_value_t* globus_rsl_value_make_concatenation (globus_rsl_value_t * *left_value*, globus_rsl_value_t * *right_value*)**

RSL concatenation constructor.

The **globus_rsl_value_make_concatenation()** (p. 8) function creates a concatenation of the values pointed to by the *left_value* and *right_value* parameters. The new node returned by this function contains a reference to these parameters' values, not a copy.

Parameters:

left_value A pointer to a RSL value to act as the left side of the concatenation. This must be a string literal or variable reference.

right_value A pointer to a RSL value to act as the right side of the concatenation. This must be a string literal or variable reference.

Returns:

The **globus_rsl_value_make_concatenation()** (p. 8) function returns a new RSL value node that contains a shallow reference to the values pointed to by the *left_value* and *right_value* parameters. If an error occurs, **globus_rsl_value_make_concatenation()** (p. 8) returns NULL.

3.3 RSL Memory Management

Functions

- `globus_rsl_t * globus_rsl_copy_recursive (globus_rsl_t *ast_node)`
- `globus_rsl_value_t * globus_rsl_value_copy_recursive (globus_rsl_value_t *globus_rsl_value_ptr)`
- `int globus_rsl_value_free (globus_rsl_value_t *val)`
- `int globus_rsl_free (globus_rsl_t *ast_node)`
- `int globus_rsl_value_free_recursive (globus_rsl_value_t *globus_rsl_value_ptr)`
- `int globus_rsl_free_recursive (globus_rsl_t *ast_node)`
- `int globus_rsl_value_list_literal_replace (globus_list_t *value_list, char *string_value)`
- `int globus_rsl_value_eval (globus_rsl_value_t *ast_node, globus_symboltable_t *symbol_table, char **string_value, int rsl_substitution_flag)`
- `int globus_rsl_eval (globus_rsl_t *ast_node, globus_symboltable_t *symbol_table)`

3.3.1 Function Documentation

3.3.1.1 `globus_rsl_t* globus_rsl_copy_recursive (globus_rsl_t *ast_node)`

Create a deep copy of an RSL syntax tree.

The `globus_rsl_copy_recursive()` (p. 9) function performs a deep copy of the RSL syntax tree pointed to by the *ast_node* parameter. All RSL nodes, value nodes, variable names, attributes, and literals will be copied to the return value.

Parameters:

ast_node An RSL syntax tree to copy.

Returns:

The `globus_rsl_copy_recursive()` (p. 9) function returns a copy of its input parameter that that can be used after the *ast_node* and its values have been freed. If an error occurs, `globus_rsl_copy_recursive()` (p. 9) returns NULL.

3.3.1.2 `globus_rsl_value_t* globus_rsl_value_copy_recursive (globus_rsl_value_t *globus_rsl_value_ptr)`

Create a deep copy of an RSL value.

The `globus_rsl_value_copy_recursive()` (p. 9) function performs a deep copy of the RSL value pointed to by the *globus_rsl_value_ptr* parameter. All variable names, attributes, literals, and value lists will be copied to the return value.

Parameters:

globus_rsl_value_ptr A pointer to an RSL value to copy.

Returns:

The `globus_rsl_value_copy_recursive()` (p. 9) function returns a copy of its input parameter that that can be used after the *globus_rsl_value_ptr* and its values have been freed. If an error occurs, `globus_rsl_value_copy_recursive()` (p. 9) returns NULL.

3.3.1.3 `int globus_rsl_value_free (globus_rsl_value_t * val)`

Free an RSL value node.

The `globus_rsl_value_free()` (p. 10) function frees the RSL value pointed to by the *val* parameter. This only frees the RSL value node itself, and not any sequence or string values associated with that node.

Parameters:

val The RSL value node to free.

Returns:

The `globus_rsl_value_free()` (p. 10) function always returns `GLOBUS_SUCCESS`.

3.3.1.4 `int globus_rsl_free (globus_rsl_t * ast_node)`

Free an RSL syntax tree node.

The `globus_rsl_free()` (p. 10) function frees the RSL syntax tree node pointed to by the *ast_node* parameter. This only frees the RSL syntax tree node itself, and not any boolean operands, relation names, or values associated with the node.

Parameters:

ast_node The RSL syntax tree node to free.

Returns:

The `globus_rsl_value_free()` (p. 10) function always returns `GLOBUS_SUCCESS`.

3.3.1.5 `int globus_rsl_value_free_recursive (globus_rsl_value_t * globus_rsl_value_ptr)`

Free an RSL value and all its child nodes.

The `globus_rsl_free_recursive()` (p. 10) function frees the RSL value node pointed to by the *globus_rsl_value_ptr*, including all literal strings, variable names, and value sequences. Any pointers to these are no longer valid after `globus_rsl_value_free_recursive()` (p. 10) returns.

Parameters:

globus_rsl_value_ptr An RSL value node to free.

Returns:

The `globus_rsl_value_free_recursive()` (p. 10) function always returns `GLOBUS_SUCCESS`.

3.3.1.6 `int globus_rsl_free_recursive (globus_rsl_t * ast_node)`

Free an RSL syntax tree and all its child nodes.

The `globus_rsl_free_recursive()` (p. 10) function frees the RSL syntax tree pointed to by the *ast_node* parameter, including all boolean operands, attribute names, and values. Any pointers to these are no longer valid after `globus_rsl_free_recursive()` (p. 10) returns.

Parameters:

ast_node An RSL parse tree to free.

Returns:

The `globus_rsl_value_free_recursive()` (p. 10) function always returns `GLOBUS_SUCCESS`.

3.3.1.7 int globus_rsl_value_list_literal_replace (globus_rsl_t * value_list, char * string_value)

Replace the first value in a value list with a literal.

The **globus_rsl_value_list_literal_replace()** (p. 11) function replaces the first value in the list pointed to by the *value_list* parameter with a new value node that is a literal string node pointing to the value of the *string_value* parameter, freeing the old value.

Parameters:

value_list The RSL value list to modify by replacing its first element.

string_value The new string value to use as a literal first element of the list pointed to by the *value_list* parameter.

Returns:

Upon success, **globus_rsl_value_list_literal_replace()** (p. 11) returns *GLOBUS_SUCCESS*, frees the current first value of *value_list* and replaces it with a new literal string node pointing to the value of the *string_value* parameter. If an error occurs, **globus_rsl_value_list_literal_replace()** (p. 11) returns 1.

3.3.1.8 int globus_rsl_value_eval (globus_rsl_value_t * ast_node, globus_symboltable_t * symbol_table, char ** string_value, int rsl_substitution_flag)

Evaluate RSL substitutions in an RSL value node.

The **globus_rsl_value_eval()** (p. 11) function modifies the value pointed to by its *ast_node* parameter by replacing all RSL substitution variable reference nodes with the literal values those variables evaluate to based on the current scope of the symbol table pointed to by the *symbol_table* parameter. It also combines string concatenations into literal string values. Any nodes which are replaced by this function are freed using **globus_rsl_value_free_recursive()** (p. 10).

Parameters:

ast_node A pointer to the RSL value node to evaluate.

symbol_table A symbol table containing current definitions of the RSL substitutions which can occur in this evaluation scope.

string_value An output parameter which is set to point to the value of the string returned by evaluating the value node pointed to by *ast_node* if it evaluates to a literal value. list pointed to by the *value_list* parameter.

rsl_substitution_flag A flag indicating whether the node pointed to by the *ast_node* parameter defines RSL substitution variables.

Returns:

Upon success, **globus_rsl_value_eval()** (p. 11) returns *GLOBUS_SUCCESS*, and replaces any RSL substitution values in the node pointed to by the *ast_node* parameter. If the node evaluates to a single literal, the *string_value* parameter is modified to point to the value of that literal. If an error occurs, **globus_rsl_value_eval()** (p. 11) returns a non-zero value.

3.3.1.9 int globus_rsl_eval (globus_rsl_t * ast_node, globus_symboltable_t * symbol_table)

Evaluate an RSL syntax tree.

The **globus_rsl_eval()** (p. 11) function modifies the RSL parse tree pointed to by its *ast_node* parameter by replacing all RSL substitution variable reference nodes with the literal values those variables evaluate to based on the current scope of the symbol table pointed to by the *symbol_table* parameter. It also combines string concatenations into literal string values. Any nodes which are replaced by this function are freed using **globus_rsl_value_free_recursive()** (p. 10).

Parameters:

ast_node A pointer to the RSL syntax tree to evaluate.

symbol_table A symbol table containing current definitions of the RSL substitutions which can occur in this evaluation scope.

Returns:

Upon success, **globus_rsl_eval()** (p. 11) returns *GLOBUS_SUCCESS*, and replaces all RSL substitution values and concatenations in *ast_node* or its child nodes with the evaluated forms described above. If an error occurs, **globus_rsl_eval()** (p. 11) returns a non-zero value.

3.4 RSL Accessor Functions

Functions

- **int globus_rsl_boolean_get_operator** (globus_rsl_t *ast_node)
- **globus_list_t * globus_rsl_boolean_get_operand_list** (globus_rsl_t *ast_node)
- **globus_list_t ** globus_rsl_boolean_get_operand_list_ref** (globus_rsl_t *boolean_node)
- **char * globus_rsl_relation_get_attribute** (globus_rsl_t *ast_node)
- **int globus_rsl_relation_get_operator** (globus_rsl_t *ast_node)
- **globus_rsl_value_t * globus_rsl_relation_get_value_sequence** (globus_rsl_t *ast_node)
- **globus_rsl_value_t * globus_rsl_relation_get_single_value** (globus_rsl_t *ast_node)
- **char * globus_rsl_value_literal_get_string** (globus_rsl_value_t *literal_node)
- **globus_list_t * globus_rsl_value_sequence_get_value_list** (globus_rsl_value_t *sequence_node)
- **globus_rsl_value_t * globus_rsl_value_variable_get_sequence** (globus_rsl_value_t *variable_node)
- **char * globus_rsl_value_variable_get_name** (globus_rsl_value_t *variable_node)
- **char * globus_rsl_value_variable_get_default** (globus_rsl_value_t *variable_node)
- **int globus_rsl_value_variable_get_size** (globus_rsl_value_t *variable_node)
- **globus_rsl_value_t * globus_rsl_value_concatenation_get_left** (globus_rsl_value_t *concatenation_node)
- **globus_rsl_value_t * globus_rsl_value_concatenation_get_right** (globus_rsl_value_t *concatenation_node)
- **globus_list_t ** globus_rsl_value_sequence_get_list_ref** (globus_rsl_value_t *sequence_node)

3.4.1 Function Documentation

3.4.1.1 **int globus_rsl_boolean_get_operator** (globus_rsl_t * *ast_node*)

Get the RSL operator used in a boolean RSL composition.

The **globus_rsl_boolean_get_operator()** (p. 12) function returns the operator that is used by the boolean RSL composition.

Parameters:

ast_node The RSL syntax tree to inspect.

Returns:

Upon success, **globus_rsl_boolean_get_operator()** (p. 12) returns one of *GLOBUS_RSL_AND*, *GLOBUS_RSL_OR*, *GLOBUS_RSL_MULTIREQ*. If an error occurs, **globus_rsl_boolean_get_operator()** (p. 12) returns -1.

3.4.1.2 **globus_list_t* globus_rsl_boolean_get_operand_list (globus_rsl_t * *ast_node*)**

Get the RSL operand list from a boolean RSL composition.

The **globus_rsl_boolean_get_operand_list()** (p. 13) function returns the list of RSL syntax tree nodes that is joined by a boolean composition.

Parameters:

ast_node The RSL syntax tree to inspect.

Returns:

Upon success, **globus_rsl_boolean_get_operand_list()** (p. 13) returns a pointer to a list of RSL syntax tree nodes that are the operand of a boolean composition operation. If an error occurs, **globus_rsl_boolean_get_operand_list()** (p. 13) returns NULL.

3.4.1.3 **globus_list_t** globus_rsl_boolean_get_operand_list_ref (globus_rsl_t * *boolean_node*)**

Get a reference to the RSL operand list from a boolean RSL composition.

The **globus_rsl_boolean_get_operand_list_ref()** (p. 13) function returns a pointer to the list of RSL syntax tree nodes that is joined by a boolean composition. If this list is modified, then the value of boolean syntax tree is modified.

Parameters:

boolean_node The RSL syntax tree to inspect.

Returns:

Upon success, **globus_rsl_boolean_get_operand_list_ref()** (p. 13) returns a pointer to the list pointer in the RSL syntax tree data structure. This list can be modified to change the operands of the boolean operation. If an error occurs, **globus_rsl_boolean_get_operand_list_ref()** (p. 13) returns NULL.

3.4.1.4 **char* globus_rsl_relation_get_attribute (globus_rsl_t * *ast_node*)**

Get an RSL relation attribute name.

The **globus_rsl_relation_get_attribute()** (p. 13) function returns a pointer to the name of the attribute in an RSL relation. This return value is a shallow reference to the attribute name.

Parameters:

ast_node The RSL relation node to inspect.

Returns:

Upon success, **globus_rsl_relation_get_attribute()** (p. 13) returns a pointer to the name of the attribute of the relation. If an error occurs, **globus_rsl_relation_get_attribute()** (p. 13) returns NULL.

3.4.1.5 **int globus_rsl_relation_get_operator (globus_rsl_t * *ast_node*)**

Get an RSL relation operator.

The **globus_rsl_relation_get_operator()** (p. 13) function returns the operation type represented by the RSL relation node pointed to by the *ast_node* parameter.

Parameters:

ast_node The RSL relation node to inspect.

Returns:

Upon success, **globus_rsl_relation_get_operator()** (p. 13) returns one of GLOBUS_RSL_EQ, GLOBUS_RSL_NEQ, GLOBUS_RSL_GT, GLOBUS_RSL_GTEQ, GLOBUS_RSL_LT, or GLOBUS_RSL_LTEQ. If an error occurs, **globus_rsl_relation_get_operator()** (p. 13) returns -1.

3.4.1.6 globus_rsl_value_t* globus_rsl_relation_get_value_sequence (globus_rsl_t * ast_node)

Get the value of an RSL relation.

The **globus_rsl_relation_get_value_sequence()** (p. 14) function returns the value of an RSL relation node pointed to by the *ast_node* parameter.

Parameters:

ast_node The RSL relation node to inspect.

Returns:

Upon success, **globus_rsl_relation_get_value_sequence()** (p. 14) returns the value sequence pointer in the RSL relation pointed to by the *ast_node* parameter. If an error occurs, **globus_rsl_relation_get_value_sequence()** (p. 14) returns NULL.

3.4.1.7 globus_rsl_value_t* globus_rsl_relation_get_single_value (globus_rsl_t * ast_node)

Get the single value of an RSL relation.

The **globus_rsl_relation_get_single_value()** (p. 14) function returns the value of an RSL relation node pointed to by the *ast_node* parameter if the value is a sequence of one value.

Parameters:

ast_node The RSL relation node to inspect.

Returns:

Upon success, **globus_rsl_relation_get_single_value()** (p. 14) returns the value pointer at the head of the RSL relation pointed to by the *ast_node* parameter. If the value sequence has more than one value or the *ast_node* points to an RSL syntax tree that is not a relation, **globus_rsl_relation_get_value_sequence()** (p. 14) returns NULL.

3.4.1.8 char* globus_rsl_value_literal_get_string (globus_rsl_value_t * literal_node)

Get the string value of an RSL literal.

The **globus_rsl_value_literal_get_string()** (p. 14) function returns the string value of an RSL literal node pointed to by the *literal_node* parameter.

Parameters:

literal_node The RSL literal node to inspect.

Returns:

Upon success, **globus_rsl_value_literal_get_string()** (p. 14) returns a pointer to the string value of the literal pointed to by the *literal_node* parameter. If the value is not a literal, **globus_rsl_value_literal_get_string()** (p. 14) returns NULL.

3.4.1.9 **globus_list_t* globus_rsl_value_sequence_get_value_list (globus_rsl_value_t * *sequence_node*)**

Get the value list from an RSL value sequence.

The **globus_rsl_value_sequence_get_value_list()** (p. 15) function returns the list of **globus_rsl_value_t** pointer values associated with the RSL value sequence pointed to by the *sequence_node* parameter.

Parameters:

sequence_node The RSL sequence node to inspect.

Returns:

Upon success, **globus_rsl_value_sequence_get_value_list()** (p. 15) returns a pointer to the list of values pointed to by the *sequence_node* parameter. If the value is not a sequence, **globus_rsl_value_literal_get_string()** (p. 14) returns NULL.

3.4.1.10 **globus_rsl_value_t* globus_rsl_value_variable_get_sequence (globus_rsl_value_t * *variable_node*)**

Get the value sequence from an RSL variable reference.

The **globus_rsl_value_variable_get_sequence()** (p. 15) function returns the sequence value associated with the RSL variable reference pointed to by the *variable_node* parameter.

Parameters:

variable_node The RSL variable node to inspect.

Returns:

Upon success, **globus_rsl_value_variable_get_sequence()** (p. 15) returns a pointer to the rsl value sequence pointed to by the *variable_node* parameter. If the value is not a variable reference, **globus_rsl_value_variable_get_sequence()** (p. 15) returns NULL.

3.4.1.11 **char* globus_rsl_value_variable_get_name (globus_rsl_value_t * *variable_node*)**

Get the name of an RSL variable reference.

The **globus_rsl_value_variable_get_name()** (p. 15) function returns a pointer to the name of the RSL variable name pointed to by the *variable_node* parameter.

Parameters:

variable_node The RSL variable node to inspect.

Returns:

Upon success, **globus_rsl_value_variable_get_name()** (p. 15) returns a pointer to the string containing the name of the variable referenced by the *variable_node* parameter. If the node is not a variable reference, **globus_rsl_value_variable_get_sequence()** (p. 15) returns NULL.

3.4.1.12 **char* globus_rsl_value_variable_get_default (globus_rsl_value_t * *variable_node*)**

Get the default value of an RSL variable reference.

The **globus_rsl_value_variable_get_default()** (p. 15) function returns a pointer to the default value of the RSL variable pointed to by the *variable_node* parameter to use if the variable's name is not bound in the current evaluation context.

Parameters:

variable_node The RSL variable node to inspect.

Returns:

Upon success, **globus_rsl_value_variable_get_default()** (p. 15) returns a pointer to the string containing the default value of the variable referenced by the *variable_node* parameter. If the node is not a variable reference or no default value exists in the RSL node, **globus_rsl_value_variable_get_default()** (p. 15) returns NULL.

3.4.1.13 int globus_rsl_value_variable_get_size (globus_rsl_value_t * *variable_node*)

Get the size of the value list within an RSL variable reference node.

The **globus_rsl_value_variable_get_size()** (p. 16) function returns the number of nodes in the RSL variable reference node pointed to by the *variable_node* parameter.

Parameters:

variable_node The RSL variable node to inspect.

Returns:

Upon success, **globus_rsl_value_variable_get_size()** (p. 16) returns the list of values within a RSL variable reference, or -1 if the node pointed to by *variable_node* is not a variable reference. If the return value is 1, then the variable has no default value included in the reference.

3.4.1.14 globus_rsl_value_t* globus_rsl_value_concatenation_get_left (globus_rsl_value_t * *concatenation_node*)

Get the left side of a concatenation value.

The **globus_rsl_value_concatenation_get_left()** (p. 16) function returns the left side of an RSL value concatenation pointed to by the *concatenation_node* parameter.

Parameters:

concatenation_node The RSL concatenation node to inspect.

Returns:

Upon success, **globus_rsl_value_concatenation_get_left()** (p. 16) returns a pointer to the left value of the concatenation values pointed to by the *concatenation_node* parameter. If an error occurs, **globus_rsl_value_concatenation_get_left()** (p. 16) returns NULL.

3.4.1.15 globus_rsl_value_t* globus_rsl_value_concatenation_get_right (globus_rsl_value_t * *concatenation_node*)

Get the right side of a concatenation value.

The **globus_rsl_value_concatenation_get_right()** (p. 16) function returns the right side of an RSL value concatenation pointed to by the *concatenation_node* parameter.

Parameters:

concatenation_node The RSL concatenation node to inspect.

Returns:

Upon success, **globus_rsl_value_concatenation_get_right()** (p. 16) returns a pointer to the right value of the concatenation values pointed to by the *concatenation_node* parameter. If an error occurs, **globus_rsl_value_concatenation_get_right()** (p. 16) returns NULL.

3.4.1.16 globus_list_t** globus_rsl_value_sequence_get_list_ref (globus_rsl_value_t * *sequence_node*)

Get a reference to the list of values in a sequence.

The **globus_rsl_value_sequence_get_list_ref()** (p. 17) function returns a reference to the list of values in a value sequence. Any changes to the elements of this list will affect the *sequence_node* parameter.

Parameters:

sequence_node The RSL sequence node to inspect.

Returns:

Upon success, **globus_rsl_value_sequence_get_list_ref()** (p. 17) returns a pointer to the list of the *globus_rsl_value_t* pointer values contained in the *sequence_node* parameter. If an error occurs, **globus_rsl_value_sequence_get_list_ref()** (p. 17) returns NULL.

3.5 List Functions

Functions

- **globus_list_t * globus_list_copy_reverse (globus_list_t * *orig*)**

3.5.1 Function Documentation

3.5.1.1 globus_list_t* globus_list_copy_reverse (globus_list_t * *orig*)

Create a reverse-order copy of a list.

The **globus_list_copy_reverse()** (p. 17) function creates and returns a copy of its input parameter, with the order of the list elements reversed. This copy is a shallow copy of list nodes, so both the list pointed to by *orig* and the returned list point to the same list element data.

Parameters:

orig A pointer to the list to copy.

Returns:

Upon success, **globus_list_copy_reverse()** (p. 17) returns a new list containing the same elements as the list pointed to by *orig* in reverse order. If an error occurs, **globus_list_copy_reverse()** (p. 17) returns NULL.

3.6 RSL Value Accessors

Functions

- **int globus_rsl_value_concatenation_set_left (globus_rsl_value_t *concatenation_node, globus_rsl_value_t *new_left_node)**
- **int globus_rsl_value_concatenation_set_right (globus_rsl_value_t *concatenation_node, globus_rsl_value_t *new_right_node)**

- **int globus_rsl_value_list_param_get** (globus_list_t *ast_node_list, int required_type, char ***value, int *value_ctr)
- **globus_list_t * globus_rsl_param_get_values** (globus_rsl_t *ast_node, char *param)
- **int globus_rsl_param_get** (globus_rsl_t *ast_node, int param_type, char *param, char ***values)

3.6.1 Function Documentation

3.6.1.1 **int globus_rsl_value_concatenation_set_left** (globus_rsl_value_t * *concatenation_node*, globus_rsl_value_t * *new_left_node*)

Set the left-hand value of a concatenation.

The **globus_rsl_value_concatenation_set_left**() (p. 18) sets the left hand side of a concatenation pointed to by *concatenation_node* to the value pointed to by *new_left_node*. If there was any previous value to the left hand side of the concatenation, it is discarded but not freed.

Parameters:

concatenation_node A pointer to the RSL value concatenation node to modify.

new_left_node A pointer to the new left hand side of the concatenation.

Returns:

Upon success, **globus_rsl_value_concatenation_set_left**() (p. 18) returns *GLOBUS_SUCCESS* and modifies the value pointed to by the *concatenation_node* parameter to use the value pointed to by the *new_left_node* parameter as its left hand side value. If an error occurs, **globus_rsl_value_concatenation_set_left**() (p. 18) returns -1.

3.6.1.2 **int globus_rsl_value_concatenation_set_right** (globus_rsl_value_t * *concatenation_node*, globus_rsl_value_t * *new_right_node*)

Set the right-hand value of a concatenation.

The **globus_rsl_value_concatenation_set_right**() (p. 18) sets the right-hand side of a concatenation pointed to by *concatenation_node* to the value pointed to by *new_right_node*. If there was any previous value to the right-hand side of the concatenation, it is discarded but not freed.

Parameters:

concatenation_node A pointer to the RSL value concatenation node to modify.

new_right_node A pointer to the new right hand side of the concatenation.

Returns:

Upon success, **globus_rsl_value_concatenation_set_right**() (p. 18) returns *GLOBUS_SUCCESS* and modifies the value pointed to by the *concatenation_node* parameter to use the value pointed to by the *new_right_node* parameter as its right hand side value. If an error occurs, **globus_rsl_value_concatenation_set_right**() (p. 18) returns -1.

3.6.1.3 **int globus_rsl_value_list_param_get** (globus_list_t * *ast_node_list*, int *required_type*, char ****value*, int * *value_ctr*)

Get the values of an RSL value list.

The **globus_rsl_value_list_param_get**() (p. 18) function copies pointers to literal string values or string pairs associated with the list of globus_rsl_value_t pointers pointed to by the *ast_node_list* parameter to the output array pointed to by the *value* parameter. It modifies the value pointed to by the *value_ctr* parameter to be the number of strings copied into the array.

Parameters:

- ast_node_list*** A pointer to a list of `globus_rsl_value_t` pointers whose values will be copied to the *value* parameter array.
- required_type*** A flag indicating whether the list is expected to contain literal strings or string pairs. This value may be one of `GLOBUS_RSL_VALUE_LITERAL` or `GLOBUS_RSL_VALUE_SEQUENCE`.
- value*** An output parameter pointing to an array of strings. This array must be at least as large as the number of elements in the list pointed to by *ast_node_list*.
- value_ctr*** An output parameter pointing to an integer that will be incremented for each string copied into the *value* array.

Returns:

Upon success, the `globus_rsl_value_list_param_get()` (p. 18) function returns `GLOBUS_SUCCESS` and modifies the values pointed to by the *value* and *value_ctr* parameters as described above. If an error occurs, `globus_rsl_value_list_param_get()` (p. 18) returns a non-zero value.

3.6.1.4 `globus_list_t* globus_rsl_param_get_values(globus_rsl_t * ast_node, char * param)`

Get the list of values for an RSL attribute.

The `globus_rsl_param_get_values()` (p. 19) function searches the RSL parse tree pointed to by the *ast_node* parameter and returns the value list that is bound to the attribute named by the *param* parameter.

Parameters:

- ast_node*** A pointer to an RSL syntax tree that will be searched. This may be a relation or boolean RSL string.
- param*** The name of the attribute to search for in the parse tree pointed to by the *ast_node* parameter.

Returns:

Upon success, the `globus_rsl_param_get_values()` (p. 19) function returns a pointer to the list of values associated with the attribute named by *param* in the RSL parse tree pointed to by *ast_node*. If an error occurs, `globus_rsl_param_get_values()` (p. 19) returns `NULL`.

3.6.1.5 `int globus_rsl_param_get(globus_rsl_t * ast_node, int param_type, char * param, char *** values)`

Get the value strings for an RSL attribute.

The `globus_rsl_param_get()` (p. 19) function searches the RSL parse tree pointed to by the *ast_node* parameter and returns an array of pointers to the strings bound to the attribute named by the *param* parameter.

Parameters:

- ast_node*** A pointer to an RSL syntax tree that will be searched. This may be a relation or boolean RSL string.
- param_type*** A flag indicating what type of values are expected for the RSL attribute named by the *param* parameter. This flag value may be `GLOBUS_RSL_PARAM_SINGLE_LITERAL`, `GLOBUS_RSL_PARAM_MULTI_LITERAL`, or `GLOBUS_RSL_PARAM_SEQUENCE`.
- param*** A string pointing to the name of the RSL attribute to search for.
- values*** An output parameter pointing to an array of strings that will be allocated and contain pointers to the RSL value strings if they match the format specified by the *param_type* flag. The caller is responsible for freeing this array, but not the strings in the array.

Returns:

Upon success, the `globus_rsl_param_get()` (p. 19) function returns `GLOBUS_SUCCESS` and modifies the *values* parameter as described above. If an error occurs, `globus_rsl_param_get()` (p. 19) returns a non-zero value.

3.7 RSL Display

Functions

- `int globus_rsl_value_print_recursive (globus_rsl_value_t *globus_rsl_value_ptr)`
- `char * globus_rsl_get_operator (int my_op)`
- `int globus_rsl_print_recursive (globus_rsl_t *ast_node)`
- `char * globus_rsl_unparse (globus_rsl_t *rsl_spec)`
- `char * globus_rsl_value_unparse (globus_rsl_value_t *rsl_value)`

3.7.1 Function Documentation

3.7.1.1 `int globus_rsl_value_print_recursive (globus_rsl_value_t * globus_rsl_value_ptr)`

Print the value of a `globus_rsl_value_t` to standard output.

The `globus_rsl_value_print_recursive()` (p. 20) function prints a string representation of the RSL value node pointed to by the *globus_rsl_value_ptr* parameter to standard output. This function is not reentrant.

Parameters:

globus_rsl_value_ptr A pointer to the RSL value to display.

Returns:

The `globus_rsl_value_print_recursive()` (p. 20) function always returns `GLOBUS_SUCCESS`.

3.7.1.2 `char* globus_rsl_get_operator (int my_op)`

Get the string representation of an RSL operator.

The `globus_rsl_get_operator()` (p. 20) function returns a pointer to a static string that represents the RSL operator passed in via the *my_op* parameter. If the operator is not value, then `globus_rsl_get_operator()` (p. 20) returns a pointer to the string "??"

Parameters:

my_op The RSL operator to return.

Returns:

The `globus_rsl_get_operator()` (p. 20) function returns a pointer to the string representation of the *my_op* parameter, or "??" if that value is not a value RSL operator.

3.7.1.3 `int globus_rsl_print_recursive (globus_rsl_t * ast_node)`

Print the value of an RSL syntax tree to standard output.

The `globus_rsl_print_recursive()` (p. 20) function prints a string representation of the RSL syntax tree pointed to by the *ast_node* parameter to standard output. This function is not reentrant.

Parameters:

ast_node A pointer to the RSL syntax tree to display.

Returns:

The `globus_rsl_print_recursive()` (p. 20) function always returns `GLOBUS_SUCCESS`.

3.7.1.4 char* globus_rsl_unparse (globus_rsl_t * rsl_spec)

Convert an RSL parse tree to a string.

The **globus_rsl_unparse()** (p. 21) function returns a new string which can be parsed into the RSL syntax tree passed as the *rsl_spec* parameter. The caller is responsible for freeing this string.

Parameters:

rsl_spec A pointer to the RSL syntax tree to unparse.

Returns:

Upon success, the **globus_rsl_unparse()** (p. 21) function returns a new string which represents the RSL parse tree passed as the *rsl_spec* parameter. If an error occurs, **globus_rsl_unparse()** (p. 21) returns NULL.

3.7.1.5 char* globus_rsl_value_unparse (globus_rsl_value_t * rsl_value)

Convert an RSL value pointer to a string.

The **globus_rsl_value_unparse()** (p. 21) function returns a new string which can be parsed into the value of an RSL relation that has the same syntactic meaning as the *rsl_value* parameter. The caller is responsible for freeing this string.

Parameters:

rsl_value A pointer to the RSL value node to unparse.

Returns:

Upon success, the **globus_rsl_value_unparse()** (p. 21) function returns a new string which represents the RSL value node passed as the *rsl_value* parameter. If an error occurs, **globus_rsl_value_unparse()** (p. 21) returns NULL.

3.8 RSL Helper Functions

Functions

- int globus_rsl_assist_attributes_canonicalize (globus_rsl_t *rsl)
- void globus_rsl_assist_string_canonicalize (char *ptr)

3.8.1 Detailed Description

The *rsl_assist* library provide a set of functions to canonicalize RSL parse trees or strings.

3.8.2 Function Documentation

3.8.2.1 int globus_rsl_assist_attributes_canonicalize (globus_rsl_t * rsl)

Canonicalize all attribute names in an RSL parse tree.

The **globus_rsl_assist_attributes_canonicalize()** (p. 21) function performs an in-place canonicalization of the RSL parse tree pointed to by its *rsl* parameter. All relation attribute names will be changed so that they lower-case, with all internal underscore characters removed.

Parameters:

rsl Pointer to the RSL parse tree to canonicalize.

Returns:

If **globus_rsl_assist_attributes_canonicalize()** (p. 21) is successful, it will ensure that all attribute names in the given RSL will be in canonical form and return **GLOBUS_SUCCESS**. If an error occurs, it will return **GLOBUS_FAILURE**.

Return values:

GLOBUS_SUCCESS Success

GLOBUS_FAILURE Failure

3.8.2.2 void globus_rsl_assist_string_canonicalize (char * *ptr*)

Canonicalize an attribute name.

The **globus_rsl_assist_string_canonicalize()** (p. 22) function modifies the NULL-terminated string pointed to by its *ptr* parameter so that it is in canonical form. The canonical form is all lower-case with all underscore characters removed.

Parameters:

ptr Pointer to the RSL string to modify in place.

Returns:

void

3.9 RSL Parsing**Functions**

- globus_rsl_t * **globus_rsl_parse** (char * *buf*)

3.9.1 Function Documentation**3.9.1.1 globus_rsl_t* globus_rsl_parse (char * *buf*)**

Parse an RSL string.

The **globus_rsl_parse()** (p. 22) function parses the string pointed to by the *buf* parameter into an RSL syntax tree. The caller is responsible for freeing that tree by calling **globus_rsl_free_recursive()** (p. 10).

Parameters:

buf A NULL-terminated string that contains an RSL relation or boolean composition.

Returns:

Upon success, the **globus_rsl_parse()** (p. 22) function returns the parse tree generated by processing its input. If an error occurs, **globus_rsl_parse()** (p. 22) returns NULL.

Index

- globus_list
 - globus_list_copy_reverse, 17
- globus_list_copy_reverse
 - globus_list, 17
- globus_rsl_accessor
 - globus_rsl_boolean_get_operand_list, 12
 - globus_rsl_boolean_get_operand_list_ref, 13
 - globus_rsl_boolean_get_operator, 12
 - globus_rsl_relation_get_attribute, 13
 - globus_rsl_relation_get_operator, 13
 - globus_rsl_relation_get_single_value, 14
 - globus_rsl_relation_get_value_sequence, 14
 - globus_rsl_value_concatenation_get_left, 16
 - globus_rsl_value_concatenation_get_right, 16
 - globus_rsl_value_literal_get_string, 14
 - globus_rsl_value_sequence_get_list_ref, 17
 - globus_rsl_value_sequence_get_value_list, 14
 - globus_rsl_value_variable_get_default, 15
 - globus_rsl_value_variable_get_name, 15
 - globus_rsl_value_variable_get_sequence, 15
 - globus_rsl_value_variable_get_size, 16
- globus_rsl_assist
 - globus_rsl_assist_attributes_canonicalize, 21
 - globus_rsl_assist_string_canonicalize, 22
- globus_rsl_assist_attributes_canonicalize
 - globus_rsl_assist, 21
- globus_rsl_assist_string_canonicalize
 - globus_rsl_assist, 22
- globus_rsl_boolean_get_operand_list
 - globus_rsl_accessor, 12
- globus_rsl_boolean_get_operand_list_ref
 - globus_rsl_accessor, 13
- globus_rsl_boolean_get_operator
 - globus_rsl_accessor, 12
- globus_rsl_constructors
 - globus_rsl_make_boolean, 6
 - globus_rsl_make_relation, 7
 - globus_rsl_value_make_concatenation, 8
 - globus_rsl_value_make_literal, 7
 - globus_rsl_value_make_sequence, 7
 - globus_rsl_value_make_variable, 8
- globus_rsl_copy_recursive
 - globus_rsl_memory, 9
- globus_rsl_eval
 - globus_rsl_memory, 11
- globus_rsl_free
 - globus_rsl_memory, 10
- globus_rsl_free_recursive
 - globus_rsl_memory, 10
- globus_rsl_get_operator
 - globus_rsl_print, 20
- globus_rsl_is_boolean
 - globus_rsl_predicates, 2
- globus_rsl_is_boolean_and
 - globus_rsl_predicates, 4
- globus_rsl_is_boolean_multi
 - globus_rsl_predicates, 4
- globus_rsl_is_boolean_or
 - globus_rsl_predicates, 4
- globus_rsl_is_relation
 - globus_rsl_predicates, 2
- globus_rsl_is_relation_attribute_equal
 - globus_rsl_predicates, 3
- globus_rsl_is_relation_eq
 - globus_rsl_predicates, 3
- globus_rsl_is_relation_lessthan
 - globus_rsl_predicates, 3
- globus_rsl_make_boolean
 - globus_rsl_constructors, 6
- globus_rsl_make_relation
 - globus_rsl_constructors, 7
- globus_rsl_memory
 - globus_rsl_copy_recursive, 9
 - globus_rsl_eval, 11
 - globus_rsl_free, 10
 - globus_rsl_free_recursive, 10
 - globus_rsl_value_copy_recursive, 9
 - globus_rsl_value_eval, 11
 - globus_rsl_value_free, 9
 - globus_rsl_value_free_recursive, 10
 - globus_rsl_value_list_literal_replace, 10
- globus_rsl_param
 - globus_rsl_param_get, 19
 - globus_rsl_param_get_values, 19
 - globus_rsl_value_concatenation_set_left, 18
 - globus_rsl_value_concatenation_set_right, 18
 - globus_rsl_value_list_param_get, 18
- globus_rsl_param_get
 - globus_rsl_param, 19
- globus_rsl_param_get_values
 - globus_rsl_param, 19
- globus_rsl_parse
 - globus_rsl_parse, 22
- globus_rsl_predicates
 - globus_rsl_is_boolean, 2
 - globus_rsl_is_boolean_and, 4
 - globus_rsl_is_boolean_multi, 4
 - globus_rsl_is_boolean_or, 4
 - globus_rsl_is_relation, 2
 - globus_rsl_is_relation_attribute_equal, 3
 - globus_rsl_is_relation_eq, 3
 - globus_rsl_is_relation_lessthan, 3
 - globus_rsl_value_is_concatenation, 6
 - globus_rsl_value_is_literal, 5
 - globus_rsl_value_is_sequence, 5
 - globus_rsl_value_is_variable, 5

- globus_rsl_print
 - globus_rsl_get_operator, 20
 - globus_rsl_print_recursive, 20
 - globus_rsl_unparse, 20
 - globus_rsl_value_print_recursive, 20
 - globus_rsl_value_unparse, 21
- globus_rsl_print_recursive
 - globus_rsl_print, 20
- globus_rsl_relation_get_attribute
 - globus_rsl_accessor, 13
- globus_rsl_relation_get_operator
 - globus_rsl_accessor, 13
- globus_rsl_relation_get_single_value
 - globus_rsl_accessor, 14
- globus_rsl_relation_get_value_sequence
 - globus_rsl_accessor, 14
- globus_rsl_unparse
 - globus_rsl_print, 20
- globus_rsl_value_concatenation_get_left
 - globus_rsl_accessor, 16
- globus_rsl_value_concatenation_get_right
 - globus_rsl_accessor, 16
- globus_rsl_value_concatenation_set_left
 - globus_rsl_param, 18
- globus_rsl_value_concatenation_set_right
 - globus_rsl_param, 18
- globus_rsl_value_copy_recursive
 - globus_rsl_memory, 9
- globus_rsl_value_eval
 - globus_rsl_memory, 11
- globus_rsl_value_free
 - globus_rsl_memory, 9
- globus_rsl_value_free_recursive
 - globus_rsl_memory, 10
- globus_rsl_value_is_concatenation
 - globus_rsl_predicates, 6
- globus_rsl_value_is_literal
 - globus_rsl_predicates, 5
- globus_rsl_value_is_sequence
 - globus_rsl_predicates, 5
- globus_rsl_value_is_variable
 - globus_rsl_predicates, 5
- globus_rsl_value_list_literal_replace
 - globus_rsl_memory, 10
- globus_rsl_value_list_param_get
 - globus_rsl_param, 18
- globus_rsl_value_literal_get_string
 - globus_rsl_accessor, 14
- globus_rsl_value_make_concatenation
 - globus_rsl_constructors, 8
- globus_rsl_value_make_literal
 - globus_rsl_constructors, 7
- globus_rsl_value_make_sequence
 - globus_rsl_constructors, 7
- globus_rsl_value_make_variable
 - globus_rsl_constructors, 8

- globus_rsl_value_print_recursive
 - globus_rsl_print, 20
- globus_rsl_value_sequence_get_list_ref
 - globus_rsl_accessor, 17
- globus_rsl_value_sequence_get_value_list
 - globus_rsl_accessor, 14
- globus_rsl_value_unparse
 - globus_rsl_print, 21
- globus_rsl_value_variable_get_default
 - globus_rsl_accessor, 15
- globus_rsl_value_variable_get_name
 - globus_rsl_accessor, 15
- globus_rsl_value_variable_get_sequence
 - globus_rsl_accessor, 15
- globus_rsl_value_variable_get_size
 - globus_rsl_accessor, 16

List Functions, 17

RSL Accessor Functions, 12

RSL Constructors, 6

RSL Display, 20

RSL Helper Functions, 21

RSL Memory Management, 9

RSL Parsing, 22

RSL Predicates, 1

RSL Value Accessors, 17