

globus rsl

7.2

Generated by Doxygen 1.7.1

Sun Jan 23 2011 13:15:48

# 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 . . . . .	2
3.2	RSL Constructors . . . . .	7
3.2.1	Function Documentation . . . . .	7
3.3	RSL Memory Management . . . . .	9
3.3.1	Function Documentation . . . . .	10
3.4	RSL Accessor Functions . . . . .	13
3.4.1	Function Documentation . . . . .	13
3.5	List Functions . . . . .	19
3.5.1	Function Documentation . . . . .	19
3.6	RSL Value Accessors . . . . .	19
3.6.1	Function Documentation . . . . .	19
3.7	RSL Display . . . . .	21
3.7.1	Function Documentation . . . . .	22
3.8	RSL Parsing . . . . .	23
3.8.1	Function Documentation . . . . .	23

## 1 Main Page

The Globus RSL library is provides the following functionality:

- [RSL Predicates](#)
- [RSL Constructors](#)
- [RSL Memory Management](#)
- [RSL Accessor Functions](#)
- [RSL Value Accessors](#)
- [RSL Display](#)
- [RSL Parsing](#)
- [List Functions](#)

## 2 Module Index

### 2.1 Modules

Here is a list of all modules:

<b>RSL Predicates</b>	<b><a href="#">2</a></b>
<b>RSL Constructors</b>	<b><a href="#">7</a></b>
<b>RSL Memory Management</b>	<b><a href="#">9</a></b>
<b>RSL Accessor Functions</b>	<b><a href="#">13</a></b>
<b>List Functions</b>	<b><a href="#">19</a></b>
<b>RSL Value Accessors</b>	<b><a href="#">19</a></b>
<b>RSL Display</b>	<b><a href="#">21</a></b>
<b>RSL Parsing</b>	<b><a href="#">23</a></b>

## 3 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) function tests whether 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\(\)\*](#) 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\(\)\*](#) function tests whether 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\\_less than\(\)\*](#) 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\(\)\*](#) function tests whether 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\\_variable\(\)\*](#) 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\(\)\*](#) function tests whether 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) returns.

## Parameters

*globus\_rsl\_value\_ptr* An RSL value node to free.

## Returns

The [\*globus\\_rsl\\_value\\_free\\_recursive\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) returns.

## Parameters

*ast\_node* An RSL parse tree to free.

## Returns

The [\*globus\\_rsl\\_value\\_free\\_recursive\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#).

#### 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#).

### 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) returns one of GLOBUS\_RSL\_AND, GLOBUS\_RSL\_OR, GLOBUS\_RSL\_MULTIREQ. If an error occurs, [\*globus\\_rsl\\_boolean\\_get\\_operator\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) returns a pointer to the name of the attribute of the relation. If an error occurs, [\*globus\\_rsl\\_relation\\_get\\_attribute\(\)\*](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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\(\)`](#) 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()` 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()` 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()` 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()` 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)\*](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) function returns `GLOBUS_SUCCESS` and modifies the *values* parameter as described above. If an error occurs, [globus\\_rsl\\_param\\_get\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) returns a pointer to the string "??"

##### Parameters

*my\_op* The RSL operator to return.

##### Returns

The [globus\\_rsl\\_get\\_operator\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) function returns a new string which represents the RSL parse tree passed as the *rsl\_spec* parameter. If an error occurs, [globus\\_rsl\\_unparse\(\)](#) 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\(\)](#) 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\(\)](#) 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\(\)](#) returns NULL.

## 3.8 RSL Parsing

### Functions

- globus\_rsl\_t \* [globus\\_rsl\\_parse](#) (char \*buf)

#### 3.8.1 Function Documentation

##### 3.8.1.1 globus\_rsl\_t\* globus\_rsl\_parse ( char \* buf )

Parse an RSL string.

The [globus\\_rsl\\_parse\(\)](#) 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\(\)](#).

##### Parameters

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



## Returns

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

# Index

- globus\_list
  - globus\_list\_copy\_reverse, 18
- globus\_list\_copy\_reverse
  - globus\_list, 18
- globus\_rsl\_accessor
  - globus\_rsl\_boolean\_get\_operand\_list, 13
  - globus\_rsl\_boolean\_get\_operand\_list\_ref, 13
  - globus\_rsl\_boolean\_get\_operator, 13
  - globus\_rsl\_relation\_get\_attribute, 14
  - globus\_rsl\_relation\_get\_operator, 14
  - globus\_rsl\_relation\_get\_single\_value, 15
  - globus\_rsl\_relation\_get\_value\_sequence, 14
  - globus\_rsl\_value\_concatenation\_get\_left, 17
  - globus\_rsl\_value\_concatenation\_get\_right, 17
  - globus\_rsl\_value\_literal\_get\_string, 15
  - globus\_rsl\_value\_sequence\_get\_list\_ref, 18
  - globus\_rsl\_value\_sequence\_get\_value\_list, 15
  - globus\_rsl\_value\_variable\_get\_default, 16
  - globus\_rsl\_value\_variable\_get\_name, 16
  - globus\_rsl\_value\_variable\_get\_sequence, 16
  - globus\_rsl\_value\_variable\_get\_size, 17
- globus\_rsl\_boolean\_get\_operand\_list
  - globus\_rsl\_accessor, 13
- globus\_rsl\_boolean\_get\_operand\_list\_ref
  - globus\_rsl\_accessor, 13
- globus\_rsl\_boolean\_get\_operator
  - globus\_rsl\_accessor, 13
- globus\_rsl\_constructors
  - globus\_rsl\_make\_boolean, 7
  - globus\_rsl\_make\_relation, 7
  - globus\_rsl\_value\_make\_concatenation, 8
  - globus\_rsl\_value\_make\_literal, 7
  - globus\_rsl\_value\_make\_sequence, 8
  - globus\_rsl\_value\_make\_variable, 8
- globus\_rsl\_copy\_recursive
  - globus\_rsl\_memory, 9
- globus\_rsl\_eval
  - globus\_rsl\_memory, 12
- globus\_rsl\_free
  - globus\_rsl\_memory, 10
- globus\_rsl\_free\_recursive
  - globus\_rsl\_memory, 11
- globus\_rsl\_get\_operator
  - globus\_rsl\_print, 22
- globus\_rsl\_is\_boolean
  - globus\_rsl\_predicates, 3
- globus\_rsl\_is\_boolean\_and
  - globus\_rsl\_predicates, 4
- globus\_rsl\_is\_boolean\_multi
  - globus\_rsl\_predicates, 5
- 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, 4
- 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, 7
- globus\_rsl\_make\_relation
  - globus\_rsl\_constructors, 7
- globus\_rsl\_memory
  - globus\_rsl\_copy\_recursive, 9
  - globus\_rsl\_eval, 12
  - globus\_rsl\_free, 10
  - globus\_rsl\_free\_recursive, 11
  - globus\_rsl\_value\_copy\_recursive, 9
  - globus\_rsl\_value\_eval, 11
  - globus\_rsl\_value\_free, 10
  - globus\_rsl\_value\_free\_recursive, 10
  - globus\_rsl\_value\_list\_literal\_replace, 11
- globus\_rsl\_param
  - globus\_rsl\_param\_get, 21
  - globus\_rsl\_param\_get\_values, 20
  - globus\_rsl\_value\_concatenation\_set\_left, 19
  - globus\_rsl\_value\_concatenation\_set\_right, 19
  - globus\_rsl\_value\_list\_param\_get, 20
- globus\_rsl\_param\_get
  - globus\_rsl\_param, 21
- globus\_rsl\_param\_get\_values
  - globus\_rsl\_param, 20
- globus\_rsl\_parse
  - globus\_rsl\_parse, 23
  - globus\_rsl\_parse, 23
- globus\_rsl\_predicates
  - globus\_rsl\_is\_boolean, 3
  - globus\_rsl\_is\_boolean\_and, 4
  - globus\_rsl\_is\_boolean\_multi, 5
  - globus\_rsl\_is\_boolean\_or, 4
  - globus\_rsl\_is\_relation, 2
  - globus\_rsl\_is\_relation\_attribute\_equal, 4
  - 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, 6
- globus\_rsl\_print
  - globus\_rsl\_get\_operator, 22
  - globus\_rsl\_print\_recursive, 22
  - globus\_rsl\_unparse, 22
  - globus\_rsl\_value\_print\_recursive, 21
  - globus\_rsl\_value\_unparse, 22

- globus\_rsl\_print\_recursive
  - globus\_rsl\_print, [22](#)
- globus\_rsl\_relation\_get\_attribute
  - globus\_rsl\_accessor, [14](#)
- globus\_rsl\_relation\_get\_operator
  - globus\_rsl\_accessor, [14](#)
- globus\_rsl\_relation\_get\_single\_value
  - globus\_rsl\_accessor, [15](#)
- globus\_rsl\_relation\_get\_value\_sequence
  - globus\_rsl\_accessor, [14](#)
- globus\_rsl\_unparse
  - globus\_rsl\_print, [22](#)
- globus\_rsl\_value\_concatenation\_get\_left
  - globus\_rsl\_accessor, [17](#)
- globus\_rsl\_value\_concatenation\_get\_right
  - globus\_rsl\_accessor, [17](#)
- globus\_rsl\_value\_concatenation\_set\_left
  - globus\_rsl\_param, [19](#)
- globus\_rsl\_value\_concatenation\_set\_right
  - globus\_rsl\_param, [19](#)
- globus\_rsl\_value\_copy\_recursive
  - globus\_rsl\_memory, [9](#)
- globus\_rsl\_value\_eval
  - globus\_rsl\_memory, [11](#)
- globus\_rsl\_value\_free
  - globus\_rsl\_memory, [10](#)
- 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, [6](#)
- globus\_rsl\_value\_list\_literal\_replace
  - globus\_rsl\_memory, [11](#)
- globus\_rsl\_value\_list\_param\_get
  - globus\_rsl\_param, [20](#)
- globus\_rsl\_value\_literal\_get\_string
  - globus\_rsl\_accessor, [15](#)
- 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, [8](#)
- globus\_rsl\_value\_make\_variable
  - globus\_rsl\_constructors, [8](#)
- globus\_rsl\_value\_print\_recursive
  - globus\_rsl\_print, [21](#)
- globus\_rsl\_value\_sequence\_get\_list\_ref
  - globus\_rsl\_accessor, [18](#)
- globus\_rsl\_value\_sequence\_get\_value\_list
  - globus\_rsl\_accessor, [15](#)

- globus\_rsl\_value\_unparse
  - globus\_rsl\_print, [22](#)
- globus\_rsl\_value\_variable\_get\_default
  - globus\_rsl\_accessor, [16](#)
- globus\_rsl\_value\_variable\_get\_name
  - globus\_rsl\_accessor, [16](#)
- globus\_rsl\_value\_variable\_get\_sequence
  - globus\_rsl\_accessor, [16](#)
- globus\_rsl\_value\_variable\_get\_size
  - globus\_rsl\_accessor, [17](#)

List Functions, [18](#)

RSL Accessor Functions, [13](#)

RSL Constructors, [6](#)

RSL Display, [21](#)

RSL Memory Management, [9](#)

RSL Parsing, [23](#)

RSL Predicates, [1](#)

RSL Value Accessors, [19](#)