

globus rsl

9.1

Generated by Doxygen 1.7.6.1

Mon Mar 11 2013 14:14:14

## Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>Module Index</b>	<b>2</b>
2.1	Modules . . . . .	2
<b>3</b>	<b>Module Documentation</b>	<b>2</b>
3.1	RSL Predicates . . . . .	2
3.1.1	Detailed Description . . . . .	2
3.1.2	Function Documentation . . . . .	3
3.2	RSL Constructors . . . . .	8
3.2.1	Function Documentation . . . . .	8
3.3	RSL Memory Management . . . . .	11
3.3.1	Function Documentation . . . . .	11
3.4	RSL Accessor Functions . . . . .	15
3.4.1	Function Documentation . . . . .	15
3.5	List Functions . . . . .	21
3.5.1	Function Documentation . . . . .	21
3.6	RSL Value Accessors . . . . .	22
3.6.1	Function Documentation . . . . .	22
3.7	RSL Display . . . . .	25
3.7.1	Function Documentation . . . . .	25
3.8	RSL Helper Functions . . . . .	27
3.8.1	Detailed Description . . . . .	27
3.8.2	Function Documentation . . . . .	27
3.9	RSL Parsing . . . . .	28
3.9.1	Function Documentation . . . . .	28

## 1 Main Page

The Globus RSL library is provides the following functionality:

- **RSL Predicates** (p. 2)
- **RSL Constructors** (p. 8)
- **RSL Memory Management** (p. 11)
- **RSL Accessor Functions** (p. 15)
- **RSL Value Accessors** (p. 22)
- **RSL Display** (p. 25)

- **RSL Parsing** (p. 28)
- **List Functions** (p. 21)

## 2 Module Index

### 2.1 Modules

Here is a list of all modules:

<b>RSL Predicates</b>	<b>2</b>
<b>RSL Constructors</b>	<b>8</b>
<b>RSL Memory Management</b>	<b>11</b>
<b>RSL Accessor Functions</b>	<b>15</b>
<b>List Functions</b>	<b>21</b>
<b>RSL Value Accessors</b>	<b>22</b>
<b>RSL Display</b>	<b>25</b>
<b>RSL Helper Functions</b>	<b>27</b>
<b>RSL Parsing</b>	<b>28</b>

## 3 Module Documentation

### 3.1 RSL Predicates

#### 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. 3) 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

<i>ast</i>	Pointer to an RSL parse tree structure.
------------	---

#### Returns

The **`globus_rsl_is_relation()`** (p. 3) 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

<i>ast</i>	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. 4) 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

<i>ast</i>	Pointer to an RSL parse tree structure.
------------	---

## Returns

The **globus\_rsl\_is\_relation\_eq()** (p. 4) 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\_lessthan ( globus\_rsl\_t \* ast )

RSL less than operation test.

The **globus\_rsl\_is\_relation\_lessthan()** (p. 4) 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

<i>ast</i>	Pointer to an RSL parse tree structure.
------------	---

## Returns

The **globus\_rsl\_is\_relation\_lessthan()** (p. 4) 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

<i>ast</i>	Pointer to an RSL parse tree structure.
<i>attribute</i>	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. 5) 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

<i>ast</i>	Pointer to an RSL parse tree structure.
------------	---

## Returns

The **globus\_rsl\_is\_boolean\_and()** (p. 5) 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. 5) 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

<i>ast</i>	Pointer to an RSL parse tree structure.
------------	---

## Returns

The **globus\_rsl\_is\_boolean\_or()** (p. 5) 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

<i>ast</i>	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. 6) 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

<i>ast</i>	Pointer to an RSL value structure.
------------	------------------------------------

## Returns

The **globus\_rsl\_value\_is\_literal()** (p. 6) 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. 6) 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

<i>ast</i>	Pointer to an RSL value structure.
------------	------------------------------------

## Returns

The **globus\_rsl\_value\_is\_sequence()** (p. 6) 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

<i>ast</i>	Pointer to an RSL value structure.
------------	------------------------------------

## Returns

The **globus\_rsl\_value\_is\_sequence()** (p. 6) 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. 7) 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

<i>ast</i>	Pointer to an RSL value structure.
------------	------------------------------------

## Returns

The **globus\_rsl\_value\_is\_concatenation()** (p. 7) 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. 8) 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

<i>operator</i>	The boolean RSL operator to use to join the RSL parse tree list pointed to by the <i>children</i> parameter. This value must be one of GLOBUS_RSL_AND, GLOBUS_RSL_OR, GLOBUS_RSL_MULTIREQ in order to create a valid RSL tree.
<i>children</i>	Pointer to a list of RSL syntax trees to combine with the boolean operation described by the <i>operator</i> parameter.

#### Returns

The **`globus_rsl_make_boolean()`** (p. 8) 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. 8) 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. 8) 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

<i>operator</i>	The RSL operator to use to relate the RSL attribute name pointed to by the <i>attributename</i> parameter and the values pointed to by the <i>value_sequence</i> 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.
<i>attributename</i>	Pointer to a string naming the attribute of the new RSL relation.
<i>value_sequence</i>	Pointer to a sequence of RSL values to use in the new RSL relation.

## Returns

The **globus\_rsl\_make\_relation()** (p. 8) 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. 8) 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. 9) 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

<i>string</i>	The literal string to be used in the new value.
---------------	---

## Returns

The **globus\_rsl\_value\_make\_literal()** (p. 9) 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. 9) 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. 9) 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

<i>value_list</i>	A pointer to a list of globus_rsl_value_t pointers.
-------------------	---

## Returns

The **globus\_rsl\_value\_make\_sequence()** (p. 9) 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. 9) 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. 9) 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

<i>sequence</i>	A pointer to a RSL value sequence.
-----------------	------------------------------------

## Returns

The **globus\_rsl\_value\_make\_variable()** (p. 9) 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. 9) 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. 10) 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

<i>left_value</i>	A pointer to a RSL value to act as the left side of the concatenation. This must be a string literal or variable reference.
<i>right_value</i>	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. 10) 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. 10) 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. 11) 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

<i>ast_node</i>	An RSL syntax tree to copy.
-----------------	-----------------------------

#### Returns

The **`globus_rsl_copy_recursive()`** (p. 11) 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. 11) 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. 11) 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

<i>globus_rsl_value_ptr</i>	A pointer to an RSL value to copy.
-----------------------------	------------------------------------

## Returns

The **globus\_rsl\_value\_copy\_recursive()** (p. 11) 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. 11) 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. 12) 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

<i>val</i>	The RSL value node to free.
------------	-----------------------------

## Returns

The **globus\_rsl\_value\_free()** (p. 12) 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. 12) 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

<i>ast_node</i>	The RSL syntax tree node to free.
-----------------	-----------------------------------

## Returns

The **globus\_rsl\_value\_free()** (p. 12) 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. 12) 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. 12) returns.

## Parameters

<i>globus_rsl_value_ptr</i>	An RSL value node to free.
-----------------------------	----------------------------

## Returns

The **globus\_rsl\_value\_free\_recursive()** (p. 12) 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. 12) 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. 12) returns.

#### Parameters

<i>ast_node</i>	An RSL parse tree to free.
-----------------	----------------------------

#### Returns

The **globus\_rsl\_value\_free\_recursive()** (p. 12) function always returns GLOBUS\_SUCCESS.

#### 3.3.1.7 int globus\_rsl\_value\_list\_literal\_replace ( globus\_list\_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. 13) 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

<i>value_list</i>	The RSL value list to modify by replacing its first element.
<i>string_value</i>	The new string value to use as a literal first element of the list pointed to by the <i>value_list</i> parameter.

#### Returns

Upon success, **globus\_rsl\_value\_list\_literal\_replace()** (p. 13) 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. 13) 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. 13) 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. 12).

#### Parameters

<i>ast_node</i>	A pointer to the RSL value node to evaluate.
<i>symbol_table</i>	A symbol table containing current definitions of the RSL substitutions which can occur in this evaluation scope.
<i>string_value</i>	An output parameter which is set to point to the value of the string returned by evaluating the value node pointed to by <i>ast_node</i> if it evaluates to a literal value. list pointed to by the <i>value_list</i> parameter.
<i>rsl_substitution_flag</i>	A flag indicating whether the node pointed to by the <i>ast_node</i> parameter defines RSL substitution variables.

## Returns

Upon success, **globus\_rsl\_value\_eval()** (p. 13) 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. 13) 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. 14) 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. 12).

## Parameters

<i>ast_node</i>	A pointer to the RSL syntax tree to evaluate.
<i>symbol_table</i>	A symbol table containing current definitions of the RSL substitutions which can occur in this evaluation scope.

## Returns

Upon success, **globus\_rsl\_eval()** (p. 14) 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. 14) 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. 15) function returns the operator that is used by the boolean RSL composition.

#### Parameters

<i>ast_node</i>	The RSL syntax tree to inspect.
-----------------	---------------------------------

#### Returns

Upon success, **globus\_rsl\_boolean\_get\_operator()** (p. 15) returns one of GLOBUS\_RSL\_AND, GLOBUS\_RSL\_OR, GLOBUS\_RSL\_MULTIREQ. If an error occurs, **globus\_rsl\_boolean\_get\_operator()** (p. 15) 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. 15) function returns the list of RSL syntax tree nodes that is joined by a boolean composition.

#### Parameters

<i>ast_node</i>	The RSL syntax tree to inspect.
-----------------	---------------------------------



## Returns

Upon success, **globus\_rsl\_boolean\_get\_operand\_list()** (p. 15) 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. 15) returns NULL.

### 3.4.1.3 globus\_rsl\_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. 16) 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

<i>boolean_node</i>	The RSL syntax tree to inspect.
---------------------	---------------------------------

## Returns

Upon success, **globus\_rsl\_boolean\_get\_operand\_list\_ref()** (p. 16) 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. 16) 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. 16) 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

<i>ast_node</i>	The RSL relation node to inspect.
-----------------	-----------------------------------

## Returns

Upon success, **globus\_rsl\_relation\_get\_attribute()** (p. 16) returns a pointer to the name of the attribute of the relation. If an error occurs, **globus\_rsl\_relation\_get\_attribute()** (p. 16) 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. 16) function returns the operation type represented by the RSL relation node pointed to by the *ast\_node* parameter.

## Parameters

<i>ast_node</i>	The RSL relation node to inspect.
-----------------	-----------------------------------

## Returns

Upon success, **globus\_rsl\_relation\_get\_operator()** (p. 16) 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. 16) 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. 17) function returns the value of an RSL relation node pointed to by the *ast\_node* parameter.

##### Parameters

<i>ast_node</i>	The RSL relation node to inspect.
-----------------	-----------------------------------

##### Returns

Upon success, **globus\_rsl\_relation\_get\_value\_sequence()** (p. 17) 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. 17) 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. 17) 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

<i>ast_node</i>	The RSL relation node to inspect.
-----------------	-----------------------------------

##### Returns

Upon success, **globus\_rsl\_relation\_get\_single\_value()** (p. 17) 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. 17) 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. 17) function returns the string value of an RSL literal node pointed to by the *literal\_node* parameter.

##### Parameters

<i>literal_node</i>	The RSL literal node to inspect.
---------------------	----------------------------------

##### Returns

Upon success, **globus\_rsl\_value\_literal\_get\_string()** (p. 17) 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. 17) 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. 17) 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

<i>sequence_node</i>	The RSL sequence node to inspect.
----------------------	-----------------------------------

#### Returns

Upon success, **globus\_rsl\_value\_sequence\_get\_value\_list()** (p. 17) 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. 17) 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. 18) function returns the sequence value associated with the RSL variable reference pointed to by the *variable\_node* parameter.

#### Parameters

<i>variable_node</i>	The RSL variable node to inspect.
----------------------	-----------------------------------

#### Returns

Upon success, **globus\_rsl\_value\_variable\_get\_sequence()** (p. 18) 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. 18) 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. 18) function returns a pointer to the name of the RSL variable name pointed to by the *variable\_node* parameter.

#### Parameters

<i>variable_node</i>	The RSL variable node to inspect.
----------------------	-----------------------------------

#### Returns

Upon success, **globus\_rsl\_value\_variable\_get\_name()** (p. 18) 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\_name()** (p. 18) 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. 18) 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

<i>variable_node</i>	The RSL variable node to inspect.
----------------------	-----------------------------------

## Returns

Upon success, **globus\_rsl\_value\_variable\_get\_default()** (p. 18) 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. 18) 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. 19) function returns the number of nodes in the RSL variable reference node pointed to by the *variable\_node* parameter.

## Parameters

<i>variable_node</i>	The RSL variable node to inspect.
----------------------	-----------------------------------

## Returns

Upon success, **globus\_rsl\_value\_variable\_get\_size()** (p. 19) 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. 19) function returns the left side of an RSL value concatenation pointed to by the *concatenation\_node* parameter.

## Parameters

<i>concatenation_node</i>	The RSL concatenation node to inspect.
---------------------------	--

## Returns

Upon success, **globus\_rsl\_value\_concatenation\_get\_left()** (p. 19) 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. 19) 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. 19) function returns the right side of an RSL value concatenation pointed to by the *concatenation\_node* parameter.

## Parameters

<i>concatenation_node</i>	The RSL concatenation node to inspect.
---------------------------	--

## Returns

Upon success, **globus\_rsl\_value\_concatenation\_get\_right()** (p. 19) 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. 19) 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. 20) 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

<i>sequence_node</i>	The RSL sequence node to inspect.
----------------------	-----------------------------------

## Returns

Upon success, **globus\_rsl\_value\_sequence\_get\_list\_ref()** (p. 20) 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. 20) 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. 21) 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

<i>orig</i>	A pointer to the list to copy.
-------------	--------------------------------

#### Returns

Upon success, **`globus_list_copy_reverse()`** (p. 21) 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. 21) 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. 22) 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

<i>concatenation_node</i>	A pointer to the RSL value concatenation node to modify.
<i>new_left_node</i>	A pointer to the new left hand side of the concatenation.

#### Returns

Upon success, **globus\_rsl\_value\_concatenation\_set\_left()** (p. 22) 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. 22) 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. 22) 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

<i>concatenation_node</i>	A pointer to the RSL value concatenation node to modify.
<i>new_right_node</i>	A pointer to the new right hand side of the concatenation.

## Returns

Upon success, **globus\_rsl\_value\_concatenation\_set\_right()** (p. 22) 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. 22) 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. 23) 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

<i>ast_node_list</i>	A pointer to a list of <i>globus_rsl_value_t</i> pointers whose values will be copied to the <i>value</i> parameter array.
<i>required_type</i>	A flag indicating whether the list is expected to contain literal strings or string pairs. This value may be one of <i>GLOBUS_RSL_VALUE_LITERAL</i> or <i>GLOBUS_RSL_VALUE_SEQUENCE</i> .
<i>value</i>	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 <i>ast_node_list</i> .
<i>value_ctr</i>	An output parameter pointing to an integer that will be incremented for each string copied into the <i>value</i> array.

## Returns

Upon success, the **globus\_rsl\_value\_list\_param\_get()** (p. 23) 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. 23) 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. 23) 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

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

## Returns

Upon success, the **globus\_rsl\_param\_get\_values()** (p. 23) 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. 23) 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. 23) 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

<i>ast_node</i>	A pointer to an RSL syntax tree that will be searched. This may be a relation or boolean RSL string.
<i>param_type</i>	A flag indicating what type of values are expected for the RSL attribute named by the <i>param</i> parameter. This flag value may be <i>GLOBUS_RSL_PARAM_SINGLE_LITERAL</i> , <i>GLOBUS_RSL_PARAM_MULTI_LITERAL</i> , or <i>GLOBUS_RSL_PARAM_SEQUENCE</i> .
<i>param</i>	A string pointing to the name of the RSL attribute to search for.
<i>values</i>	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 <i>param_type</i> 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. 23) function returns *GLOBUS\_SUCCESS* and modifies the *values* parameter as described above. If an error occurs, **globus\_rsl\_param\_get()** (p. 23) 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. 25) 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

<i>globus_rsl_value_ptr</i>	A pointer to the RSL value to display.
-----------------------------	--

#### Returns

The **globus\_rsl\_value\_print\_recursive()** (p. 25) 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. 25) 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. 25) returns a pointer to the string "??"

#### Parameters

<i>my_op</i>	The RSL operator to return.
--------------	-----------------------------

#### Returns

The **globus\_rsl\_get\_operator()** (p. 25) 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. 25) 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

<i>ast_node</i>	A pointer to the RSL syntax tree to display.
-----------------	--

## Returns

The **globus\_rsl\_print\_recursive()** (p. 25) 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. 26) 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

<i>rsl_spec</i>	A pointer to the RSL syntax tree to unparse.
-----------------	--

## Returns

Upon success, the **globus\_rsl\_unparse()** (p. 26) 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. 26) 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. 26) 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

<i>rsl_value</i>	A pointer to the RSL value node to unparse.
------------------	---

## Returns

Upon success, the **globus\_rsl\_value\_unparse()** (p. 26) 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. 26) 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. 27) 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

<i>rsl</i>	Pointer to the RSL parse tree to canonicalize.
------------	--

#### Returns

If **globus\_rsl\_assist\_attributes\_canonicalize()** (p. 27) 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

<i>GLOBUS_SUCCESS</i>	Success
<i>GLOBUS_FAILURE</i>	Failure

##### 3.8.2.2 void globus\_rsl\_assist\_string\_canonicalize ( char \* ptr )

Canonicalize an attribute name.

The **globus\_rsl\_assist\_string\_canonicalize()** (p. 27) 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

<i>ptr</i>	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. 28) 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. 12).

#### Parameters

<i>buf</i>	A NULL-terminated string that contains an RSL relation or boolean composition.
------------	--

#### Returns

Upon success, the **globus\_rsl\_parse()** (p. 28) function returns the parse tree generated by processing its input. If an error occurs, **globus\_rsl\_parse()** (p. 28) returns NULL.