RTI Routing Service

Version 7.7.0

Chapter 1

RTI Routing Service

RTI Routing Service∗ supports the integration and scaling of disparate and geographically dispersed systems. Using off-the-shelf or custom developed plugins, *Routing Service* supports the definition of data transformations and adapters that can interface to multiple protocols in addition to DDS, such as MQTT or legacy code written to the network socket API.

You can learn how to use *Routing Service* through the `RTI Routing Service User's Manual`.

A reference to the API can be found here:

- **RTI Routing Service Adapter API** (p. 46)
- **RTI Routing Service Processor API** (p. 9)
- **RTI Routing Library API** (p. 30)
- **RTI Routing Service Transformation API** (p. 39)

1.1 Feedback and Support for this Release

For more information, visit our knowledge base (`https://community.rti.com/kb`) to see sample code, general information on RTI Connext, performance information, troubleshooting tips, and technical details.

By its very nature, the knowledge base is continuously evolving and improving. We hope that you will find it helpful. If there are questions that you would like to see addressed or comments you would like to share, please send email to `support@rti.com`. We can only guarantee a response for customers with a current maintenance contract or subscription. To purchase a maintenance contract or subscription, contact your local RTI representative (see `http://www.rti.com/company/contact.html`), send an email request to `sales@rti.com`, or call +1 (408) 990-7400.

Please do not hesitate to contact RTI with questions or comments about this release. We welcome any input on how to improve RTI Routing Service∗ and *RTI Connext DDS* to suit your needs.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 5

Module Documentation

5.1 RTI Routing Service Processor API

Entity responsible for administering inputs and outputs.

Data Structures

- struct **RTI_RoutingServiceLoanedSamples**

  A pair of sample and sample info arrays. They are assumed to be of the same length.
- struct **RTI_RoutingServiceProcessor**

  Main Processor class.
- struct **RTI_RoutingServiceProcessorPlugin**

  ProcessorPlugin class.

Macros

- #define **RTI_RoutingServiceProcessorPlugin_initialize**(plugin)

  Utility macro to initialize a ProcessorPlugin object.

Typedefs

- typedef void(∗ **RTI_RoutingServiceProcessor_OnRouteEventFcn**) (void ∗processor_data, RTI_Routing↩ ServiceRouteEvent ∗route_event, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that processes an event that occurred in the Route where the Processor is installed.
- typedef void(∗ **RTI_RoutingServiceProcessor_UpdateFcn**) (void ∗processor_data, const struct **RTI_↩ RoutingServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that updates a Processor configuration.
- typedef struct **RTI_RoutingServiceProcessorPlugin** ∗(∗ **RTI_RoutingServiceProcessorPlugin_CreateFcn**) (const struct **RTI_RoutingServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that creates a ProcessorPlugin.
- typedef void(∗ **RTI_RoutingServiceProcessorPlugin_DeleteFcn**) (struct **RTI_RoutingServiceProcessor↩ Plugin** ∗plugin, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that deletes a ProcessorPlugin.
- typedef struct **RTI_RoutingServiceProcessor** ∗(∗ **RTI_RoutingServiceProcessorPlugin_Create↩ ProcessorFcn**) (void ∗processor_plugin_data, RTI_RoutingServiceRoute ∗route, const struct **RTI_Routing↩ ServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that creates a route Processor.

Enumerations

- enum **RTI_RoutingServiceRouteEventKind**

  Representation of the different event kinds triggered for a Route.

Functions

- const char * **RTI_RoutingServiceInput_get_name** (RTI_RoutingServiceInput *self)

  Returns the name of the specified input.
- const struct **RTI_RoutingServiceStreamInfo** * **RTI_RoutingServiceInput_get_stream_info** (RTI_Routing↩ ServiceInput *self)
- DDS_Boolean **RTI_RoutingServiceInput_is_active** (RTI_RoutingServiceInput *self)

  Checks whether an Input is active.
- DDS_Boolean **RTI_RoutingServiceInput_take** (RTI_RoutingServiceInput *self, struct **RTI_RoutingService**↩ **LoanedSamples** *loaned_samples)

  Takes all the available samples in this Input into the specified loaned samples instance.
- DDS_Boolean **RTI_RoutingServiceInput_take_w_selector** (RTI_RoutingServiceInput *self, struct **RTI_**↩ **RoutingServiceLoanedSamples** *loaned_samples, const struct RTI_RoutingServiceSelectorState *selector)

  Takes all the available samples in this Input into the specified loaned samples instance, filtering samples based on the given data selector.
- DDS_Boolean **RTI_RoutingServiceInput_read** (RTI_RoutingServiceInput *self, struct **RTI_RoutingService**↩ **LoanedSamples** *loaned_samples)

  Reads all the available samples in this Input into the specified loaned samples instance.
- DDS_Boolean **RTI_RoutingServiceInput_read_w_selector** (RTI_RoutingServiceInput *self, struct **RTI_**↩ **RoutingServiceLoanedSamples** *loaned_samples, const struct RTI_RoutingServiceSelectorState *selector)

  Reads all the available samples in this Input into the specified loaned samples instance, filtering samples based on the given data selector. Cannot be null.
- DDS_Boolean **RTI_RoutingServiceInput_return_loan** (RTI_RoutingServiceInput *self, struct **RTI_Routing**↩ **ServiceLoanedSamples** *loaned_samples)

  Returns a loan on the read or taken samples.
- RTI_RoutingServiceSelectorStateQueryData **RTI_RoutingServiceInput_create_content_query** (RTI_↩ RoutingServiceInput *self, RTI_RoutingServiceSelectorStateQueryData old_query_data, const struct RTI_↩ RoutingServiceSelectorContent *content)

  Creates a query object that selects the data with the specified filter.
- DDS_Boolean **RTI_RoutingServiceInput_delete_content_query** (RTI_RoutingServiceInput *self, RTI_↩ RoutingServiceSelectorStateQueryData query_data)

  Deletes a content query created from this Input.
- const char * **RTI_RoutingServiceOutput_get_name** (RTI_RoutingServiceOutput *self)

  Returns the name of the specified output.
- const struct **RTI_RoutingServiceStreamInfo** * **RTI_RoutingServiceOutput_get_stream_info** (RTI_Routing↩ ServiceOutput *self)

  Returns the type information associated with the specified Output. The type information refers to the format of the stream that this Output's write operation expects to receive from the processor of the route.
- DDS_Boolean **RTI_RoutingServiceOutput_write** (RTI_RoutingServiceOutput *self, const **RTI_Routing**↩ **ServiceSample** *sample_array, const **RTI_RoutingServiceSampleInfo** *sample_info_array, int *array_length)

  Writes a list of data and information samples.
- DDS_Boolean **RTI_RoutingServiceOutput_write_sample** (RTI_RoutingServiceOutput *self, const **RTI_**↩ **RoutingServiceSample** data, const **RTI_RoutingServiceSampleInfo** info)

  Writes a single sample, optionally with metatada indicated by an info sample.

- int **RTI_RoutingServiceRoute_get_input_count** (RTI_RoutingServiceRoute *self)

  Returns the number of inputs associated with this Route.
- DDS_Boolean **RTI_RoutingServiceRoute_is_input_enabled** (RTI_RoutingServiceRoute *self, int index)

  Checks whether the Input with the specified index is enabled.
- RTI_RoutingServiceInput * **RTI_RoutingServiceRoute_get_input_at** (RTI_RoutingServiceRoute *self, int index)

  Returns the Input at the specified index.
- RTI_RoutingServiceInput * **RTI_RoutingServiceRoute_lookup_input_by_name** (RTI_RoutingServiceRoute *self, const char *input_name)

  Looks up the specified Input by its configuration name.
- int **RTI_RoutingServiceRoute_get_output_count** (RTI_RoutingServiceRoute *self)

  Returns the number of outputs associated with this Route.
- DDS_Boolean **RTI_RoutingServiceRoute_is_output_enabled** (RTI_RoutingServiceRoute *self, int index)

  Checks whether the Output at the specified index is enabled.
- RTI_RoutingServiceOutput * **RTI_RoutingServiceRoute_get_output_at** (RTI_RoutingServiceRoute *self, int index)

  Returns the Output at the specified index.
- RTI_RoutingServiceOutput * **RTI_RoutingServiceRoute_lookup_output_by_name** (RTI_RoutingService↩Route *self, const char *output_name)

  Looks up the specified Output by its configuration name.
- void **RTI_RoutingServiceRoute_wakeup_route** (RTI_RoutingServiceRoute *route)

  Sends a wakeup event to the specified Route. This operation can be safely called from any thread context.
- void **RTI_RoutingServiceRoute_get_period** (RTI_RoutingServiceRoute *self, struct DDS_Duration_t *period)

  Get the current period of the periodic event for this Route.
- void **RTI_RoutingServiceRoute_set_period** (RTI_RoutingServiceRoute *self, const struct DDS_Duration_↩t *period)

  Changes the periodic event period for this route.
- RTI_RoutingServiceInput * **RTI_RoutingServiceRoute_get_first_input** (RTI_RoutingServiceRoute *self)

  Returns the first enabled Input in this route.
- RTI_RoutingServiceInput * **RTI_RoutingServiceRoute_get_next_input** (RTI_RoutingServiceRoute *self, RTI_RoutingServiceInput *prev_input)

  Returns the next enabled Input sibling to the specified input.
- RTI_RoutingServiceOutput * **RTI_RoutingServiceRoute_get_first_output** (RTI_RoutingServiceRoute *self)

  Returns the first enabled output in this route.
- RTI_RoutingServiceOutput * **RTI_RoutingServiceRoute_get_next_output** (RTI_RoutingServiceRoute *self, RTI_RoutingServiceOutput *prev_output)

  Returns the next enabled Output sibling to the specified output.
- const char * **RTI_RoutingServiceRoute_get_full_name** (RTI_RoutingServiceRoute *self)

  Returns the fully-qualified name of this Route, derived from the XML configuration.
- **RTI_RoutingServiceRouteEventKind** **RTI_RoutingServiceRouteEvent_get_kind** (RTI_RoutingService↩RouteEvent *self)

  Returns the kind of this RouteEvent.
- RTI_RoutingServiceRoute * **RTI_RoutingServiceRouteEvent_get_route** (RTI_RoutingServiceRouteEvent *self)

  Returns the Route that triggered the event.
- void * **RTI_RoutingServiceRouteEvent_get_event_data** (RTI_RoutingServiceRouteEvent *self)

  Returns the data associated with the event.
- void * **RTI_RoutingServiceRouteEvent_get_affected_entity** (RTI_RoutingServiceRouteEvent *self)

  Returns the entity that is directly affected by the event.

5.1.1 Detailed Description

Entity responsible for administering inputs and outputs.

5.1.2 Macro Definition Documentation

5.1.2.1 RTI_RoutingServiceProcessorPlugin_initialize

```
#define RTI_RoutingServiceProcessorPlugin_initialize(
              plugin )
```

Value:

```
{                                                                       \
    struct RTI_RoutingServiceVersion rsVersion = RTI_ROUTING_SERVICE_VERSION; \
    struct RTI_RoutingServiceVersion processorVersion = {0,0,0,0};      \
    (plugin)->_init = RTI_ROUTING_SERVICE_PROCESSOR_PLUGIN_INIT_NUMBER; \
    (plugin)->_rs_version = rsVersion;                                  \
    (plugin)->plugin_version = processorVersion;                        \
    (plugin)->plugin_delete = 0;                                        \
    (plugin)->create_processor = 0;                                     \
    (plugin)->delete_processor = 0;                                     \
    (plugin)->processor_plugin_data = 0;                                \
}
```

Utility macro to initialize a ProcessorPlugin object.

This macro must be called to initialize the value returned by RTI_RoutingServiceProcessorPlugin_CreateFcn.

Parameters

plugin	Pointer to the ProcessorPlugin object.

See also

RTI_RoutingServiceProcessorPlugin_CreateFcn (p. 13)

5.1.3 Typedef Documentation

5.1.3.1 RTI_RoutingServiceProcessor_OnRouteEventFcn

```
typedef void(* RTI_RoutingServiceProcessor_OnRouteEventFcn) (void *processor_data, RTI_Routing↩
ServiceRouteEvent *route_event,  RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that processes an event that occurred in the Route where the Processor is installed.

Parameters

processor_data	**In**. Implementation data of the Processor associated with the Route that triggered the event.
route_event	**In**. Event that was triggered within the Route.
env	**Out**. Environment for error indications.

5.1.3.2 RTI_RoutingServiceProcessor_UpdateFcn

```
typedef void(* RTI_RoutingServiceProcessor_UpdateFcn) (void *processor_data, const struct RTI_↩
RoutingServiceProperties *properties, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that updates a Processor configuration.

Parameters

processor_data	**In**. Implementation data.
properties	**In**. New configuration properties.
env	**Out**. Environment for error indications.

5.1.3.3 RTI_RoutingServiceProcessorPlugin_CreateFcn

```
typedef struct RTI_RoutingServiceProcessorPlugin *(* RTI_RoutingServiceProcessorPlugin_CreateFcn)
(const struct RTI_RoutingServiceProperties *properties, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that creates a ProcessorPlugin.

Parameters

properties	**In**. Configuration properties for the ProcessorPlugin.
env	**Out**. Environment for error indications.

Returns

New ProcessorPlugin instance if successful. Otherwise, `NULL`.

See also

RTI_RoutingServiceProcessorPlugin_DeleteFcn (p. 13)

5.1.3.4 RTI_RoutingServiceProcessorPlugin_DeleteFcn

```
typedef void(* RTI_RoutingServiceProcessorPlugin_DeleteFcn) (struct RTI_RoutingServiceProcessor↩
Plugin *plugin, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that deletes a ProcessorPlugin.

Parameters

plugin	**In**. ProcessorPlugin to be deleted.
env	**Out**. Environment for error indications.

See also

RTI_RoutingServiceProcessorPlugin_CreateFcn (p. 13)

5.1.3.5 RTI_RoutingServiceProcessorPlugin_CreateProcessorFcn

```
typedef struct RTI_RoutingServiceProcessor *(* RTI_RoutingServiceProcessorPlugin_CreateProcessor↩
Fcn) (void *processor_plugin_data, RTI_RoutingServiceRoute *route, const struct RTI_Routing↩
ServiceProperties *properties, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that creates a route Processor.

This function is called when a Processor is created.

Parameters

processor_plugin_data	implementation data of the plugin from which the Processor is created
route	Processor to be deleted.
properties	Configuration properties for the Processor. These properties correspond to the properties specified within the tag <processor>.
env	Environment for error indications.

Returns

New processor if successful. Otherwise, error.

See also

RTI_RoutingServiceProcessorPlugin::DeleteProcessorFcn

5.1.4 Enumeration Type Documentation

5.1.4.1 RTI_RoutingServiceRouteEventKind

```
enum RTI_RoutingServiceRouteEventKind
```

Representation of the different event kinds triggered for a Route.

5.1.5 Function Documentation

5.1.5.1 RTI_RoutingServiceInput_get_name()

```
const char * RTI_RoutingServiceInput_get_name (
            RTI_RoutingServiceInput * self )
```

Returns the name of the specified input.

The input name is given by the attribute of the corresponding XML tag in the configuration.

Parameters

self	**In**. Input for which the name is returned.

5.1.5.2 RTI_RoutingServiceInput_get_stream_info()

```
const struct RTI_RoutingServiceStreamInfo * RTI_RoutingServiceInput_get_stream_info (
            RTI_RoutingServiceInput * self )
```

Returns the type information associated with the specified Input. The TypeInformation refers to the format of the stream that the specified Input's read operation expects to receive.

Parameters

self	**In**. Input for which the type information is returned.

5.1.5.3 RTI_RoutingServiceInput_is_active()

```
DDS_Boolean RTI_RoutingServiceInput_is_active (
            RTI_RoutingServiceInput * self )
```

Checks whether an Input is active.

An Input is active when it has data available but not yet read. That is, the data available notification has been received but a call to read or take the samples has not been made.

Parameters

self	**In**. Input to be checked. Cannot be null.

Returns

DDS_BOOLEAN_FALSE if the Input is inactive, otherwise, DDS_BOOLEAN_TRUE.

5.1.5.4 RTI_RoutingServiceInput_take()

```
DDS_Boolean RTI_RoutingServiceInput_take (
            RTI_RoutingServiceInput * self,
            struct  RTI_RoutingServiceLoanedSamples * loaned_samples )
```

Takes all the available samples in this Input into the specified loaned samples instance.

Precondition

Input is enabled

This operation will call `take()` on the underlying RTI_RoutingServiceStreamReader, and the samples will be loaned into the given loaned samples collection. Samples loaned have to be returned by calling **RTI_RoutingServiceInput_↩ return_loan()** (p. 19).

Parameters

self	**In**. Input to take the samples from. Cannot be null.
loaned_samples	**Out**. Samples taken will be stored here. Cannot be null.

Returns

DDS_BOOLEAN_FALSE if the samples could not be taken, DDS_BOOLEAN_TRUE otherwise.

See also

RTI_RoutingServiceLoanedSamples (p. 85)

5.1.5.5 RTI_RoutingServiceInput_take_w_selector()

```
DDS_Boolean RTI_RoutingServiceInput_take_w_selector (
            RTI_RoutingServiceInput * self,
            struct  RTI_RoutingServiceLoanedSamples * loaned_samples,
            const struct RTI_RoutingServiceSelectorState * selector )
```

Takes all the available samples in this Input into the specified loaned samples instance, filtering samples based on the given data selector.

Precondition

Input is enabled

This operation will call `take()` on the underlying RTI_RoutingServiceStreamReader, and the samples will be loaned into the given loaned samples collection. Samples loaned have to be returned by calling **RTI_RoutingServiceInput_↩ return_loan()** (p. 19).

Parameters

self	**In**. Input to take the samples from.
loaned_samples	**Out**. Samples taken will be stored here.
selector	**In**. Selection criteria for the samples to be taken.

Returns

`DDS_BOOLEAN_FALSE` if the samples could not be taken, `DDS_BOOLEAN_TRUE` otherwise.

See also

RTI_RoutingServiceLoanedSamples (p. 85)

RTI_RoutingServiceSelectorState

5.1.5.6 RTI_RoutingServiceInput_read()

```
DDS_Boolean RTI_RoutingServiceInput_read (
            RTI_RoutingServiceInput * self,
            struct  RTI_RoutingServiceLoanedSamples * loaned_samples )
```

Reads all the available samples in this Input into the specified loaned samples instance.

Precondition

Input is enabled

This operation will call `read()` on the underlying RTI_RoutingServiceStreamReader, and the samples will be loaned into the given loaned samples collection. Samples loaned have to be returned by calling **RTI_RoutingServiceInput_↩ return_loan()** (p. 19).

Parameters

self	**In**. Input to read the samples from.
loaned_samples	**Out**. Samples read will be stored here.

Returns

DDS_BOOLEAN_FALSE) if the samples could not be read, DDS_BOOLEAN_TRUE otherwise.

See also

RTI_RoutingServiceLoanedSamples (p. 85)

5.1.5.7 RTI_RoutingServiceInput_read_w_selector()

```
DDS_Boolean RTI_RoutingServiceInput_read_w_selector (
            RTI_RoutingServiceInput * self,
            struct  RTI_RoutingServiceLoanedSamples * loaned_samples,
            const struct RTI_RoutingServiceSelectorState * selector )
```

Reads all the available samples in this Input into the specified loaned samples instance, filtering samples based on the given data selector. Cannot be null.

Precondition

Input is enabled

This operation will call read() on the underlying RTI_RoutingServiceStreamReader, and the samples will be loaned into the given loaned samples collection. Samples loaned have to be returned by calling **RTI_RoutingServiceInput_↩ return_loan()** (p. 19).

Parameters

self	**In**. Input to read the samples from. Cannot be null.
loaned_samples	**Out**. Samples taken will be stored here. Cannot be null.
selector	**In**. Selection criteria for the samples to be taken.

Returns

DDS_BOOLEAN_FALSE if the samples could not be read, DDS_BOOLEAN_TRUE otherwise.

See also

RTI_RoutingServiceLoanedSamples (p. 85)

RTI_RoutingServiceSelectorState

RTI_RoutingServiceInput_return_loan (p. 19)

5.1.5.8 RTI_RoutingServiceInput_return_loan()

```
DDS_Boolean RTI_RoutingServiceInput_return_loan (
            RTI_RoutingServiceInput * self,
            struct  RTI_RoutingServiceLoanedSamples * loaned_samples )
```

Returns a loan on the read or taken samples.

Call this method to indicate that you're done accessing the collection of samples obtained by an earlier invocation of **RTI_RoutingServiceInput_take()** (p. 16), **RTI_RoutingServiceInput_take_w_selector()** (p. 16), **RTI_Routing↩ ServiceInput_read()** (p. 17) or **RTI_RoutingServiceInput_read_w_selector()** (p. 18).

Parameters

self	**In**. Input the samples were taken/read from. Cannot be null.
loaned_samples	**Inout**. The loaned samples to be returned. Cannot be null.

Returns

DDS_BOOLEAN_FALSE if the samples could not be returned successfully, DDS_BOOLEAN_TRUE otherwise.

See also

RTI_RoutingServiceLoanedSamples (p. 85)

5.1.5.9 RTI_RoutingServiceInput_create_content_query()

```
RTI_RoutingServiceSelectorStateQueryData RTI_RoutingServiceInput_create_content_query (
            RTI_RoutingServiceInput * self,
            RTI_RoutingServiceSelectorStateQueryData old_query_data,
            const struct RTI_RoutingServiceSelectorContent * content )
```

Creates a query object that selects the data with the specified filter.

Precondition

Input is enabled.

This operation allows reading data with a SelectorState that contains a query object returned by this operation.

A query object type is implementation-dependent and guaranteed to to be used only within the same StreamReader that created it. Because a query object may be an expensive resource, this operation allows receiving a previously created query for a potential reuse and update of its filter.

Parameters

self	**In**. Input used to create the content query. Cannot be null.
old_query_data	**Inout**. A previously created query object that is provided for potential reuse and update of its filter.
content	**In**. The filter to be applied. Cannot be null.

Returns

The new or updated query object.

See also

RTI_RoutingServiceSelectorStateQueryData

RTI_RoutingServiceSelectorContent

5.1.5.10 RTI_RoutingServiceInput_delete_content_query()

```
DDS_Boolean RTI_RoutingServiceInput_delete_content_query (
             RTI_RoutingServiceInput * self,
             RTI_RoutingServiceSelectorStateQueryData query_data )
```

Deletes a content query created from this Input.

Precondition

Input is enabled

Parameters

self	**In**. Input used to create the content query. Cannot be null.
query_data	**In**. The query object to be deleted. Cannot be null.

Returns

`DDS_BOOLEAN_FALSE` if the operation fails, `DDS_BOOLEAN_TRUE` otherwise.

5.1.5.11 RTI_RoutingServiceOutput_get_name()

```
const char * RTI_RoutingServiceOutput_get_name (
             RTI_RoutingServiceOutput * self )
```

Returns the name of the specified output.

The output name is given by the attribute of the corresponding XML tag in the configuration.

Parameters

self	**In**. Output for which the name is required.

5.1.5.12 RTI_RoutingServiceOutput_get_stream_info()

```
const struct  RTI_RoutingServiceStreamInfo * RTI_RoutingServiceOutput_get_stream_info (
            RTI_RoutingServiceOutput * self )
```

Returns the type information associated with the specified Output. The type information refers to the format of the stream that this Output's write operation expects to receive from the processor of the route.

The TypeInformation may be the one coming from the Transformation if this Output has one. In this case, the type information is the one corresponding to the Output side of the Transformation.

Parameters

self	**In**. Output for which the type information is required. Cannot be null.

5.1.5.13 RTI_RoutingServiceOutput_write()

```
DDS_Boolean RTI_RoutingServiceOutput_write (
            RTI_RoutingServiceOutput * self,
            const  RTI_RoutingServiceSample * sample_array,
            const  RTI_RoutingServiceSampleInfo * sample_info_array,
            int * array_length )
```

Writes a list of data and information samples.

Precondition

Output enabled

Parameters

self	**In**. The Output where to write the samples. Cannot be null.
sample_array	**In**. Array of samples to write. Cannot be null.
sample_info_array	**In**. Array of information samples to write.
array_length	**Inout**. The length of the sample and info arrays. Output is the number of samples written. Cannot be null.

Returns

DDS_BOOLEAN_FALSE if the write operation was not successful. DDS_BOOLEAN_TRUE otherwise.

5.1.5.14 RTI_RoutingServiceOutput_write_sample()

```
DDS_Boolean RTI_RoutingServiceOutput_write_sample (
            RTI_RoutingServiceOutput * self,
            const  RTI_RoutingServiceSample data,
            const  RTI_RoutingServiceSampleInfo info )
```

Writes a single sample, optionally with metatada indicated by an info sample.

Precondition

Output enabled

Parameters

self	**In**. The Output where to write the samples. Cannot be null.
data	**In**. Data sample to be written. Cannot be null.
info	**In**. Associated information sample.

Returns

DDS_BOOLEAN_FALSE if the sample could not be written, DDS_BOOLEAN_TRUE otherwise.

5.1.5.15 RTI_RoutingServiceRoute_get_input_count()

```
int RTI_RoutingServiceRoute_get_input_count (
            RTI_RoutingServiceRoute * self )
```

Returns the number of inputs associated with this Route.

5.1.5.16 RTI_RoutingServiceRoute_is_input_enabled()

```
DDS_Boolean RTI_RoutingServiceRoute_is_input_enabled (
            RTI_RoutingServiceRoute * self,
            int index )
```

Checks whether the Input with the specified index is enabled.

Precondition

index >= 0 and index < RTI_RoutingServiceRoute_get_input_count(self)

Note: This function will return `DDS_BOOLEAN_FALSE` both when all the parameters are correct and the input is disabled, or when any of the parameters are incorrect. It's the responsibility of the caller to ensure the parameters are valid, so that the returned value is, too.

Parameters

self	**In**. The Route for which to check the Input status. Cannot be null.
index	**In**. The index, starting at 0, of the Input for which the status is required.

5.1.5.17 RTI_RoutingServiceRoute_get_input_at()

```
RTI_RoutingServiceInput * RTI_RoutingServiceRoute_get_input_at (
            RTI_RoutingServiceRoute * self,
            int index )
```

Returns the Input at the specified index.

Precondition

index >= 0 and index < RTI_RoutingServiceRoute_get_input_count(self)

Inputs are ordered according to the order of input declarations in the XML configuration. Then indexes are assigned increasingly starting at zero from the first declaration.

Returns

The Input at the specified index, or `NULL` if the Input is not enabled or the index is out of bounds.

5.1.5.18 RTI_RoutingServiceRoute_lookup_input_by_name()

```
RTI_RoutingServiceInput * RTI_RoutingServiceRoute_lookup_input_by_name (
            RTI_RoutingServiceRoute * self,
            const char * input_name )
```

Looks up the specified Input by its configuration name.

The Input configuration name is the value of the name attribute specified within the <input> tag in the XML configuration.

NOTE: The condition that the input must be enabled will be removed in a future version. This is being tracked with issue ID ROUTING-1131.

Parameters

self	**In**. The Route for which to obtain the Input. Cannot be null.
input_name	**In**. Name of the input configuration. Cannot be null.

Returns

The Input corresponding to the specified configuration name or NULL if it could not be found or if the input is not enabled.

5.1.5.19 RTI_RoutingServiceRoute_get_output_count()

```
int RTI_RoutingServiceRoute_get_output_count (
            RTI_RoutingServiceRoute * self )
```

Returns the number of outputs associated with this Route.

5.1.5.20 RTI_RoutingServiceRoute_is_output_enabled()

```
DDS_Boolean RTI_RoutingServiceRoute_is_output_enabled (
            RTI_RoutingServiceRoute * self,
            int index )
```

Checks whether the Output at the specified index is enabled.

Precondition

index >= 0 and index < RTI_RoutingServiceRoute_get_output_count(self)

Note: This function will return `DDS_BOOLEAN_FALSE` both when all the parameters are correct and the output is disabled, or when any of the parameters are incorrect. It's the responsibility of the caller to ensure the parameters are valid, so that the returned value is, too.

Parameters

self	**In**. The Route for which to check the Output status. Cannot be null.
index	**In**. The index, starting at 0, of the Output for which the status is required.

5.1.5.21 RTI_RoutingServiceRoute_get_output_at()

```
RTI_RoutingServiceOutput * RTI_RoutingServiceRoute_get_output_at (
            RTI_RoutingServiceRoute * self,
            int index )
```

Returns the Output at the specified index.

Precondition

index >= 0 and index < RTI_RoutingServiceRoute_get_output_count(self)

Outputs are ordered according to the order of output declarations in the XML configuration. Then indexes are assigned increasingly starting at zero from the first declaration.

Returns

The Output at the specified index, or `NULL` if the Output is not enabled or the index is out of bounds.

5.1.5.22 RTI_RoutingServiceRoute_lookup_output_by_name()

```
RTI_RoutingServiceOutput * RTI_RoutingServiceRoute_lookup_output_by_name (
            RTI_RoutingServiceRoute * self,
            const char * output_name )
```

Looks up the specified Output by its configuration name.

The Output configuration name is the value of the name attribute specified within the <output> tag in the XML configuration.

NOTE: The condition that the input must be enabled will be removed in a future version. This is being tracked with issue ID ROUTING-1131.

Parameters

self	**In**. The Route for which to obtain the Output. Cannot be null.
output_name	**In**. Name of the output configuration. Cannot be null.

Returns

The Output corresponding to the specified configuration name or NULL if it could not be found or if the output is not enabled.

5.1.5.23 RTI_RoutingServiceRoute_wakeup_route()

```
void RTI_RoutingServiceRoute_wakeup_route (
            RTI_RoutingServiceRoute * route )
```

Sends a wakeup event to the specified Route. This operation can be safely called from any thread context.

5.1.5.24 RTI_RoutingServiceRoute_get_period()

```
void RTI_RoutingServiceRoute_get_period (
            RTI_RoutingServiceRoute * self,
            struct DDS_Duration_t * period )
```

Get the current period of the periodic event for this Route.

Parameters

self	**In**. The Route for which to obtain the period. Cannot be null.
period	**Out**. The current periodic event period.

5.1.5.25 RTI_RoutingServiceRoute_set_period()

```
void RTI_RoutingServiceRoute_set_period (
            RTI_RoutingServiceRoute * self,
            const struct DDS_Duration_t * period )
```

Changes the periodic event period for this route.

Precondition

not DDS_Duration_is_zero(period)

not DDS_Duration_is_infinite(period)

This operation can be called many times within the same periodic event, in which only the last call will make effect.

The operation has no effect if the periodic event is not enabled in the route.

Parameters

self	**In**. The Route for which to set the period. Cannot be null.
period	**In**. New session period for periodic events.

5.1.5.26 RTI_RoutingServiceRoute_get_first_input()

```
RTI_RoutingServiceInput * RTI_RoutingServiceRoute_get_first_input (
            RTI_RoutingServiceRoute * self )
```

Returns the first enabled Input in this route.

This function, along with **RTI_RoutingServiceRoute_get_next_input()** (p. 28), provides iterator-based capabilities to iterate through a Route's enabled Inputs.

5.1.5.27 RTI_RoutingServiceRoute_get_next_input()

```
RTI_RoutingServiceInput * RTI_RoutingServiceRoute_get_next_input (
            RTI_RoutingServiceRoute * self,
            RTI_RoutingServiceInput * prev_input )
```

Returns the next enabled Input sibling to the specified input.

This function, along with **RTI_RoutingServiceRoute_get_first_input()** (p. 28), provides iterator-based capabilities to iterate through a Route's enabled Inputs.

5.1.5.28 RTI_RoutingServiceRoute_get_first_output()

```
RTI_RoutingServiceOutput * RTI_RoutingServiceRoute_get_first_output (
            RTI_RoutingServiceRoute * self )
```

Returns the first enabled output in this route.

This function, along with **RTI_RoutingServiceRoute_get_next_output()** (p. 28), provides iterator-based capabilities to iterate through a Route's enabled Outputs.

5.1.5.29 RTI_RoutingServiceRoute_get_next_output()

```
RTI_RoutingServiceOutput * RTI_RoutingServiceRoute_get_next_output (
            RTI_RoutingServiceRoute * self,
            RTI_RoutingServiceOutput * prev_output )
```

Returns the next enabled Output sibling to the specified output.

This function, along with **RTI_RoutingServiceRoute_get_first_output()** (p. 28), provides iterator-based capabilities to iterate through a Route's enabled Outputs.

5.1.5.30 RTI_RoutingServiceRoute_get_full_name()

```
const char * RTI_RoutingServiceRoute_get_full_name (
            RTI_RoutingServiceRoute * self )
```

Returns the fully-qualified name of this Route, derived from the XML configuration.

5.1.5.31 RTI_RoutingServiceRouteEvent_get_kind()

```
RTI_RoutingServiceRouteEventKind RTI_RoutingServiceRouteEvent_get_kind (
            RTI_RoutingServiceRouteEvent * self )
```

Returns the kind of this RouteEvent.

Returns

The Route that triggered the event, or `RTI_ROUTING_SERVICE_ROUTE_EVENT_NONE` if the parameter passed is `null`.

5.1.5.32 RTI_RoutingServiceRouteEvent_get_route()

```
RTI_RoutingServiceRoute * RTI_RoutingServiceRouteEvent_get_route (
            RTI_RoutingServiceRouteEvent * self )
```

Returns the Route that triggered the event.

Returns

The Route that triggered the event, or `NULL` if the parameter passed is null.

5.1.5.33 RTI_RoutingServiceRouteEvent_get_event_data()

```
void * RTI_RoutingServiceRouteEvent_get_event_data (
            RTI_RoutingServiceRouteEvent * self )
```

Returns the data associated with the event.

The event data depends on the kind of events. See documentation of individual events for the type of event data they provide.

Returns

The data associated with the event, or `NULL` if the parameter passed is null.

5.1.5.34 RTI_RoutingServiceRouteEvent_get_affected_entity()

```
void * RTI_RoutingServiceRouteEvent_get_affected_entity (
            RTI_RoutingServiceRouteEvent * self )
```

Returns the entity that is directly affected by the event.

Each kind of event can be affected by different types of entities. See documentation of individual events for the type of affected entity.

Returns

The entity affected by the event, or `NULL` if the parameter passed is null.

5.2 RTI Routing Library API

Prototype of the function that gets called upon reception of the shutdown command.

Data Structures

- struct **RTI_RoutingService**
  RTI Routing Service.
- struct **RTI_RoutingServiceTransportConfig**
  Association between a transport alias and its create function pointer.
- struct **RTI_RoutingServiceProperty**
  Configuration of RTI Routing Service.

Macros

- #define **RTI_RoutingServiceProperty_INITIALIZER**
  *The initial values for an **RTI_RoutingServiceProperty** (p. 91) instance.*

Functions

- struct **RTI_RoutingService** * **RTI_RoutingService_new** (const struct **RTI_RoutingServiceProperty** *property)
  Create a new RTI Routing Service instance.
- void **RTI_RoutingService_delete** (struct **RTI_RoutingService** *self)
  Stop and delete an RTI Routing Service instance.
- DDS_Boolean **RTI_RoutingService_start** (struct **RTI_RoutingService** *self)
  Start RTI Routing Service.
- DDS_Boolean **RTI_RoutingService_stop** (struct **RTI_RoutingService** *self)
  Stop RTI Routing Service.
- DDS_Boolean **RTI_RoutingService_attach_adapter_plugin** (struct **RTI_RoutingService** *self, void *adapter, const char *plugin_name)

Attach an adapter to be used by routing service when it is started.

- DDS_Boolean **RTI_RoutingService_attach_transformation_plugin** (struct **RTI_RoutingService** *self, struct **RTI_RoutingServiceTransformationPlugin** *transformation_plugin, const char *plugin_name)

  Attach a transformation plugin to be used by Routing Service when it is started.
- DDS_Boolean **RTI_RoutingService_attach_processor_plugin** (struct **RTI_RoutingService** *self, void *processor_plugin, const char *plugin_name)

  Attach a processor to be used by Routing Service when it is started.
- DDS_Boolean **RTI_RoutingService_set_remote_shutdown_hook** (struct **RTI_RoutingService** *self, const struct RTI_RoutingServiceRemoteShutdownHook *shutdown_hook)

  Set the remote shutdown hook in this Routing Service instance.
- void **RTI_RoutingService_execute_command** (struct **RTI_RoutingService** *self, struct RTI_Service_Admin↩ _CommandReply **reply, const struct RTI_Service_Admin_CommandRequest *request)

  Executes an Administration command on this service.
- void **RTI_RoutingService_return_reply** (struct **RTI_RoutingService** *self, struct RTI_Service_Admin_↩ CommandReply *reply)

  Returns the reply object that is obtained as result of executing a command.
- DDS_Boolean **RTI_RoutingService_initialize_globals** (void)
- DDS_Boolean **RTI_RoutingService_finalize_globals** (void)
- const char * **RTI_RoutingService_get_build_number_string** (void)

  Return the build ID of this library.
- DDS_Boolean **RTI_RoutingService_is_started** (struct **RTI_RoutingService** *self)

  Query whether this Routing Service is currently started.

Variables

- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_DEBUG**

  Verbosity level: exceptions + warnings + info + periodic + content.
- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_ALL**

  Verbosity level: exceptions + warnings + info + periodic.
- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_INFO**

  Verbosity level: exceptions + warnings + info.
- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_WARNINGS**

  Verbosity level: exceptions + warnings.
- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS**

  Verbosity level: exceptions.
- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_SILENT**

  Verbosity level: silent.

5.2.1 Detailed Description

Prototype of the function that gets called upon reception of the shutdown command.

Routing Service can be deployed as a C library linked into your application on select architectures.

Definition of the interface that handles the remote shutdown.

This API allows you to create, configure and start RTI Routing Service instances from your application.

The following code shows the typical use of the API:

```
struct RTI_RoutingServiceProperty property = RTI_RoutingServiceProperty_INITIALIZER;
struct RTI_RoutingService * service = NULL;
property.cfg_file = "my_routing_service_cfg.xml";
property.service_name = "my_routing_service";
...
service = RTI_RoutingService_new(&property);
if(service == NULL) {
   printf("Error...");
   return -1;
}
if(!RTI_RoutingService_start(service)) {
   printf("Error...");
   RTI_RoutingService_delete(service);
   return -1;
}
while(keep_running) {
    sleep();
    ...
}
RTI_RoutingService_delete(service);
return 0;
```

Instead of a file, you can use XML strings to configure RTI Routing Service. See **RTI_RoutingServiceProperty** (p. 91) for more information.

To build your application you need to link with the RTI Routing Service library in <**RTI Routing Service home**>/**bin**/<**architecture**>/

If you are using the C API on a Windows, Linux, MAC or INTEGRITY platform: See the example in <**RTI Routing Service home**>/**example**/**wrapperApp** Example makefiles and project files for several architecures are provided. Also see the README.txt file in the wrapperApp/src directory.

	Linux/macOS Systems	Windows Systems
Shared Libraries	librtirsinfrastructure.so	rtirsinfrastructure.dll
	librtidlc.so	rtidlc.dll
	librtiapputilsc.so	rtiapputilsc.dll
	libnddsmetp.so	nddsmetp.dll
	librticonnextmsgc.so	rticonnextmsgc.dll
	libnddsc.so	nddsc.dll
	librtixml2.so	rtixml2.dll
	libnddscore.so	nddscore.dll
Headers	**routingservice**/**routingservice_service.h** (p. 143)	

5.2.1.0.1 Development Requirements

5.2.2 Macro Definition Documentation

5.2.2.1 RTI_RoutingServiceProperty_INITIALIZER

```
#define RTI_RoutingServiceProperty_INITIALIZER
```

The initial values for an **RTI_RoutingServiceProperty** (p. 91) instance.

5.2.3 Function Documentation

5.2.3.1 RTI_RoutingService_new()

```
struct  RTI_RoutingService * RTI_RoutingService_new (
            const struct  RTI_RoutingServiceProperty * property )
```

Create a new RTI Routing Service instance.

Parameters

property	The properties to configure RTI Routing Service. This parameter is copied internally, so the user is responsible for releasing any memory allocated inside this structure.

MT Safety:

On non-Linux, non-Windows systems (i.e. VxWorks): UNSAFE for multiple threads to simultaneously make the FIRST call to **RTI_RoutingService_new()** (p. 33). Subsequent calls are thread safe. On Windows and Linux systems, these calls are thread-safe.

5.2.3.2 RTI_RoutingService_delete()

```
void RTI_RoutingService_delete (
            struct  RTI_RoutingService * self )
```

Stop and delete an RTI Routing Service instance.

See also

RTI_RoutingService_stop (p. 34)

Parameters

self	An **RTI_RoutingService** (p. 79) instance created with **RTI_RoutingService_new** (p. 33)

MT Safety:

This method is not thread-safe. Calling this method from different threads for the same Routing Service instance may result in undefined behavior.

5.2.3.3 RTI_RoutingService_start()

```
DDS_Boolean RTI_RoutingService_start (
            struct  RTI_RoutingService * self )
```

Start RTI Routing Service.

This is a non-blocking operation. RTI Routing Service will create its own set of threads to perform its tasks.

Parameters

self	An **RTI_RoutingService** (p. 79) instance created with **RTI_RoutingService_new** (p. 33)

5.2.3.4 RTI_RoutingService_stop()

```
DDS_Boolean RTI_RoutingService_stop (
            struct  RTI_RoutingService * self )
```

Stop RTI Routing Service.

This function won't return the execution control until the instance is fully stopped.

Parameters

self	An **RTI_RoutingService** (p. 79) instance created with **RTI_RoutingService_new** (p. 33)

MT Safety:

This method is not thread-safe. Calling this method from different threads for the same Routing Service instance may result in undefined behavior.

5.2.3.5 RTI_RoutingService_attach_adapter_plugin()

```
DDS_Boolean RTI_RoutingService_attach_adapter_plugin (
            struct  RTI_RoutingService * self,
            void * adapter,
            const char * plugin_name )
```

Attach an adapter to be used by routing service when it is started.

By using this function, an adapter can be statically compiled, created in your application and have routing service load it, instead of registering a shared library and a create function in the configuration. The name passed into this function is the name that has to be used in the configuration to instantiate connections from the plugin.

Example:

```
service = RTI_RoutingService_new(&property);
myAdapter = MyAdapter_create();
RTI_RoutingService_attach_adapter_plugin(service, myAdapter, "MyAdapter");
RTI_RoutingService_start(service);
...
```

And our configuration would look like this:

```
<dds>
 <!-- No need to register the plugin in
      <plugin_library><adapter_plugin>
 -->
 <routing_service name="example">
  <domain_route name="myadapter_to_dds">
    <connection name="MyConnection" plugin_name="MyAdapter">
       ...
    </connection>
    ...
  </domain_route>
 </routing_service>
</dds>
```

This function can be called as many times as desired to attach several plugins.

Note: The RTI Routing Service Adapter SDK is required.

Precondition

Routing Service must not be started.

Parameters

self	An **RTI_RoutingService** (p. 79) instance not started yet (or stopped)
adapter	The adapter plugin to be attached
plugin_name	The name used for this plugin in the <connection> tags in the configuration

5.2.3.6 RTI_RoutingService_attach_transformation_plugin()

```
DDS_Boolean RTI_RoutingService_attach_transformation_plugin (
             struct  RTI_RoutingService * self,
             struct  RTI_RoutingServiceTransformationPlugin * transformation_plugin,
             const char * plugin_name )
```

Attach a transformation plugin to be used by Routing Service when it is started.

See also

RTI_RoutingService_attach_adapter_plugin (p. 34)

5.2.3.7 RTI_RoutingService_attach_processor_plugin()

```
DDS_Boolean RTI_RoutingService_attach_processor_plugin (
            struct  RTI_RoutingService * self,
            void * processor_plugin,
            const char * plugin_name )
```

Attach a processor to be used by Routing Service when it is started.

See also

RTI_RoutingService_attach_adapter_plugin (p. 34)

5.2.3.8 RTI_RoutingService_set_remote_shutdown_hook()

```
DDS_Boolean RTI_RoutingService_set_remote_shutdown_hook (
            struct  RTI_RoutingService * self,
            const struct RTI_RoutingServiceRemoteShutdownHook * shutdown_hook )
```

Set the remote shutdown hook in this Routing Service instance.

The shutdown hook will be notified upon reception of a remote shutdown command.

The operation will fail if the RS is already started.

5.2.3.9 RTI_RoutingService_execute_command()

```
void RTI_RoutingService_execute_command (
            struct  RTI_RoutingService * self,
            struct RTI_Service_Admin_CommandReply ** reply,
            const struct RTI_Service_Admin_CommandRequest * request )
```

Executes an Administration command on this service.

This operation directly executes the specified `request` in this service as if it was directly received from an administration requester.

The request can represent any of the available administration operations as documented in the Administration chapter in the user's manual. This operation gives you an alternative way to control the service using the same remote administration interface but allowing you to call the operation directly.

Parameters

self	**In**. The Routing Service instance.
reply	**out**. Result of the processing of the command request.
request	**In**. Request to be processed.

5.2.3.10 RTI_RoutingService_return_reply()

```
void RTI_RoutingService_return_reply (
            struct  RTI_RoutingService * self,
            struct RTI_Service_Admin_CommandReply * reply )
```

Returns the reply object that is obtained as result of executing a command.

This operation allows the service to release the memory associated to the reply if needed.

Parameters

self	**In**. The Routing Service instance
reply	**In**. Reply result of the processing of the command request.

5.2.3.11 RTI_RoutingService_initialize_globals()

```
DDS_Boolean RTI_RoutingService_initialize_globals (
            void  )
```

[DEPRECATED]

This operation has been deprecated and will be removed in future releases. Routing Service does not require external initialization of the global state.

5.2.3.12 RTI_RoutingService_finalize_globals()

```
DDS_Boolean RTI_RoutingService_finalize_globals (
            void  )
```

[DEPRECATED]

This operation has been deprecated and will be removed in future releases. Routing Service does not require external finalization of the global state.

5.2.3.13 RTI_RoutingService_get_build_number_string()

```
const char * RTI_RoutingService_get_build_number_string (
            void  )
```

Return the build ID of this library.

The build ID uniquely identifies a specific build of the Routing Service library.

5.2.3.14 RTI_RoutingService_is_started()

```
DDS_Boolean RTI_RoutingService_is_started (
            struct  RTI_RoutingService * self )
```

Query whether this Routing Service is currently started.

5.2.4 Variable Documentation

5.2.4.1 RTI_ROUTING_SERVICE_LOG_VERBOSITY_DEBUG

```
const int RTI_ROUTING_SERVICE_LOG_VERBOSITY_DEBUG  [extern]
```

Verbosity level: exceptions + warnings + info + periodic + content.

5.2.4.2 RTI_ROUTING_SERVICE_LOG_VERBOSITY_ALL

```
const int RTI_ROUTING_SERVICE_LOG_VERBOSITY_ALL  [extern]
```

Verbosity level: exceptions + warnings + info + periodic.

5.2.4.3 RTI_ROUTING_SERVICE_LOG_VERBOSITY_INFO

```
const int RTI_ROUTING_SERVICE_LOG_VERBOSITY_INFO  [extern]
```

Verbosity level: exceptions + warnings + info.

5.2.4.4 RTI_ROUTING_SERVICE_LOG_VERBOSITY_WARNINGS

```
const int RTI_ROUTING_SERVICE_LOG_VERBOSITY_WARNINGS  [extern]
```

Verbosity level: exceptions + warnings.

5.2.4.5 RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS

```
const int RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS  [extern]
```

Verbosity level: exceptions.

5.2.4.6 RTI_ROUTING_SERVICE_LOG_VERBOSITY_SILENT

```
const int RTI_ROUTING_SERVICE_LOG_VERBOSITY_SILENT  [extern]
```

Verbosity level: silent.

5.3 RTI Routing Service Transformation API

This module describes the Transformation API.

Data Structures

- struct **RTI_RoutingServiceTransformationPlugin**
  Transformation plugin.

Macros

- #define **RTI_RoutingServiceTransformationPlugin_initialize**(transf)
  Initializes the plugin structure.

Typedefs

- typedef void * **RTI_RoutingServiceTransformation**
  Transformation.
- typedef struct **RTI_RoutingServiceTransformationPlugin** *(* **RTI_RoutingServiceTransformationPlugin_CreateFcn**) (const struct **RTI_RoutingServiceProperties** *properties, **RTI_RoutingServiceEnvironment** *env)
  Prototype of the function that creates a transformation plugin.
- typedef void(* **RTI_RoutingServiceTransformationPlugin_DeleteFcn**) (struct **RTI_RoutingServiceTransformationPlugin** *plugin, **RTI_RoutingServiceEnvironment** *env)
  Prototype of the function that deletes a transformation plugin.
- typedef **RTI_RoutingServiceTransformation**(* **RTI_RoutingServiceTransformationPlugin_CreateTransformationFcn**) (struct **RTI_RoutingServiceTransformationPlugin** *plugin, const struct **RTI_RoutingServiceTypeInfo** *input_type_info, const struct **RTI_RoutingServiceTypeInfo** *output_type_info, const struct **RTI_RoutingServiceProperties** *properties, **RTI_RoutingServiceEnvironment** *env)
  Prototype of the function that creates a route transformation.

- typedef void(* **RTI_RoutingServiceTransformationPlugin_DeleteTransformationFcn**) (struct **RTI_Routing↩ ServiceTransformationPlugin** *plugin, **RTI_RoutingServiceTransformation** transformation, **RTI_Routing↩ ServiceEnvironment** *env)

  Prototype of the function that deletes a transformation.
- typedef void(* **RTI_RoutingServiceTransformation_TransformFcn**) (**RTI_RoutingServiceTransformation** transformation, **RTI_RoutingServiceSample** **out_sample_lst, **RTI_RoutingServiceSampleInfo** **out_info↩ _lst, int *out_count, **RTI_RoutingServiceSample** *in_sample_lst, **RTI_RoutingServiceSampleInfo** *in_info↩ _lst, int in_count, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that transforms a sequence of input samples into a sequence of output samples.
- typedef void(* **RTI_RoutingServiceTransformation_ReturnLoanFcn**) (**RTI_RoutingServiceTransformation** transformation, **RTI_RoutingServiceSample** *sample_lst, **RTI_RoutingServiceSampleInfo** *info_lst, int count, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that returns the loan on the output samples and infos.
- typedef void(* **RTI_RoutingServiceTransformation_UpdateFcn**) (**RTI_RoutingServiceTransformation** transformation, const struct **RTI_RoutingServiceProperties** *properties, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that updates a transformation configuration.

5.3.1 Detailed Description

This module describes the Transformation API.

An RTI Routing Service route transforms the incoming data using a *transformation*, which is an object created by a transformation plugin.

Transformation plugins implement the transformation API described in this module and must be provided as shared libraries that RTI RTI Routing Service will load dynamically.

To register a transformation plugin with RTI Routing Service, you must use the tag <transformation_plugin> within <transformation_library>. For example:

```
<dds>
    ...
    <transformation_library name="MyTransfLib">
        <transformation_plugin name="MyTransfPlugin">
            <dll>mytransformation</dll>
            <create_function>
               MyTransfPlugin_create
           </create_function>
       </transformation_plugin>
       ...
    </transformation_library>
    ...
    <routing_service>
    ...
    </routing_service>
    ...
</dds>
```

Once a transformation plugin is registered, an Output can use it to create a data transformation. For example:

```
<topic_route name="SquareSwitchCoord">
    <input participant="1">
        <topic_name>Square</topic_name>
        <registered_type_name>ShapeType</registered_type_name>
    </input>
    <output>
        <topic_name>Square</topic_name>
        <registered_type_name>ShapeType</registered_type_name>
    </output>
    <transformation plugin_name="MyTransfLib::MyTransPlugin">
        <property>
            <value>
                <element>
                    <name>X</name>
                    <value>Y</value>
                </element>
                <element>
                    <name>Y</name>
                    <value>X</value>
                </element>
            </value>
        </property>
    </transformation>
</topic_route>
```

For additional information on configuring transformations, see the RTI Routing Service User's Manual.

5.3.2 Development Requirements

	Linux or macOS Systems	Windows Systems
Shared Library	librtirsinfrastructure.so If the transformation must work with adapters providing DDS_DynamicData or DDS↩ _SampleInfo (such as the built-in DDS adapter): libnddsc.so and libnddscore.so	rtirsinfrastructure.dll If the transformation must work with adapters providing DDS_DynamicData or DDS↩ _SampleInfo (such as the built-in DDS adapter): nddsc.dll and nddscore.dll
Header	**routingservice_transformation.h** (p. 154)If the transformation must work with adapters providing DDS_DynamicDa	

5.3.3 Macro Definition Documentation

5.3.3.1 RTI_RoutingServiceTransformationPlugin_initialize

```
#define RTI_RoutingServiceTransformationPlugin_initialize(
              transf )
```

Initializes the plugin structure.

This macro must be called to initialize the return value of RTI_RoutingServiceTransformationPlugin_CreateFcn

Parameters

transf	Pointer to the transformation plugin structure

See also

RTI_RoutingServiceTransformationPlugin_CreateFcn (p. 42)

5.3.4 Typedef Documentation

5.3.4.1 RTI_RoutingServiceTransformation

```
typedef void* RTI_RoutingServiceTransformation
```

Transformation.

A route can transform the incoming data using transformation objects.

The transformation objects are created by transformation plugins.

5.3.4.2 RTI_RoutingServiceTransformationPlugin_CreateFcn

```
typedef struct RTI_RoutingServiceTransformationPlugin *( * RTI_RoutingServiceTransformation↩
Plugin_CreateFcn) (const struct RTI_RoutingServiceProperties *properties, RTI_RoutingService↩
Environment *env)
```

Prototype of the function that creates a transformation plugin.

The name of the function that implements this prototype must be provided to RTI Routing Service using the tag <create_function> when the transformation plugin is registered.

For example:

```
<plugin_library name="MyTransfLib">
    <transformation_plugin name="MyTransfPlugin">
        <dll>mytransformation</dll>
        <create_function>
           MyTransfPlugin_create
        </create_function>
    </transformation_plugin>
    ...
</plugin_library>
```

This is the only function that is not part of **RTI_RoutingServiceTransformationPlugin** (p. 100).

Required: yes

Parameters

properties	<<*in*>> Configuration properties for the transformation.
env	<<*inout*>> Environment for error indications.

Returns

New plugin instance if successful. Otherwise, NULL.

See also

RTI_RoutingServiceTransformationPlugin_DeleteFcn (p. 43)

5.3.4.3 RTI_RoutingServiceTransformationPlugin_DeleteFcn

```
typedef void(* RTI_RoutingServiceTransformationPlugin_DeleteFcn) (struct RTI_RoutingService↩
TransformationPlugin *plugin, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that deletes a transformation plugin.

Transformation plugins are deleted when RTI Routing Service is closed.

Required: yes

Parameters

plugin	<<*in*>> Transformation plugin to be deleted.
env	<<*inout*>> Environment for error indications.

See also

RTI_RoutingServiceTransformationPlugin_CreateFcn (p. 42)

5.3.4.4 RTI_RoutingServiceTransformationPlugin_CreateTransformationFcn

```
typedef RTI_RoutingServiceTransformation( * RTI_RoutingServiceTransformationPlugin_CreateTransformation↩
Fcn) (struct RTI_RoutingServiceTransformationPlugin *plugin, const struct RTI_RoutingService↩
TypeInfo *input_type_info, const struct RTI_RoutingServiceTypeInfo *output_type_info, const struct
RTI_RoutingServiceProperties *properties, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that creates a route transformation.

This function is called when the route containing the transformation is ready to forward data.

The format associated with the input and output types depends on the format provided by the route adapters.

For the built-in DDS adapter, the format of the types is DDS_TypeCode.

Required: yes

Parameters

plugin	<<*in*>> Transformation plugin that will be used to create the transformation.
input_type_info	<<*in*>> Type information associated with the input samples.
output_type_info	<<*in*>> Type information associated with the output samples.
properties	<<*in*>> Configuration properties for the transformation. These properties corresponds to the properties specified within the tag <transformation>.
env	<<*inout*>> Environment for error indications.

Returns

New transformation if successful. Otherwise, error.

See also

RTI_RoutingServiceTransformationPlugin_DeleteTransformationFcn (p. 44)

5.3.4.5 RTI_RoutingServiceTransformationPlugin_DeleteTransformationFcn

```
typedef void(* RTI_RoutingServiceTransformationPlugin_DeleteTransformationFcn) (struct RTI_↩
RoutingServiceTransformationPlugin *plugin, RTI_RoutingServiceTransformation transformation,
RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that deletes a transformation.

This function is called when the route containing the transformation is disabled.

Parameters

plugin	<<*in*>> Transformation plugin that will be used to delete the transformation.
transformation	<<*in*>> Transformation to be deleted.
env	<<*inout*>> Environment for error indications.

See also

RTI_RoutingServiceTransformationPlugin_CreateTransformationFcn (p. 43)

5.3.4.6 RTI_RoutingServiceTransformation_TransformFcn

```
typedef void(* RTI_RoutingServiceTransformation_TransformFcn) ( RTI_RoutingServiceTransformation
transformation, RTI_RoutingServiceSample **out_sample_lst, RTI_RoutingServiceSampleInfo **out_↩
info_lst, int *out_count, RTI_RoutingServiceSample *in_sample_lst, RTI_RoutingServiceSampleInfo
*in_info_lst, int in_count, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that transforms a sequence of input samples into a sequence of output samples.

When RTI Routing Service is done using the output samples, it will 'return the loan' to the transformation by calling **RTI_RoutingServiceTransformation_ReturnLoanFcn** (p. 45).

The number of output samples can be different than the number of input samples.

The memory used by the output samples and sample infos is managed by the transformation. The transformation must allocate the memory for the output samples and sample infos and free it when RTI Routing Service calls **RTI_Routing↩ ServiceTransformation_ReturnLoanFcn** (p. 45).

The format associated with the input and output samples and sample infos depends on the format provided and consumed by the route StreamReader and StreamWriter.

For the built-in DDS adapter, the format of the samples is DDS_DynamicData and the format of the sample info is DDS_SampleInfo.

Required: yes

Parameters

transformation	<<*in*>> Transformation that will transform the samples.
out_sample_lst	<<*out*>> Array that will hold the output samples. This array will be provided by the transformation. *Note*: A transformation cannot return the input sample array as the output sample array; otherwise RTI Routing Service will fail. In general, the memory used for the output samples should be managed by the transformation.
out_info_lst	<<*out*>> Array that will hold the output sample infos. This array will be provided by the transformation. *Note*: A transformation cannot return the input sample info array as the output info array; otherwise RTI Routing Service will fail. In general, the memory used for the output sample infos should be managed by the transformation and freed when RTI Routing Service calls **RTI_RoutingServiceTransformation_ReturnLoanFcn** (p. 45).
out_count	<<*out*>> Number of output samples. The value must be greater than or equal to zero.
in_sample_lst	<<*in*>> Array of input samples.
in_info_lst	<<*in*>> Array of input sample infos.
in_count	<<*in*>> Number of input samples.
env	<<*inout*>> Environment for error indications.

See also

RTI_RoutingServiceTransformation_ReturnLoanFcn (p. 45)

5.3.4.7 RTI_RoutingServiceTransformation_ReturnLoanFcn

```
typedef void(* RTI_RoutingServiceTransformation_ReturnLoanFcn) ( RTI_RoutingServiceTransformation
transformation,  RTI_RoutingServiceSample *sample_lst,  RTI_RoutingServiceSampleInfo *info_lst,
int count,  RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that returns the loan on the output samples and infos.

This function is called by RTI Routing Service to indicate that it is done accessing the array of data samples obtained by an earlier invocation of **RTI_RoutingServiceTransformation_TransformFcn** (p. 44).

The memory used by the output samples and sample infos is managed by the transformation and should be freed in this call if it was allocated when RTI Routing Service calls **RTI_RoutingServiceTransformation_ReturnLoanFcn** (p. 45).

Required: yes

Parameters

transformation	<<*in*>> Transformation that owns the samples and sample infos.
sample_lst	<<*in*>> Array of samples.
info_lst	<<*in*>> Array of sample infos.
count	<<*in*>> Number of samples in the array.
env	<<*inout*>> Environment for error indications.

5.3.4.8 RTI_RoutingServiceTransformation_UpdateFcn

```
typedef void(* RTI_RoutingServiceTransformation_UpdateFcn) ( RTI_RoutingServiceTransformation
transformation, const struct  RTI_RoutingServiceProperties *properties,  RTI_RoutingService↩
Environment *env)
```

Prototype of the function that updates a transformation configuration.

Parameters

transformation	<<*in*>> Transformation.
properties	<<*in*>> New configuration properties.
env	<<*inout*>> Environment for error indications.

5.4 RTI Routing Service Adapter API

This module describes the C Adapter API.

Data Structures

- struct **RTI_RoutingServiceStreamReaderListener**

  StreamReader listener used to notify Routing Service that new data is available.
- struct **RTI_RoutingServiceAdapterPlugin**

  Adapter plugin.

Macros

- #define **RTI_RoutingServiceAdapterPlugin_initialize**(adapter)

  Initializes the adapter plugin structure.

Typedefs

- typedef void * **RTI_RoutingServiceStreamWriter**

  StreamWriter.
- typedef int(* **RTI_RoutingServiceStreamWriter_WriteFcn**) (**RTI_RoutingServiceStreamWriter** stream_writer, const **RTI_RoutingServiceSample** *sample_list, const **RTI_RoutingServiceSampleInfo** *info_list, int count, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that writes a collection of data samples to an output stream.
- typedef void * **RTI_RoutingServiceStreamReader**

  StreamReader.
- typedef void(* **RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback**) (**RTI_RoutingServiceStreamReader** stream_reader, void *listener_data)

  Prototype of the callback used to notify of new samples.
- typedef void(* **RTI_RoutingServiceStreamReader_ReadFcn**) (**RTI_RoutingServiceStreamReader** stream_reader, **RTI_RoutingServiceSample** **sample_list, **RTI_RoutingServiceSampleInfo** **info_list, int *count, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that reads a collection of data samples and sample infos from an input stream.
- typedef void(* **RTI_RoutingServiceStreamReader_ReturnLoanFcn**) (**RTI_RoutingServiceStreamReader** stream_reader, **RTI_RoutingServiceSample** *sample_list, **RTI_RoutingServiceSampleInfo** *info_list, int count, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that returns the loan on the read samples and infos.
- typedef void * **RTI_RoutingServiceSession**

  Session.
- typedef void * **RTI_RoutingServiceConnection**

  Connection.
- typedef **RTI_RoutingServiceSession**(* **RTI_RoutingServiceConnection_CreateSessionFcn**) (**RTI_RoutingServiceConnection** connection, const struct **RTI_RoutingServiceProperties** *properties, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that creates a Session.
- typedef void(* **RTI_RoutingServiceConnection_DeleteSessionFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceSession** session, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that deletes a Session.
- typedef **RTI_RoutingServiceStreamReader**(* **RTI_RoutingServiceConnection_CreateStreamReaderFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceSession** session, const struct **RTI_RoutingServiceStreamInfo** *stream_info, const struct **RTI_RoutingServiceProperties** *properties, const struct **RTI_RoutingServiceStreamReaderListener** *listener, **RTI_RoutingServiceEnvironment** *env)

Prototype of the function that creates a StreamReader.

- typedef void(∗ **RTI_RoutingServiceConnection_DeleteStreamReaderFcn**) (**RTI_RoutingService↩ Connection** connection, **RTI_RoutingServiceStreamReader** stream_reader, **RTI_RoutingService↩ Environment** ∗env)

  Prototype of the function that deletes a StreamReader.

- typedef **RTI_RoutingServiceStreamWriter**(∗ **RTI_RoutingServiceConnection_CreateStreamWriterFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceSession** session, const struct **RTI_↩ RoutingServiceStreamInfo** ∗stream_info, const struct **RTI_RoutingServiceProperties** ∗properties, **RTI_↩ RoutingServiceEnvironment** ∗env)

  Prototype of the function that creates a StreamWriter.

- typedef void(∗ **RTI_RoutingServiceConnection_DeleteStreamWriterFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceStreamWriter** stream_writer, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that deletes a StreamWriter.

- typedef **RTI_RoutingServiceStreamReader**(∗ **RTI_RoutingServiceConnection_GetDiscoveryReaderFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that gets a built-in discovery StreamReader.

- typedef **RTI_RoutingServiceTypeRepresentation**(∗ **RTI_RoutingServiceConnection_CopyType↩ RepresentationFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceTypeRepresentation↩ Kind** type_representation_kind, **RTI_RoutingServiceTypeRepresentation** type_representation, **RTI_↩ RoutingServiceEnvironment** ∗env)

  Prototype of the function that copies a type representation.

- typedef void(∗ **RTI_RoutingServiceConnection_DeleteTypeRepresentationFcn**) (**RTI_RoutingService↩ Connection** connection, **RTI_RoutingServiceTypeRepresentationKind** type_representation_kind, **RTI_↩ RoutingServiceTypeRepresentation** type_representation, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that deletes a type representation.

- typedef void ∗ **RTI_RoutingServiceAdapterEntity**

  Adapter entity.

- typedef void(∗ **RTI_RoutingServiceAdapterEntity_UpdateFcn**) (**RTI_RoutingServiceAdapterEntity** entity, const struct **RTI_RoutingServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that updates the configuration of an adapter entity.

- typedef void(∗ **RTI_RoutingServiceAdapterPlugin_DeleteFcn**) (struct **RTI_RoutingServiceAdapterPlugin** ∗plugin, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that deletes an adapter plugin.

- typedef struct **RTI_RoutingServiceAdapterPlugin** ∗(∗ **RTI_RoutingServiceAdapterPlugin_CreateFcn**) (const struct **RTI_RoutingServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that creates an adapter plugin.

Variables

- **RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback RTI_RoutingServiceStream↩ ReaderListener::on_data_available**

  Prototype of the callback used to notify of new samples.

5.4.1 Detailed Description

This module describes the C Adapter API.

Adapters are pluggable components that allow *RTI Routing Service* to consume and produce data for different data domains (e.g., Connext DDS, MQTT, Socket, etc.).

By default, *Routing Service* is distributed with a builtin DDS adapter. Any other adapter plugins must be provided as external components registered through the XML configuration or through the Library API.

The following figure shows an overview of the *Routing Service* adapter.

Input adapters are used to collect data samples from different data domains, such as DDS or MQTT. The input samples are processed by the Routing Service engine and are passed along to output adapters through the Processor, applying any Transformation beforehand if present.

For additional details about Adapter configuration see the `RTI Routing Service User's Manual`.

5.4.2 Development Requirements

	Linux/macOS Systems	Windows Systems
Shared Libraries	librtirsinfrastructure.so	rtirsinfrastructure.dll
	libnddsc.so	nddsc.dll
	libnddscore.so	nddscore.dll
Headers	**routingservice_adapter.h** (p. 118)	

5.4.3 Architecture

The Adapter architecture is shown in the class diagram below.

The sequence diagram in this figure shows when the different adapter entities are created.

- An **RTI_RoutingServiceAdapterPlugin** (p. 79) object is created as soon as RTI Routing Service starts and finds out about the plug-ins that will be used by underlying DomainRoute connections.
- A **RTI_RoutingServiceConnection** (p. 54) object is created when the DomainRoute (<domain_route>) that contain it is enabled.
- A **RTI_RoutingServiceSession** (p. 54) object is created when the associated service Session (<session>) is enabled.
- An inputs **RTI_RoutingServiceStreamReader** (p. 52) is created if the Route is enabled and the creation mode condition associated with the <input> tag becomes true.
- An outputs **RTI_RoutingServiceStreamWriter** (p. 51) is created if the route is enabled and the creation mode condition associated with the <output> tag becomes true.

5.4.3.1 Stream Discovery

A Route cannot forward data until the type representations (e.g., TypeCode) associated with the input and output streams are available.

If a Route refers to types that are not defined in the configuration file, RTI Routing Service has to discover their type representation (e.g., TypeCode) before creating the **RTI_RoutingServiceStreamReader** (p. 52) and **RTI_Routing↩ ServiceStreamWriter** (p. 51). The adapter discovery API is used to provide stream and type information in a data domain to Routing Service∗.

The discovery API consists of the following methods:

- **RTI_RoutingServiceAdapterPlugin::connection_get_input_stream_discovery_reader** (p. 83)
- **RTI_RoutingServiceAdapterPlugin::connection_get_output_stream_discovery_reader** (p. 83)

These methods provide access to **RTI_RoutingServiceStreamReader** (p. 52) used to discover streams in the data domain associated with a connection.

The input stream discovery **RTI_RoutingServiceStreamReader** (p. 52) provides information about input streams. An input stream is a stream from which an input's **RTI_RoutingServiceStreamReader** (p. 52) reads data. Notification of disposed scenarios, where an input stream disappears, are also made using the input stream discovery StreamReader.

In the builtin DDS adapter, the input stream discovery StreamReader is associated with the publication builtin Data↩ Reader of the DomainParticipant.

The output stream discovery **RTI_RoutingServiceStreamReader** (p. 52) provides information about output streams. An output stream is a stream to which an output's **RTI_RoutingServiceStreamWriter** (p. 51) can write data. Notification of disposed scenarios, where an output stream disappears, are also made using the output stream discovery Stream↩ Reader.

In the builtin DDS adapter, the output stream discovery StreamReader is associated with the subscription builtin Data↩ Reader of the DomainParticipant.

The samples provided by the stream discovery StreamReaders have the type **RTI_RoutingServiceStreamInfo** (p. 97).

5.4.4 Macro Definition Documentation

5.4.4.1 RTI_RoutingServiceAdapterPlugin_initialize

```
#define RTI_RoutingServiceAdapterPlugin_initialize(
              adapter )
```

Initializes the adapter plugin structure.

This macro must be called to initialize the return value of RTI_RoutingServiceAdapterPlugin_CreateFcn

Parameters

adapter	Pointer to the adapter plugin structure

See also

RTI_RoutingServiceAdapterPlugin_CreateFcn (p. 61)

5.4.5 Typedef Documentation

5.4.5.1 RTI_RoutingServiceStreamWriter

```
typedef void* RTI_RoutingServiceStreamWriter
```

StreamWriter.

A StreamWriter provides a way to write samples of a specific type in a data domain.

In the XML configuration file, StreamWriters are associated with the tag <output> within <route> and <auto_route>.

The StreamWriter type is a typedef to a 'void *' pointer. The concrete implementation is up to the adapter implementor.

5.4.5.2 RTI_RoutingServiceStreamWriter_WriteFcn

```
typedef int(* RTI_RoutingServiceStreamWriter_WriteFcn) ( RTI_RoutingServiceStreamWriter stream_writer, const RTI_RoutingServiceSample *sample_list, const RTI_RoutingServiceSampleInfo *info_list, int count, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that writes a collection of data samples to an output stream.

Required: Only when the adapter is used to write data.

Parameters

stream_writer	<<*in*>> Stream writer.
sample_list	<<*in*>> Array of samples. The data representation associated with the samples will be given by the value of the connection attribute com.rti.routingservice.adapter.data_representation_kind that is obtained using the associated **RTI_RoutingServiceStreamInfo** (p. 97).
info_list	<<*in*>> Array of sample infos. The info representation associated with the sample infos will be given by the value of the connection attribute com.rti.routingservice.adapter.info_representation_kind that is obtained using the associated **RTI_RoutingServiceStreamInfo** (p. 97).
count	<<*in*>> Number of samples in the sample list
env	<<*inout*>> Environment for error indications.

Returns

Number of samples written.

5.4.5.3 RTI_RoutingServiceStreamReader

```
typedef void*  RTI_RoutingServiceStreamReader
```

StreamReader.

A StreamReader provides a way to read samples of a specific type from a data domain.

In the XML configuration file, StreamReaders are associated with the tag <input> within <route> and <auto_route>.

The StreamReader type is a typedef to a 'void *' pointer. The concrete implementation is up to the adapter implementor.

5.4.5.4 RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback

```
typedef void(* RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback) ( RTI_Routing↩
ServiceStreamReader stream_reader, void *listener_data)
```

Prototype of the callback used to notify of new samples.

When a StreamReader receives new data, it will use this callback to notify RTI Routing Service that there are new samples.

Required: Only when the adapter is used to read data.

Parameters

stream_reader	<<*in*>> Stream reader.
listener_data	<<*inout*>> Data associated with the listener when the listener is set.

5.4.5.5 RTI_RoutingServiceStreamReader_ReadFcn

```
typedef void(* RTI_RoutingServiceStreamReader_ReadFcn) ( RTI_RoutingServiceStreamReader stream↩
_reader,  RTI_RoutingServiceSample **sample_list,  RTI_RoutingServiceSampleInfo **info_list, int
*count,  RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that reads a collection of data samples and sample infos from an input stream.

When RTI Routing Service is done using the samples, it will 'return the loan' to the StreamReader by calling **RTI_↩ RoutingServiceStreamReader_ReturnLoanFcn** (p. 53).

Required: Only when the adapter is used to read data.

Parameters

stream_reader	<<*in*>> Stream reader.
sample_list	<<*out*>> Array that will hold the output samples. This array will be provided by the StreamReader. The contents of the array are typically structures of the type DDS_DynamicData (see the RTI Connext documentation). But in general, the data representation associated with the output samples will be given by the value of the connection attribute com.rti.routingservice.adapter.data_representation_kind that is obtained using the associated **RTI_RoutingServiceStreamInfo** (p. 97).
info_list	<<*out*>> Array that will hold the output sample infos. This array will be provided by the StreamReader. It can be NULL if there is no info associated to the samples. The contents of the array are typically structures of the type DDS_SampleInfo (see the RTI Connext documentation). But in general the info representation associated with the output sample infos will be given by the value of the connection attribute com.rti.routingservice.adapter.info_representation_kind that is obtained using the associated **RTI_RoutingServiceStreamInfo** (p. 97).
count	<<*out*>> Number of output samples. The value must be greater than or equal to zero.
env	<<*inout*>> Environment for error indications.

See also

RTI_RoutingServiceStreamReader_ReturnLoanFcn (p. 53)

5.4.5.6 RTI_RoutingServiceStreamReader_ReturnLoanFcn

```
typedef void(* RTI_RoutingServiceStreamReader_ReturnLoanFcn) ( RTI_RoutingServiceStreamReader
stream_reader,  RTI_RoutingServiceSample *sample_list,  RTI_RoutingServiceSampleInfo *info_list,
int count,  RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that returns the loan on the read samples and infos.

RTI Routing Service calls this method to indicate that it is done accessing the collection of data samples and sample infos obtained by an earlier invocation of **RTI_RoutingServiceStreamReader_ReadFcn** (p. 52).

Required: Only when the adapter is used to read data.

Parameters

stream_reader	<<*in*>> Stream reader.
sample_list	<<*in*>> Array of samples.
info_list	<<*in*>> Array of infos.
count	<<*in*>> Number of samples in the sample list.
env	<<*inout*>> Environment for error indications.

See also

RTI_RoutingServiceStreamReader_ReadFcn (p. 52)

5.4.5.7 RTI_RoutingServiceSession

```
typedef void* RTI_RoutingServiceSession
```

Session.

A Session is a concurrency unit within a connection that has an associated set of StreamReaders and StreamWriters. Access to the StreamReaders and StreamWriters in the same Session is serialized by RTI Routing Service.

In the XML configuration file, Sessions are associated with the tag <session> within a domain route. For each <session> tag, RTI Routing Service will create two adapter Sessions, one per connection.

The Session type is a typedef to a 'void *' pointer. The concrete implementation is up to the adapter implementor.

5.4.5.8 RTI_RoutingServiceConnection

```
typedef void* RTI_RoutingServiceConnection
```

Connection.

A Connection object provides access to a data domain (such as a DDS domain or a JMS network provider).

In the XML configuration file, Connections are created using the tag <connection> within a DomainRoute.

The Connection type is a typedef to a 'void *' pointer. The concrete implementation is up to the adapter implementor.

5.4.5.9 RTI_RoutingServiceConnection_CreateSessionFcn

```
typedef RTI_RoutingServiceSession( * RTI_RoutingServiceConnection_CreateSessionFcn) ( RTI_↩
RoutingServiceConnection connection, const struct RTI_RoutingServiceProperties *properties, RTI↩
_RoutingServiceEnvironment *env)
```

Prototype of the function that creates a Session.

A Session is a concurrency unit within a Connection that has an associated set of StreamReaders and StreamWriters. Access to the StreamReaders and StreamWriters in the same Session is serialized by RTI Routing Service.

Session objects are created when the associated routing service sessions are enabled.

In the XML configuration file, Sessions are associated with the tag <session> within a domain route.

Required: No

Parameters

connection	<<*in*>> Connection.
properties	<<*in*>> Configuration properties for the Session.
env	<<*inout*>> Environment for error indications.

Returns

New Session if successful. Otherwise, NULL.

See also

RTI_RoutingServiceConnection_DeleteSessionFcn (p. 55)

5.4.5.10 RTI_RoutingServiceConnection_DeleteSessionFcn

```
typedef void(* RTI_RoutingServiceConnection_DeleteSessionFcn) ( RTI_RoutingServiceConnection connection,
RTI_RoutingServiceSession session,  RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that deletes a Session.

Session objects are deleted when the routing service sessions that contain them are disabled.

Required: No

Parameters

connection	<<*in*>> Connection.
session	<<*in*>> Session to be deleted.
env	<<*inout*>> Environment for error indications.

See also

RTI_RoutingServiceConnection_CreateSessionFcn (p. 54)

5.4.5.11 RTI_RoutingServiceConnection_CreateStreamReaderFcn

```
typedef  RTI_RoutingServiceStreamReader( * RTI_RoutingServiceConnection_CreateStreamReaderFcn) (
RTI_RoutingServiceConnection connection,  RTI_RoutingServiceSession session, const struct  RTI↩
_RoutingServiceStreamInfo *stream_info, const struct  RTI_RoutingServiceProperties *properties,
```

```
const struct RTI_RoutingServiceStreamReaderListener *listener, RTI_RoutingServiceEnvironment
*env)
```

Prototype of the function that creates a StreamReader.

A StreamReader provides a way to read samples of a specific type from a data domain.

In the XML configuration file, StreamReaders are associated with the tag <input> within <route> or <auto_route>.

This function is called when the route is enabled and the 'creation mode' condition associated with the route's input becomes true.

Required: Only when the adapter is used to read data.

Parameters

connection	<<*in*>> Connection.
session	<<*in*>> Session associated with the StreamReader. This parameter is NULL if Sessions are not used by the adapter.
stream_info	<<*in*>> Name of the stream and type representation.
properties	<<*in*>> Configuration properties for the StreamReader.
listener	<<*in*>> The listener of the StreamReader used to notify the routing service when new data is available.
env	<<*inout*>> Environment for error indications.

Returns

New StreamReader if successful. Otherwise, NULL.

See also

RTI_RoutingServiceConnection_DeleteStreamReaderFcn (p. 56)

5.4.5.12 RTI_RoutingServiceConnection_DeleteStreamReaderFcn

```
typedef void(* RTI_RoutingServiceConnection_DeleteStreamReaderFcn) ( RTI_RoutingServiceConnection
connection, RTI_RoutingServiceStreamReader stream_reader, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that deletes a StreamReader.

A StreamReader object is deleted when the route that contains it is disabled, when the 'creation mode' condition associated with the route's input becomes false or when RTI Routing Service is closed.

Required: Only when the adapter is used to read data.

Parameters

connection	<<*in*>> Connection.
stream_reader	<<*in*>> StreamReader to be deleted.
env	<<*inout*>> Environment for error indications.

See also

RTI_RoutingServiceConnection_CreateStreamReaderFcn (p. 55)

5.4.5.13 RTI_RoutingServiceConnection_CreateStreamWriterFcn

```
typedef RTI_RoutingServiceStreamWriter( * RTI_RoutingServiceConnection_CreateStreamWriterFcn) (
RTI_RoutingServiceConnection connection, RTI_RoutingServiceSession session, const struct RTI↩
_RoutingServiceStreamInfo *stream_info, const struct RTI_RoutingServiceProperties *properties,
RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that creates a StreamWriter.

A StreamWriter provides a way to write samples of a specific type in a data domain.

In the XML configuration file, StreamWriters are associated with the tag <output> within <route> or <auto_route>.

This function is called when the route is enabled and the 'creation mode' condition associated with the route's output becomes true.

Required: Only when the adapter is used to write data.

Parameters

connection	<<*in*>> Connection.
session	<<*in*>> Session associated with the StreamWriter. This parameter is NULL if Sessions are not used by the adapter.
stream_info	<<*in*>> Name of the stream and type representation.
properties	<<*in*>> Configuration properties for the StreamWriter.
env	<<*inout*>> Environment for error indications.

Returns

New StreamWriter if successful. Otherwise, NULL.

See also

RTI_RoutingServiceConnection_DeleteStreamWriterFcn (p. 57)

5.4.5.14 RTI_RoutingServiceConnection_DeleteStreamWriterFcn

```
typedef void(* RTI_RoutingServiceConnection_DeleteStreamWriterFcn) ( RTI_RoutingServiceConnection
connection,  RTI_RoutingServiceStreamWriter stream_writer,  RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that deletes a StreamWriter.

A StreamWriter object is deleted when the route or domain route that contains it is disabled, when the 'creation mode' condition associated with the route's output becomes false or when RTI Routing Service is closed.

Required: Only when the adapter is used to write data.

Parameters

connection	<<*in*>> Connection.
stream_writer	<<*in*>> StreamWriter to be deleted.
env	<<*inout*>> Environment for error indications.

See also

RTI_RoutingServiceConnection_CreateStreamWriterFcn (p. 57)

5.4.5.15 RTI_RoutingServiceConnection_GetDiscoveryReaderFcn

```
typedef  RTI_RoutingServiceStreamReader( * RTI_RoutingServiceConnection_GetDiscoveryReaderFcn) (
RTI_RoutingServiceConnection connection,  RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that gets a built-in discovery StreamReader.

There are two built-in discovery StreamReaders:

The first StreamReader provides information about output streams. An output stream is a stream to which StreamWriters can write data. Disposed scenarios, where an output streams dissapears, are also notified using this StreamReader.

The second StreamReader provides information about input streams. An input stream is a stream from which StreamReaders can read data. Disposed scenarios, where an output streams dissapears, are also notified using this StreamReader.

The StreamReaderListeners associated with the built-in discovery StreamReaders are provided as parameters to **RTI_RoutingServiceAdapterPlugin_CreateConnectionFcn** (p. 121).

Required: No

The implementation of this function is optional. However, if none of the adapters in a domain route implement the discovery API, the routes' types must be declared in the XML configuration file.

Parameters

connection	<<*in*>> Connection.
env	<<*inout*>> Environment for error indications.

Returns

Built-in dicovery StreamReader if successful. Otherwise, NULL.

5.4.5.16 RTI_RoutingServiceConnection_CopyTypeRepresentationFcn

```
typedef RTI_RoutingServiceTypeRepresentation( * RTI_RoutingServiceConnection_CopyTypeRepresentation↩
Fcn) ( RTI_RoutingServiceConnection connection, RTI_RoutingServiceTypeRepresentationKind type↩
_representation_kind, RTI_RoutingServiceTypeRepresentation type_representation, RTI_Routing↩
ServiceEnvironment *env)
```

Prototype of the function that copies a type representation.

This function is part of the adapter discovery API and is used by RTI Routing Service to copy the type representation associated with the discovered streams.

Required: No (Tied to the implementation **RTI_RoutingServiceConnection_GetDiscoveryReaderFcn** (p. 58)).

Parameters

connection	<<*in*>> Connection.
type_representation_kind	<<*in*>> Type representation kind.
type_representation	<<*in*>> Type representation to be copied.
env	<<*inout*>> Environment for error indications.

See also

Standard Type Representation Kinds (p. 75)

RTI_RoutingServiceConnection_DeleteTypeRepresentationFcn (p. 59)

5.4.5.17 RTI_RoutingServiceConnection_DeleteTypeRepresentationFcn

```
typedef void(* RTI_RoutingServiceConnection_DeleteTypeRepresentationFcn) ( RTI_RoutingService↩
Connection connection, RTI_RoutingServiceTypeRepresentationKind type_representation_kind, RTI_↩
RoutingServiceTypeRepresentation type_representation, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that deletes a type representation.

This function is part of the adapter discovery API.

Required: No (Tied to the implementation **RTI_RoutingServiceConnection_GetDiscoveryReaderFcn** (p. 58)).

Parameters

connection	<<*in*>> Connection.
type_representation_kind	<<*in*>> Type representation kind.
type_representation	<<*in*>> Type representation to be deleted.
env	<<*inout*>> Environment for error indications.

See also

Standard Type Representation Kinds (p. 75)

RTI_RoutingServiceConnection_CopyTypeRepresentationFcn (p. 59)

5.4.5.18 RTI_RoutingServiceAdapterEntity

```
typedef void*  RTI_RoutingServiceAdapterEntity
```

Adapter entity.

The adapter entities are:

- Connection
- Session
- StreamReader
- StreamWriter

5.4.5.19 RTI_RoutingServiceAdapterEntity_UpdateFcn

```
typedef void(* RTI_RoutingServiceAdapterEntity_UpdateFcn) ( RTI_RoutingServiceAdapterEntity entity,
const struct  RTI_RoutingServiceProperties *properties,  RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that updates the configuration of an adapter entity.

This function is called when remote administration is used.

Adapter entities that can be updated are:

- Connection
- Session
- StreamReader
- StreamWriter

Required: No. Implement this function only when remote configuration is needed.

Parameters

entity	<<*in*>> Entity.
properties	<<*in*>> New configuration properties.
env	<<*inout*>> Environment for error indications.

5.4.5.20 RTI_RoutingServiceAdapterPlugin_DeleteFcn

```
typedef void(* RTI_RoutingServiceAdapterPlugin_DeleteFcn) (struct  RTI_RoutingServiceAdapterPlugin
*plugin,  RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that deletes an adapter plugin.

Adapter plugins are deleted when RTI Routing Service is closed.

Required: yes

Parameters

plugin	<<*in*>> Adapter plugin to be deleted.
env	<<*inout*>> Environment for error indications.

See also

RTI_RoutingServiceAdapterPlugin_DeleteFcn (p. 61)

5.4.5.21 RTI_RoutingServiceAdapterPlugin_CreateFcn

```
typedef struct  RTI_RoutingServiceAdapterPlugin *( * RTI_RoutingServiceAdapterPlugin_CreateFcn)
(const struct  RTI_RoutingServiceProperties *properties,  RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that creates an adapter plugin.

The name of the function that implements this prototype must be provided to RTI Routing Service using the tag <create_function> when the adapter plugin is registered. For example:

```
<dds>
    ...
    <plugin_library name="MyAdapterLib">
        <adapter_plugin name="MyAdapterPlugin">
           <dll>mycadapter</dll>
           <create_function>
               MyAdapterPlugin_create
```

```
            </create_function>
        </adapter_plugin>
        ...
    </plugin_library>
    ...
    <routing_service>
    ...
    </routing_service>
    ...
</dds>
```

Required: yes

Parameters

properties	Configuration properties for the adapter.
env	<<*inout*>> Environment for error indications.

Returns

New plugin instance if successful. Otherwise, NULL.

See also

RTI_RoutingServiceAdapterPlugin_DeleteFcn (p. 61)

RTI_RoutingServiceAdapterPlugin_initialize (p. 50)

5.4.6 Variable Documentation

5.4.6.1 on_data_available

RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback RTI_RoutingServiceStreamReader↩
Listener::on_data_available

Prototype of the callback used to notify of new samples.

When a StreamReader receives new data, it will use this callback to notify RTI Routing Service that there are new samples.

Required: Only when the adapter is used to read data.

Parameters

stream_reader	<<*in*>> Stream reader.
listener_data	<<*inout*>> Data associated with the listener when the listener is set.

5.5 RTI Routing Service

RTI Routing Service.

Modules

- **RTI Routing Service Processor API**
  Entity responsible for administering inputs and outputs.
- **RTI Routing Library API**
  Prototype of the function that gets called upon reception of the shutdown command.
- **RTI Routing Service Transformation API**
  This module describes the Transformation API.
- **RTI Routing Service Adapter API**
  This module describes the C Adapter API.
- **RTI Routing Service Infrastructure**
  This module contains common definitions used across other functional modules.

5.5.1 Detailed Description

RTI Routing Service.

5.6 RTI Routing Service Infrastructure

This module contains common definitions used across other functional modules.

Modules

- **Standard Type Representation Kinds**
  Standard type Representation kinds.
- **Standard Data Representation Kinds**
  Standard data representation kinds.
- **Standard Error Codes**
  Standard error codes.

Data Structures

- struct **RTI_RoutingServiceNameValue**
  Configuration property.
- struct **RTI_RoutingServiceProperties**
  Set of configuration properties.
- struct **RTI_RoutingServiceVersion**
  Represents the version of a plugin or RTI Routing Service itself.
- struct **RTI_RoutingServiceTypeInfo**
  Type information.
- struct **RTI_RoutingServiceStringSeq**
  Definition of a String sequence.
- struct **RTI_RoutingServiceStreamInfo**
  Stream information.

Macros

- #define **RTI_USER_DLL_EXPORT**

  Utility macro that can be used to export symbols in a shared library on Windows platform. For non-windows, the definion of this macro is empty.
- #define **RTI_UNUSED_PARAMETER**(x) (void)(x)

  This can be used by C client and example code to avoid unused parameter warnings in the compiler. This is not strictly necessary in C++, where just removing the parameter name should yield the same result.
- #define **RTI_ROUTING_SERVICE_ERROR_MAX_LENGTH** 1024

  Maximum length of an error message.
- #define **RTI_ROUTING_SERVICE_APP_NAME_PROPERTY_NAME** RTI_ROUTING_SERVICE_PROPERTY↩ _PREFIX".app_name"

  Name of the property that provides the RTI Routing Service given application name.
- #define **RTI_ROUTING_SERVICE_GROUP_PROPERTY_NAME** RTI_ROUTING_SERVICE_PROPERTY_↩ PREFIX".group_name"
- #define **RTI_ROUTING_SERVICE_VERSION_PROPERTY_NAME** RTI_ROUTING_SERVICE_PROPERTY_↩ PREFIX".version"

  Name of the property that provides the RTI Routing Service version as string.
- #define **RTI_ROUTING_SERVICE_VERBOSITY_PROPERTY_NAME** RTI_ROUTING_SERVICE_PROPERTY↩ _PREFIX".verbosity"

  Name of the property that provides verbosity in use by RTI Routing Service.
- #define **RTI_ROUTING_SERVICE_ENTITY_RESOURCE_NAME_PROPERTY_NAME** RTI_ROUTING_↩ SERVICE_PROPERTY_PREFIX".entity.resource_name"

  Name of the property that provides the resource name of the entity that owns the adapter entity.

Typedefs

- typedef struct RTI_RoutingServiceEnvironmentImpl **RTI_RoutingServiceEnvironment**

  The environment permits the return of error information in the RTI Routing Service API and information retreival (version and verbosity).
- typedef int **RTI_RoutingServiceTypeRepresentationKind**

  Type representation kind.
- typedef void * **RTI_RoutingServiceTypeRepresentation**

  Type representation.
- typedef int **RTI_RoutingServiceDataRepresentationKind**

  Data representation kind.
- typedef void * **RTI_RoutingServiceSample**

  Stream sample.
- typedef void * **RTI_RoutingServiceSampleInfo**

  Stream sample info. In DDS, this is a DDS_SampleInfo object.

Enumerations

- enum **RTI_RoutingServiceVerbosity** {
  RTI_ROUTING_SERVICE_VERBOSITY_NONE = 0 ,
  RTI_ROUTING_SERVICE_VERBOSITY_EXCEPTION ,
  RTI_ROUTING_SERVICE_VERBOSITY_WARN ,
  RTI_ROUTING_SERVICE_VERBOSITY_INFO ,
  RTI_ROUTING_SERVICE_VERBOSITY_DEBUG }

  Verbosity used by Routing Service.

Functions

- void **RTI_RoutingServiceLogger_log** (NDDS_Config_LogLevel log_level, const char *format,...)

  Logs as message with the specified level.
- const char * **RTI_RoutingServiceProperties_lookup_property** (const struct **RTI_RoutingServiceProperties** *self, const char *name)

  Searches for a property given its name.
- void **RTI_RoutingServiceEnvironment_set_error_w_params** (**RTI_RoutingServiceEnvironment** *self, DDS_Boolean overwrite, int error_code, int native_error_code, const char *error_format,...)

  Assigns an error into the environment.
- void **RTI_RoutingServiceEnvironment_set_error** (**RTI_RoutingServiceEnvironment** *self, const char *error_format,...)

  Assigns an error into the environment.
- void **RTI_RoutingServiceEnvironment_fatal_error** (**RTI_RoutingServiceEnvironment** *self, const char *error_format,...)

  Assigns a fatal error into the environment.
- void **RTI_RoutingServiceEnvironment_clear_error** (**RTI_RoutingServiceEnvironment** *self)

  Clears an error (if any) set in this environment.
- **RTI_RoutingServiceVerbosity** **RTI_RoutingServiceEnvironment_get_verbosity** (const **RTI_Routing↩ ServiceEnvironment** *self)

  Retrieves the verbosity that Routing Service is using.
- DDS_Boolean **RTI_RoutingServiceEnvironment_error_occurred** (const **RTI_RoutingServiceEnvironment** *self)

  Checks whether an error has been set in this environment.
- const char * **RTI_RoutingServiceEnvironment_get_error_message** (const **RTI_RoutingService↩ Environment** *self)

  Returns the error message this environment contains.
- struct **RTI_RoutingServiceStreamInfo** * **RTI_RoutingServiceStreamInfo_new_discovered** (const char *stream_name, const char *registered_type_name, **RTI_RoutingServiceTypeRepresentationKind** type_↩ representation_kind, **RTI_RoutingServiceTypeRepresentation** type_representation)

  Creates a stream info for a newly discovered stream.
- struct **RTI_RoutingServiceStreamInfo** * **RTI_RoutingServiceStreamInfo_new_disposed** (const char *stream_name)

  Creates a stream info for a disposed stream. Disposed streams are no longer available in a data domain.
- void **RTI_RoutingServiceStreamInfo_delete** (struct **RTI_RoutingServiceStreamInfo** *self)

  Destroys a stream info.

5.6.1 Detailed Description

This module contains common definitions used across other functional modules.

5.6.2 Macro Definition Documentation

5.6.2.1 RTI_USER_DLL_EXPORT

```
#define RTI_USER_DLL_EXPORT
```

Utility macro that can be used to export symbols in a shared library on Windows platform. For non-windows, the definion of this macro is empty.

5.6.2.2 RTI_UNUSED_PARAMETER

```
#define RTI_UNUSED_PARAMETER(
              x ) (void)(x)
```

This can be used by C client and example code to avoid unused parameter warnings in the compiler. This is not strictly necessary in C++, where just removing the parameter name should yield the same result.

5.6.2.3 RTI_ROUTING_SERVICE_ERROR_MAX_LENGTH

```
#define RTI_ROUTING_SERVICE_ERROR_MAX_LENGTH 1024
```

Maximum length of an error message.

Error messages longer than this value are truncated.

5.6.2.4 RTI_ROUTING_SERVICE_APP_NAME_PROPERTY_NAME

```
#define RTI_ROUTING_SERVICE_APP_NAME_PROPERTY_NAME  RTI_ROUTING_SERVICE_PROPERTY_PREFIX".app_name"
```

Name of the property that provides the RTI Routing Service given application name.

5.6.2.5 RTI_ROUTING_SERVICE_GROUP_PROPERTY_NAME

```
#define RTI_ROUTING_SERVICE_GROUP_PROPERTY_NAME  RTI_ROUTING_SERVICE_PROPERTY_PREFIX".group_name"
```

Name of the property that provides the configured group name.

5.6.2.6 RTI_ROUTING_SERVICE_VERSION_PROPERTY_NAME

```
#define RTI_ROUTING_SERVICE_VERSION_PROPERTY_NAME  RTI_ROUTING_SERVICE_PROPERTY_PREFIX".version"
```

Name of the property that provides the RTI Routing Service version as string.

5.6.2.7 RTI_ROUTING_SERVICE_VERBOSITY_PROPERTY_NAME

```
#define RTI_ROUTING_SERVICE_VERBOSITY_PROPERTY_NAME  RTI_ROUTING_SERVICE_PROPERTY_PREFIX".verbosity"
```

Name of the property that provides verbosity in use by RTI Routing Service.

It can take on of the following stings:

- NONE
- EXCEPTION
- WARN
- INFO
- DEBUG.

5.6.2.8 RTI_ROUTING_SERVICE_ENTITY_RESOURCE_NAME_PROPERTY_NAME

```
#define RTI_ROUTING_SERVICE_ENTITY_RESOURCE_NAME_PROPERTY_NAME  RTI_ROUTING_SERVICE_PROPERTY_↩
PREFIX".entity.resource_name"
```

Name of the property that provides the resource name of the entity that owns the adapter entity.

5.6.3 Typedef Documentation

5.6.3.1 RTI_RoutingServiceEnvironment

```
typedef struct RTI_RoutingServiceEnvironmentImpl  RTI_RoutingServiceEnvironment
```

The environment permits the return of error information in the RTI Routing Service API and information retreival (version and verbosity).

This is the last parameter of each operation.

See also

RTI_RoutingServiceEnvironment_set_error (p. 71)

RTI_RoutingServiceEnvironment_get_verbosity (p. 73)

5.6.3.2 RTI_RoutingServiceTypeRepresentationKind

```
typedef int  RTI_RoutingServiceTypeRepresentationKind
```

Type representation kind.

The range [0-100] is reserved for RTI use. Within that range are some predefined type representations (**Standard Type Representation Kinds** (p. 75)).

5.6.3.3 RTI_RoutingServiceTypeRepresentation

```
typedef void*  RTI_RoutingServiceTypeRepresentation
```

Type representation.

If the representation kind is **RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_DYNAMIC_TYPE** (p. 76), the representation will be an RTI Connext TypeCode.

If the representation kind is **RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_XML** (p. 76), the representation will be an XML string.

5.6.3.4 RTI_RoutingServiceDataRepresentationKind

```
typedef int  RTI_RoutingServiceDataRepresentationKind
```

Data representation kind.

The range [0-100] is reserved for RTI use. Within that range are some predefined data representations (**Standard Data Representation Kinds** (p. 77)).

5.6.3.5 RTI_RoutingServiceSample

```
typedef void*  RTI_RoutingServiceSample
```

Stream sample.

Samples are data messages generated by adapters and transformations.

If the representation kind is **RTI_ROUTING_SERVICE_DATA_REPRESENTATION_DYNAMIC_DATA** (p. 77), the sample will be a DynamicData object.

If the representation kind is **RTI_ROUTING_SERVICE_DATA_REPRESENTATION_XML** (p. 77), the sample will be an XML string.

5.6.3.6 RTI_RoutingServiceSampleInfo

```
typedef void*  RTI_RoutingServiceSampleInfo
```

Stream sample info. In DDS, this is a DDS_SampleInfo object.

5.6.4 Enumeration Type Documentation

5.6.4.1 RTI_RoutingServiceVerbosity

enum **RTI_RoutingServiceVerbosity**

Verbosity used by Routing Service.

Enumerator

RTI_ROUTING_SERVICE_VERBOSITY_NONE	No logging (-verbosity 0)
RTI_ROUTING_SERVICE_VERBOSITY_EXCEPTION	Exceptions (-verbosity 1)
RTI_ROUTING_SERVICE_VERBOSITY_WARN	Warnings (-verbosity 2)
RTI_ROUTING_SERVICE_VERBOSITY_INFO	Information (-verbosity 3)
RTI_ROUTING_SERVICE_VERBOSITY_DEBUG	Debug information (-verbosity 5 and 6)

5.6.5 Function Documentation

5.6.5.1 RTI_RoutingServiceLogger_log()

```
void RTI_RoutingServiceLogger_log (
            NDDS_Config_LogLevel log_level,
            const char * format,
             ... )
```

Logs as message with the specified level.

The message is specified with a format and a format parameter, in a similar fashion to the standard C printf() operation.

The generated log will be part of logging stream of the running RoutingService, if the `log_level` is part of the configured verbosity. The result log message may include additional information according to the Connext logging configuration, such as Advlog Context, thread ID, line number, etc. Additionally, the result log message will contain a newline character at the end, so the format does not need to contain it.

Parameters

in	*log_level*	Log level associated to the message
in	*format*	message format specification
in	...	variable-length argument (stdarg)

5.6.5.2 RTI_RoutingServiceProperties_lookup_property()

```
const char * RTI_RoutingServiceProperties_lookup_property (
            const struct  RTI_RoutingServiceProperties * self,
            const char * name )
```

Searches for a property given its name.

Parameters

self	<<*in*>> Cannot be NULL.
name	<<*in*>> Property name. Cannot be NULL.

Returns

On success, the function returns the value of the first property with the given name. Otherwise, the function returns NULL.

5.6.5.3 RTI_RoutingServiceEnvironment_set_error_w_params()

```
void RTI_RoutingServiceEnvironment_set_error_w_params (
              RTI_RoutingServiceEnvironment * self,
             DDS_Boolean overwrite,
             int error_code,
             int native_error_code,
             const char * error_format,
              ... )
```

Assigns an error into the environment.

Routing Service will consider that a function call failed when an error has been set into the environment

Parameters

self	<<*in*>> Cannot be NULL.
overwrite	<<*in*>> If the environment already contains an error, setting this parameter to DDS_BOOLEAN_TRUE will overwrite it.
error_code	<<*in*>> One of the **Standard Error Codes** (p. 78).
native_error_code	<<*in*>> Native error code (specific to the transformation or adapter plugin).
error_format	<<*in*>> String that contains the error text. It can optionally contain format tags that are substituted by the values specified in subsequent argument(s). Cannot be NULL. The format tags are the same tags used by the function printf in the ANSI C standard.

See also

RTI_RoutingServiceEnvironment_set_error (p. 71)

5.6.5.4 RTI_RoutingServiceEnvironment_set_error()

```
void RTI_RoutingServiceEnvironment_set_error (
              RTI_RoutingServiceEnvironment * self,
```

```
            const char * error_format,
             ...  )
```

Assigns an error into the environment.

Routing Service will consider that a function call failed when an error has been set into the environment

This function sets the error code to **RTI_ROUTING_SERVICE_ERROR** (p. 78) and the native error code to 0.

If the environment already contains an error, the error is not overwritten.

Parameters

self	<<*in*>> Cannot be NULL.
error_format	<<*in*>> String that contains the error text. It can optionally contain format tags that are substituted by the values specified in subsequent argument(s). Cannot be NULL. The format tags are the same tags used by the function printf in the ANSI C standard.

See also

RTI_RoutingServiceEnvironment_set_error_w_params (p. 71)

5.6.5.5 RTI_RoutingServiceEnvironment_fatal_error()

```
void RTI_RoutingServiceEnvironment_fatal_error (
             RTI_RoutingServiceEnvironment * self,
            const char * error_format,
             ...  )
```

Assigns a fatal error into the environment.

Routing Service will consider that a function call failed when an error has been set into the environment. Additionally, a fatal error will cause the routing service to call the shutdown hook.

This function sets the error code to **RTI_ROUTING_SERVICE_FATAL_ERROR** (p. 78) and the native error code to 0. Any previous error in the environment will be overwritten.

Parameters

self	<<*in*>> Cannot be NULL.
error_format	<<*in*>> String that contains the error text. It can optionally contain format tags that are substituted by the values specified in subsequent argument(s). Cannot be NULL. The format tags are the same tags used by the function printf in the ANSI C standard.

See also

RTI_RoutingServiceEnvironment_set_error_w_params (p. 71)

5.6.5.6 RTI_RoutingServiceEnvironment_clear_error()

```
void RTI_RoutingServiceEnvironment_clear_error (
             RTI_RoutingServiceEnvironment * self )
```

Clears an error (if any) set in this environment.

Parameters

self	<<*in*>> Cannot be NULL.

5.6.5.7 RTI_RoutingServiceEnvironment_get_verbosity()

```
RTI_RoutingServiceVerbosity RTI_RoutingServiceEnvironment_get_verbosity (
            const  RTI_RoutingServiceEnvironment * self )
```

Retrieves the verbosity that Routing Service is using.

Parameters

self	<<*in*>> Cannot be NULL.

Returns

The verbosity

5.6.5.8 RTI_RoutingServiceEnvironment_error_occurred()

```
DDS_Boolean RTI_RoutingServiceEnvironment_error_occurred (
            const  RTI_RoutingServiceEnvironment * self )
```

Checks whether an error has been set in this environment.

Parameters

self	<<*in*>> Cannot be NULL.

Returns

A value different than zero if true

5.6.5.9 RTI_RoutingServiceEnvironment_get_error_message()

```
const char * RTI_RoutingServiceEnvironment_get_error_message (
            const  RTI_RoutingServiceEnvironment * self )
```

Returns the error message this environment contains.

Parameters

self	<<*in*>> Cannot be NULL.

Returns

The error message or NULL if no error is set

5.6.5.10 RTI_RoutingServiceStreamInfo_new_discovered()

```
struct  RTI_RoutingServiceStreamInfo * RTI_RoutingServiceStreamInfo_new_discovered (
             const char * stream_name,
             const char * registered_type_name,
              RTI_RoutingServiceTypeRepresentationKind type_representation_kind,
              RTI_RoutingServiceTypeRepresentation type_representation )
```

Creates a stream info for a newly discovered stream.

Parameters

stream_name	<<*in*>> Stream name. Cannot be NULL.
registered_type_name	<<*in*>> Type name. Cannot be NULL.
type_representation_kind	<<*in*>> Type representation kind.
type_representation	<<*in*>> Type representation. Cannot be NULL.

Returns

Newly created stream info, or NULL if there is an error.

5.6.5.11 RTI_RoutingServiceStreamInfo_new_disposed()

```
struct  RTI_RoutingServiceStreamInfo * RTI_RoutingServiceStreamInfo_new_disposed (
            const char * stream_name )
```

Creates a stream info for a disposed stream. Disposed streams are no longer available in a data domain.

Parameters

stream_name	<<*in*>> Stream name. Cannot be NULL.

Returns

Newly created stream info, or NULL if there is an error.

5.6.5.12 RTI_RoutingServiceStreamInfo_delete()

```
void RTI_RoutingServiceStreamInfo_delete (
            struct  RTI_RoutingServiceStreamInfo * self )
```

Destroys a stream info.

Parameters

self	<<*in*>> Cannot be NULL.

5.7 Standard Type Representation Kinds

Standard type Representation kinds.

Macros

- #define **RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_DYNAMIC_TYPE**

  Dynamic type representation.

- #define **RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_XML**

  [Not supported] *XML type representation.*
- #define **RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_JAVA_OBJECT**

  [Not supported] *Java object type representation.*

5.7.1 Detailed Description

Standard type Representation kinds.

5.7.2 Macro Definition Documentation

5.7.2.1 RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_DYNAMIC_TYPE

```
#define RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_DYNAMIC_TYPE
```

Dynamic type representation.

Types with this representation are provided as RTI Connext TypeCodes.

For more information about TypeCodes, see the "Data Types and DDS Data Samples" chapter in the RTI Connext DDS Core Libraries User's Manual.

5.7.2.2 RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_XML

```
#define RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_XML
```

[Not supported] XML type representation.

Types with this representation are provided as XML strings. The XML schema is the RTI Connext XML schema for type representation.

For more information about XML representation, see the "Data Types and DDS Data Samples" chapter in the RTI Connext DDS Core Libraries User's Manual.

5.7.2.3 RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_JAVA_OBJECT

```
#define RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_JAVA_OBJECT
```

[Not supported] Java object type representation.

Types with this representation are provided as Java objects.

5.8 Standard Data Representation Kinds

Standard data representation kinds.

Macros

- #define **RTI_ROUTING_SERVICE_DATA_REPRESENTATION_DYNAMIC_DATA**
  Dynamic data representation.
- #define **RTI_ROUTING_SERVICE_DATA_REPRESENTATION_XML**
  [Not supported] *XML data representation.*
- #define **RTI_ROUTING_SERVICE_DATA_REPRESENTATION_JAVA_OBJECT**
  [Not supported] *Java object data representation.*

5.8.1 Detailed Description

Standard data representation kinds.

5.8.2 Macro Definition Documentation

5.8.2.1 RTI_ROUTING_SERVICE_DATA_REPRESENTATION_DYNAMIC_DATA

```
#define RTI_ROUTING_SERVICE_DATA_REPRESENTATION_DYNAMIC_DATA
```

Dynamic data representation.

Samples with this representation are provided as RTI Connext DynamicData objects.

For more information about TypeCodes, see the "Data Types and DDS Data Samples" chapter in the RTI Connext DDS Core Libraries User's Manual.

5.8.2.2 RTI_ROUTING_SERVICE_DATA_REPRESENTATION_XML

```
#define RTI_ROUTING_SERVICE_DATA_REPRESENTATION_XML
```

[Not supported] XML data representation.

Samples with this representation are provided as XML strings. These samples are created according to the RTI Connext XML schema for type representation.

For more information about XML representation, see the "Data Types and DDS Data Samples" chapter in the RTI Connext DDS Core Libraries User's Manual.

5.8.2.3 RTI_ROUTING_SERVICE_DATA_REPRESENTATION_JAVA_OBJECT

```
#define RTI_ROUTING_SERVICE_DATA_REPRESENTATION_JAVA_OBJECT
```

[Not supported] Java object data representation.

Samples with this representation are provided as Java objects.

5.9 Standard Error Codes

Standard error codes.

Macros

- #define **RTI_ROUTING_SERVICE_ERROR**
  Generic, unspecified error.
- #define **RTI_ROUTING_SERVICE_FATAL_ERROR**
  Fatal error. It will cause the routing service to shut down.

5.9.1 Detailed Description

Standard error codes.

5.9.2 Macro Definition Documentation

5.9.2.1 RTI_ROUTING_SERVICE_ERROR

```
#define RTI_ROUTING_SERVICE_ERROR
```

Generic, unspecified error.

5.9.2.2 RTI_ROUTING_SERVICE_FATAL_ERROR

```
#define RTI_ROUTING_SERVICE_FATAL_ERROR
```

Fatal error. It will cause the routing service to shut down.

Chapter 6

Data Structure Documentation

6.1 RTI_RoutingService Struct Reference

RTI Routing Service.

`#include <routingservice_service.h>`

6.1.1 Detailed Description

RTI Routing Service.

6.2 RTI_RoutingServiceAdapterPlugin Struct Reference

Adapter plugin.

`#include <routingservice_adapter.h>`

Data Fields

- struct **RTI_RoutingServiceVersion plugin_version**
  The version of this adapter plugin.
- **RTI_RoutingServiceAdapterPlugin_DeleteFcn adapter_plugin_delete**
  Handles the deletion of the adapter plugin (required).
- **RTI_RoutingServiceAdapterPlugin_CreateConnectionFcn adapter_plugin_create_connection**
  Handles the creation of connections (required).
- **RTI_RoutingServiceAdapterPlugin_DeleteConnectionFcn adapter_plugin_delete_connection**
  Handles the deletion of connections (required).
- **RTI_RoutingServiceConnection_CreateSessionFcn connection_create_session**
  Handles the creation of sessions (optional).

- **RTI_RoutingServiceConnection_DeleteSessionFcn connection_delete_session**

  Handles the deletion of sessions (optional).
- **RTI_RoutingServiceConnection_CreateStreamReaderFcn connection_create_stream_reader**

  Handles the creation of stream readers (optional for write-only adapters).
- **RTI_RoutingServiceConnection_DeleteStreamReaderFcn connection_delete_stream_reader**

  Handles the deletion of stream readers (optional for write-only adapters).
- **RTI_RoutingServiceConnection_CreateStreamWriterFcn connection_create_stream_writer**

  Handles the creation of stream writers (optional for read-only adapters).
- **RTI_RoutingServiceConnection_DeleteStreamWriterFcn connection_delete_stream_writer**

  Handles the deletion of stream writers (optional for read-only adapters).
- **RTI_RoutingServiceConnection_GetDiscoveryReaderFcn connection_get_input_stream_discovery_reader**

  Gets the input stream discovery reader (optional).
- **RTI_RoutingServiceConnection_GetDiscoveryReaderFcn connection_get_output_stream_discovery_reader**

  Gets the output stream discovery reader (optional).
- **RTI_RoutingServiceConnection_CopyTypeRepresentationFcn connection_copy_type_representation**

  Handles the copy of type representations (optional).
- **RTI_RoutingServiceConnection_DeleteTypeRepresentationFcn connection_delete_type_representation**

  Handles the deletion of type representations (optional).
- RTI_RoutingServiceConnection_ToStringFcn **connection_to_string**

  Returns the string representation of a connection for logging purposes (optional).
- **RTI_RoutingServiceAdapterEntity_UpdateFcn connection_update**

  Handles configuration changes in a connection (optional).
- **RTI_RoutingServiceAdapterEntity_UpdateFcn session_update**

  ***[Not supported]** Handles configuration changes in a session (optional).*
- **RTI_RoutingServiceStreamReader_ReadFcn stream_reader_read**

  Reads from an input stream (optional for write-only adapters).
- **RTI_RoutingServiceStreamReader_ReturnLoanFcn stream_reader_return_loan**

  Returns the loan on the read samples and infos (optional for write-only adapters).
- **RTI_RoutingServiceAdapterEntity_UpdateFcn stream_reader_update**

  ***[Not supported]** Handles configuration changes in a stream reader (optional).*
- **RTI_RoutingServiceStreamWriter_WriteFcn stream_writer_write**

  Writes to an output stream (optional for read-only adapters).
- **RTI_RoutingServiceAdapterEntity_UpdateFcn stream_writer_update**

  ***[Not supported]** Handles configuration changes in a stream writer (optional).*
- void * **user_object**

  A place for adapter implementors to keep a pointer to data that may be needed by the implementation.

6.2.1 Detailed Description

Adapter plugin.

This structure contains the plugin implementation as a set of function pointers.

C adapters are registered with RTI Routing Service using the XML tag <adapter_plugin>

For example:

```
<dds>
    ...
    <plugin_library name="MyAdapterLib">
        <adapter_plugin name="MyAdapterPlugin">
           <dll>mycadapter</dll>
           <create_function>
               MyAdapterPlugin_create
           </create_function>
        </adapter_plugin>
        ...
    </plugin_library>
    ...
    <routing_service>
    ...
    </routing_service>
    ...
</dds>
```

In the previous example, MyAdapterPlugin_create is the function that creates and instance of the adapter plugin.

This function must have the following prototype:

See also

RTI_RoutingServiceAdapterPlugin_CreateFcn (p. 61)

6.2.2 Field Documentation

6.2.2.1 plugin_version

`struct` **RTI_RoutingServiceVersion** RTI_RoutingServiceAdapterPlugin::plugin_version

The version of this adapter plugin.

6.2.2.2 adapter_plugin_delete

RTI_RoutingServiceAdapterPlugin_DeleteFcn RTI_RoutingServiceAdapterPlugin::adapter_plugin_delete

Handles the deletion of the adapter plugin (required).

6.2.2.3 adapter_plugin_create_connection

RTI_RoutingServiceAdapterPlugin_CreateConnectionFcn RTI_RoutingServiceAdapterPlugin::adapter_↩plugin_create_connection

Handles the creation of connections (required).

6.2.2.4 adapter_plugin_delete_connection

RTI_RoutingServiceAdapterPlugin_DeleteConnectionFcn RTI_RoutingServiceAdapterPlugin::adapter_↩plugin_delete_connection

Handles the deletion of connections (required).

6.2.2.5 connection_create_session

RTI_RoutingServiceConnection_CreateSessionFcn RTI_RoutingServiceAdapterPlugin::connection_↩create_session

Handles the creation of sessions (optional).

6.2.2.6 connection_delete_session

RTI_RoutingServiceConnection_DeleteSessionFcn RTI_RoutingServiceAdapterPlugin::connection_↩delete_session

Handles the deletion of sessions (optional).

6.2.2.7 connection_create_stream_reader

RTI_RoutingServiceConnection_CreateStreamReaderFcn RTI_RoutingServiceAdapterPlugin::connection_↩create_stream_reader

Handles the creation of stream readers (optional for write-only adapters).

6.2.2.8 connection_delete_stream_reader

RTI_RoutingServiceConnection_DeleteStreamReaderFcn RTI_RoutingServiceAdapterPlugin::connection_↩delete_stream_reader

Handles the deletion of stream readers (optional for write-only adapters).

6.2.2.9 connection_create_stream_writer

RTI_RoutingServiceConnection_CreateStreamWriterFcn RTI_RoutingServiceAdapterPlugin::connection_↩
create_stream_writer

Handles the creation of stream writers (optional for read-only adapters).

6.2.2.10 connection_delete_stream_writer

RTI_RoutingServiceConnection_DeleteStreamWriterFcn RTI_RoutingServiceAdapterPlugin::connection_↩
delete_stream_writer

Handles the deletion of stream writers (optional for read-only adapters).

6.2.2.11 connection_get_input_stream_discovery_reader

RTI_RoutingServiceConnection_GetDiscoveryReaderFcn RTI_RoutingServiceAdapterPlugin::connection_↩
get_input_stream_discovery_reader

Gets the input stream discovery reader (optional).

6.2.2.12 connection_get_output_stream_discovery_reader

RTI_RoutingServiceConnection_GetDiscoveryReaderFcn RTI_RoutingServiceAdapterPlugin::connection_↩
get_output_stream_discovery_reader

Gets the output stream discovery reader (optional).

6.2.2.13 connection_copy_type_representation

RTI_RoutingServiceConnection_CopyTypeRepresentationFcn RTI_RoutingServiceAdapterPlugin::connection↩
_copy_type_representation

Handles the copy of type representations (optional).

6.2.2.14 connection_delete_type_representation

RTI_RoutingServiceConnection_DeleteTypeRepresentationFcn RTI_RoutingServiceAdapterPlugin::connection↩
_delete_type_representation

Handles the deletion of type representations (optional).

6.2.2.15 connection_to_string

RTI_RoutingServiceConnection_ToStringFcn RTI_RoutingServiceAdapterPlugin::connection_to_string

Returns the string representation of a connection for logging purposes (optional).

6.2.2.16 connection_update

RTI_RoutingServiceAdapterEntity_UpdateFcn RTI_RoutingServiceAdapterPlugin::connection_update

Handles configuration changes in a connection (optional).

6.2.2.17 session_update

RTI_RoutingServiceAdapterEntity_UpdateFcn RTI_RoutingServiceAdapterPlugin::session_update

[Not supported] Handles configuration changes in a session (optional).

6.2.2.18 stream_reader_read

RTI_RoutingServiceStreamReader_ReadFcn RTI_RoutingServiceAdapterPlugin::stream_reader_read

Reads from an input stream (optional for write-only adapters).

6.2.2.19 stream_reader_return_loan

RTI_RoutingServiceStreamReader_ReturnLoanFcn RTI_RoutingServiceAdapterPlugin::stream_reader_↩
return_loan

Returns the loan on the read samples and infos (optional for write-only adapters).

6.2.2.20 stream_reader_update

RTI_RoutingServiceAdapterEntity_UpdateFcn RTI_RoutingServiceAdapterPlugin::stream_reader_update

[Not supported] Handles configuration changes in a stream reader (optional).

6.2.2.21 stream_writer_write

RTI_RoutingServiceStreamWriter_WriteFcn RTI_RoutingServiceAdapterPlugin::stream_writer_write

Writes to an output stream (optional for read-only adapters).

6.2.2.22 stream_writer_update

RTI_RoutingServiceAdapterEntity_UpdateFcn RTI_RoutingServiceAdapterPlugin::stream_writer_update

[Not supported] Handles configuration changes in a stream writer (optional).

6.2.2.23 user_object

void* RTI_RoutingServiceAdapterPlugin::user_object

A place for adapter implementors to keep a pointer to data that may be needed by the implementation.

6.3 RTI_RoutingServiceLoanedSamples Struct Reference

A pair of sample and sample info arrays. They are assumed to be of the same length.

#include <routingservice_processor.h>

6.3.1 Detailed Description

A pair of sample and sample info arrays. They are assumed to be of the same length.

See also

RTI_RoutingServiceSample (p. 68)
RTI_RoutingServiceSampleInfo (p. 68)

6.4 RTI_RoutingServiceNameValue Struct Reference

Configuration property.

`#include <routingservice_infrastructure.h>`

Data Fields

- char * **name**
  Property name.
- void * **value**
  Property value.

6.4.1 Detailed Description

Configuration property.

A property is a name/value pair.

6.4.2 Field Documentation

6.4.2.1 name

`char* RTI_RoutingServiceNameValue::name`

Property name.

6.4.2.2 value

```
void* RTI_RoutingServiceNameValue::value
```

Property value.

6.5 RTI_RoutingServiceProcessor Struct Reference

Main Processor class.

```
#include <routingservice_processor.h>
```

Data Fields

- **RTI_RoutingServiceProcessor_OnRouteEventFcn on_route_event**
  *Handles event processing (**required**).*
- **RTI_RoutingServiceProcessor_UpdateFcn update**
  *Handles configuration changes in a Processor (**optional**).*
- void * **processor_data**
  Implementation data. Provided by implementors, and passed back in API calls.

6.5.1 Detailed Description

Main Processor class.

A Route can notify a Processor about different events related to inputs and outputs.

See also

RTI_RoutingServiceRouteEvent

6.5.2 Field Documentation

6.5.2.1 on_route_event

```
RTI_RoutingServiceProcessor_OnRouteEventFcn RTI_RoutingServiceProcessor::on_route_event
```

Handles event processing (**required**).

6.5.2.2 update

RTI_RoutingServiceProcessor_UpdateFcn RTI_RoutingServiceProcessor::update

Handles configuration changes in a Processor (**optional**).

6.5.2.3 processor_data

void* RTI_RoutingServiceProcessor::processor_data

Implementation data. Provided by implementors, and passed back in API calls.

6.6 RTI_RoutingServiceProcessorPlugin Struct Reference

ProcessorPlugin class.

#include <routingservice_processor.h>

Data Fields

- struct **RTI_RoutingServiceVersion plugin_version**
  The version of this ProcessorPlugin.
- **RTI_RoutingServiceProcessorPlugin_DeleteFcn plugin_delete**
  Handles the deletion of the ProcessorPlugin.
- **RTI_RoutingServiceProcessorPlugin_CreateProcessorFcn create_processor**
  Handles the creation of Processors.
- RTI_RoutingServiceProcessorPlugin_DeleteProcessorFcn **delete_processor**
  Handles the deletion of Processors.
- void * **processor_plugin_data**
  A placeholder for Processor implementors to keep a pointer to data that may be needed by the implementation.

6.6.1 Detailed Description

ProcessorPlugin class.

ProcessorPlugins are the entry points to the Processor API. They provide a factory for Processors.

6.6.2 Field Documentation

6.6.2.1 plugin_version

struct **RTI_RoutingServiceVersion** RTI_RoutingServiceProcessorPlugin::plugin_version

The version of this ProcessorPlugin.

6.6.2.2 plugin_delete

RTI_RoutingServiceProcessorPlugin_DeleteFcn RTI_RoutingServiceProcessorPlugin::plugin_delete

Handles the deletion of the ProcessorPlugin.

6.6.2.3 create_processor

RTI_RoutingServiceProcessorPlugin_CreateProcessorFcn RTI_RoutingServiceProcessorPlugin::create_↩
processor

Handles the creation of Processors.

6.6.2.4 delete_processor

RTI_RoutingServiceProcessorPlugin_DeleteProcessorFcn RTI_RoutingServiceProcessorPlugin::delete_↩
processor

Handles the deletion of Processors.

6.6.2.5 processor_plugin_data

void* RTI_RoutingServiceProcessorPlugin::processor_plugin_data

A placeholder for Processor implementors to keep a pointer to data that may be needed by the implementation.

6.7 RTI_RoutingServiceProperties Struct Reference

Set of configuration properties.

#include <routingservice_infrastructure.h>

Data Fields

- struct **RTI_RoutingServiceNameValue** * **properties**

  Array of configuration properties.
- int **count**

  Number of properties in the array.
- DDS_Boolean **string_values**

  *A non-zero value indicates that all the values of the properties are strings (char *)*

6.7.1 Detailed Description

Set of configuration properties.

Configuration properties for adapters and transformations.

6.7.2 Field Documentation

6.7.2.1 properties

```
struct RTI_RoutingServiceNameValue* RTI_RoutingServiceProperties::properties
```

Array of configuration properties.

6.7.2.2 count

```
int RTI_RoutingServiceProperties::count
```

Number of properties in the array.

6.7.2.3 string_values

```
DDS_Boolean RTI_RoutingServiceProperties::string_values
```

A non-zero value indicates that all the values of the properties are strings (char *)

6.8 RTI_RoutingServiceProperty Struct Reference

Configuration of RTI Routing Service.

```
#include <routingservice_service.h>
```

Data Fields

- char * **cfg_file**

  Path to an RTI Routing Service configuration file.
- const char ** **cfg_strings**

  XML configuration represented as strings.
- int **cfg_strings_count**

  *Size of the array **cfg_strings** (p. 92).*
- char * **service_name**

  The name of the Routing Service instance to run.
- char * **application_name**

  Assigns a name to the execution of the RTI Routing Service.
- DDS_Boolean **enforce_xsd_validation**

  Controls whether the service applies XSD validation to the loaded configuration.
- int **service_verbosity**

  The verbosity of the service.
- int **dds_verbosity**

  The verbosity of RTI Connext core libraries.
- int **domain_id_base**

  Value that is added to the domain IDs of the domain routes in the XML configuration.
- char * **plugin_search_path**

  This path is used to look for plug-in libraries.
- DDS_Boolean **dont_start_service**

  *Set this to true to if you do not want RTI Routing Service enabled when **RTI_RoutingService_start** (p. 33) is called.*
- DDS_Boolean **enable_administration**

  Set this to true to enable remote administration or false to disable it.
- int **administration_domain_id**

  *If **enable_administration** (p. 94) is true, this is the domain ID to use for remote administration.*
- DDS_Boolean **enable_monitoring**

  Set it to true to enable remote monitoring or false to disable it.
- int **monitoring_domain_id**

  *If **enable_monitoring** (p. 95) is true, this is the domain ID to use for remote monitoring.*
- char * **internal_clock**

  Clock value that is set as Routing Service internal clock. The clock must be defined as a comma-delimited list of clocks, in the order of preference. Valid clock names are "realtime," "system," and "monotonic." |br|.
- DDS_Boolean **skip_default_files**

  Set it to true to avoid loading the standard files usually loaded by RTI Routing Service.
- DDS_Boolean **identify_execution**

  Set this to true to append the host name and process ID to the RTI Routing Service execution name.
- struct **RTI_RoutingServiceTransportConfig** **registered_transports** [8]

Transports to be loaded by RTI Connext.

- int **registered_transports_count**

  *Number of transports configured in **registered_transports** (p. 96).*

- char * **license_file_name**

  Path to an RTI Routing Service license file.

- struct **RTI_RoutingServiceProperties** **user_environment**

  Dictionary of user variables. The dictionary provides a parallel way to expand XML configuration variables in the form , when they are not defined in the environment.

6.8.1 Detailed Description

Configuration of RTI Routing Service.

This structure must be initialized with **RTI_RoutingServiceProperty_INITIALIZER** (p. 32).

6.8.2 Field Documentation

6.8.2.1 cfg_file

```
char* RTI_RoutingServiceProperty::cfg_file
```

Path to an RTI Routing Service configuration file.

If not `NULL`, this file is loaded; otherwise, if `cfg_strings` is not `NULL`, that XML code is loaded.

[default] NULL.

6.8.2.2 cfg_strings

```
const char** RTI_RoutingServiceProperty::cfg_strings
```

XML configuration represented as strings.

An array of strings that altogether make up an XML document to configure RTI Routing Service. This parameter is used only if **cfg_file** (p. 92) is `NULL`.

For example:

```
int MY_ROUTING_SERVICE_CFG_SIZE = 3;
const char * MY_ROUTING_SERVICE_CFG[MY_ROUTING_SERVICE_CFG_SIZE] =
                         {"<dds><routing_service>",
                          "<domain_route><participant><domai",
                          "n_id>0...</dds>"};
property.cfg_strings = MY_ROUTING_SERVICE_CFG;
property.cfg_strings_count = MY_ROUTING_SERVICE_CFG_SIZE;
```

The reason for using an array instead of one single string is to get around the limited size of literal strings in C. In general, if you create the XML string dynamically using one single string in the array, setting **cfg_strings_count** (p. 92) to 1 is enough:

```
property.cfg_strings = malloc(sizeof(char *));
property.cfg_strings[0] = "<dds><routing_service>...</dds>";
property.cfg_strings_count = 1;
```

If your target system doesn't support a file system, you can use XML strings to configure the service. To ease this process, a utility is shipped to generate a C string array from a text file. You can find this utility in [RTI Routing Service installation directory]/resource/perl/cStringifyFile.pl

[default] NULL.

6.8.2.3 cfg_strings_count

```
int RTI_RoutingServiceProperty::cfg_strings_count
```

Size of the array **cfg_strings** (p. 92).

[default] 0.

6.8.2.4 service_name

```
char* RTI_RoutingServiceProperty::service_name
```

The name of the Routing Service instance to run.

This is the name used to find the <routing_service> XML tag in the configuration file; the name that will be used to refer to this execution in remote administration and monitoring.

[default] NULL (use **RTI_RoutingService** (p. 79))

6.8.2.5 application_name

```
char* RTI_RoutingServiceProperty::application_name
```

Assigns a name to the execution of the RTI Routing Service.

Remote commands and status information will refer to the routing service using this name. In addition, the name of DomainParticipants created by RTI Routing Service will be based on this name.

[default] service_name if this value is different than NULL. Otherwise, "RTI_Routing_Service".

6.8.2.6 enforce_xsd_validation

```
DDS_Boolean RTI_RoutingServiceProperty::enforce_xsd_validation
```

Controls whether the service applies XSD validation to the loaded configuration.

[default] DDS_BOOLEAN_TRUE

6.8.2.7 service_verbosity

```
int RTI_RoutingServiceProperty::service_verbosity
```

The verbosity of the service.

Values:

- **RTI_ROUTING_SERVICE_LOG_VERBOSITY_SILENT** (p. 39)
- **RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS** (p. 38)
- **RTI_ROUTING_SERVICE_LOG_VERBOSITY_WARNINGS** (p. 38)
- **RTI_ROUTING_SERVICE_LOG_VERBOSITY_INFO** (p. 38)

[default] RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS

6.8.2.8 dds_verbosity

```
int RTI_RoutingServiceProperty::dds_verbosity
```

The verbosity of RTI Connext core libraries.

Values:

- **RTI_ROUTING_SERVICE_LOG_VERBOSITY_SILENT** (p. 39)
- **RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS** (p. 38)
- **RTI_ROUTING_SERVICE_LOG_VERBOSITY_WARNINGS** (p. 38)
- **RTI_ROUTING_SERVICE_LOG_VERBOSITY_INFO** (p. 38)

[default] RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS

6.8.2.9 domain_id_base

```
int RTI_RoutingServiceProperty::domain_id_base
```

Value that is added to the domain IDs of the domain routes in the XML configuration.

By using this, an XML file can use relative domain IDs.

[default] 0

6.8.2.10 plugin_search_path

```
char* RTI_RoutingServiceProperty::plugin_search_path
```

This path is used to look for plug-in libraries.

If the plug-in class libraries specified in the XML file don't contain a full path, RTI Routing Service looks for them in this directory. If not present, it will rely on the system library path.

[default] NULL (current directory)

6.8.2.11 dont_start_service

```
DDS_Boolean RTI_RoutingServiceProperty::dont_start_service
```

Set this to true to if you do not want RTI Routing Service enabled when **RTI_RoutingService_start** (p. 33) is called.

RTI Routing Service can be enabled afterwards through remote administration.

[default] DDS_BOOLEAN_FALSE

6.8.2.12 enable_administration

```
DDS_Boolean RTI_RoutingServiceProperty::enable_administration
```

Set this to true to enable remote administration or false to disable it.

[default] DDS_BOOLEAN_FALSE

6.8.2.13 administration_domain_id

```
int RTI_RoutingServiceProperty::administration_domain_id
```

If **enable_administration** (p. 94) is true, this is the domain ID to use for remote administration.

Takes precedence over the XML configuration. If **enable_administration** (p. 94) is false, this value is not used even if remote administration is enabled in the XML configuration.

[default] 0

6.8.2.14 enable_monitoring

```
DDS_Boolean RTI_RoutingServiceProperty::enable_monitoring
```

Set it to true to enable remote monitoring or false to disable it.

[default] DDS_BOOLEAN_FALSE

6.8.2.15 monitoring_domain_id

```
int RTI_RoutingServiceProperty::monitoring_domain_id
```

If **enable_monitoring** (p. 95) is true, this is the domain ID to use for remote monitoring.

Takes precedence over the XML configuration. If **enable_monitoring** (p. 95) is false, this value is not used, even if remote monitoring is enabled in the XML configuration.

[default] 0

6.8.2.16 internal_clock

```
char* RTI_RoutingServiceProperty::internal_clock
```

Clock value that is set as Routing Service internal clock. The clock must be defined as a comma-delimited list of clocks, in the order of preference. Valid clock names are "realtime," "system," and "monotonic." |br|.

By using this, Routing Service can use different clocks.

[default] realtime

6.8.2.17 skip_default_files

```
DDS_Boolean RTI_RoutingServiceProperty::skip_default_files
```

Set it to true to avoid loading the standard files usually loaded by RTI Routing Service.

Only the configuration in **cfg_file** (p. 92) or **cfg_strings** (p. 92) will be loaded.

[default] DDS_BOOLEAN_FALSE

6.8.2.18 identify_execution

```
DDS_Boolean RTI_RoutingServiceProperty::identify_execution
```

Set this to true to append the host name and process ID to the RTI Routing Service execution name.

Used to get unique names for remote administration and monitoring.

[default] DDS_BOOLEAN_FALSE

6.8.2.19 registered_transports

```
struct RTI_RoutingServiceTransportConfig RTI_RoutingServiceProperty::registered_transports[8]
```

Transports to be loaded by RTI Connext.

See also

registered_transports_count (p. 96)

Precondition

Maximum 8 transports

6.8.2.20 registered_transports_count

```
int RTI_RoutingServiceProperty::registered_transports_count
```

Number of transports configured in **registered_transports** (p. 96).

Precondition

Minimum 0 (no custom transport to load), maximum 8.

[default] 0

6.8.2.21 license_file_name

```
char* RTI_RoutingServiceProperty::license_file_name
```

Path to an RTI Routing Service license file.

If not NULL, this file is checked for a valid license; otherwise, default location will be used. This parameter is only used if your installation requires a license file.

[default] NULL.

6.8.2.22 user_environment

```
struct RTI_RoutingServiceProperties RTI_RoutingServiceProperty::user_environment
```

Dictionary of user variables. The dictionary provides a parallel way to expand XML configuration variables in the form , when they are not defined in the environment.

[default] empty

6.9 RTI_RoutingServiceStreamInfo Struct Reference

Stream information.

```
#include <routingservice_infrastructure.h>
```

Data Fields

- char * **stream_name**

  The stream name.
- struct **RTI_RoutingServiceTypeInfo** **type_info**

  The type information associated with the stream.
- DDS_Boolean **disposed**

  Indicates whether the stream is a newly discovered stream or a disposed stream that no longer exists.

6.9.1 Detailed Description

Stream information.

A stream in RTI Routing Service is a typed data channel to consume/produce data in one data domain.

6.9.2 Field Documentation

6.9.2.1 stream_name

```
char* RTI_RoutingServiceStreamInfo::stream_name
```

The stream name.

6.9.2.2 type_info

```
struct RTI_RoutingServiceTypeInfo RTI_RoutingServiceStreamInfo::type_info
```

The type information associated with the stream.

This field is invalid for disposed streams.

6.9.2.3 disposed

```
DDS_Boolean RTI_RoutingServiceStreamInfo::disposed
```

Indicates whether the stream is a newly discovered stream or a disposed stream that no longer exists.

6.10 RTI_RoutingServiceStreamReaderListener Struct Reference

StreamReader listener used to notify Routing Service that new data is available.

```
#include <routingservice_adapter.h>
```

Data Fields

- void * **listener_data**
  A place for RTI Routing Service to keep a pointer to data that is needed on the on_data_available notification.
- **RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback on_data_available**
  Prototype of the callback used to notify of new samples.

6.10.1 Detailed Description

StreamReader listener used to notify Routing Service that new data is available.

RTI Routing Service uses the session threads (there is one per <session> tag) to read data from StreamReaders.

Each session thread will block waiting for new data using a WaitSet. When a StreamReader receives new data, it will use the StreamReaderListener's **RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback** (p. 52) callback operation to wake up the associated session thread. After that, the session thread will invoke the StreamReader's **RTI_RoutingServiceStreamReader_ReadFcn** (p. 52) operation to get the new data.

The following figure describes how the session thread reads samples from a StreamReader.

6.10.2 Field Documentation

6.10.2.1 listener_data

```
void* RTI_RoutingServiceStreamReaderListener::listener_data
```

A place for RTI Routing Service to keep a pointer to data that is needed on the on_data_available notification.

The value of this field is assigned by RTI Routing Service. The adapter implementor must make it available as a parameter in the **RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback** (p. 52) call.

6.11 RTI_RoutingServiceStringSeq Struct Reference

Definition of a String sequence.

```
#include <routingservice_infrastructure.h>
```

Data Fields

- char ** **element_array**
  Array of elements.
- int **element_count**
  Number of elements in the array.
- int **element_count_max**
  maximum capacity of the array.

6.11.1 Detailed Description

Definition of a String sequence.

6.11.2 Field Documentation

6.11.2.1 element_array

```
char** RTI_RoutingServiceStringSeq::element_array
```

Array of elements.

6.11.2.2 element_count

```
int RTI_RoutingServiceStringSeq::element_count
```

Number of elements in the array.

6.11.2.3 element_count_max

```
int RTI_RoutingServiceStringSeq::element_count_max
```

maximum capacity of the array.

6.12 RTI_RoutingServiceTransformationPlugin Struct Reference

Transformation plugin.

```
#include <routingservice_transformation.h>
```

Data Fields

- struct **RTI_RoutingServiceVersion plugin_version**
  The version of this transformation plugin.
- **RTI_RoutingServiceTransformationPlugin_DeleteFcn transformation_plugin_delete**
  Handles the deletion of the transformation plugin.
- **RTI_RoutingServiceTransformationPlugin_CreateTransformationFcn transformation_plugin_create_↩ transformation**
  Handles the creation of transformations.
- **RTI_RoutingServiceTransformationPlugin_DeleteTransformationFcn transformation_plugin_delete_↩ transformation**
  Handles the deletion of transformations.
- **RTI_RoutingServiceTransformation_TransformFcn transformation_transform**
  Handles the transformation of samples.
- **RTI_RoutingServiceTransformation_ReturnLoanFcn transformation_return_loan**
  Handles the return of the loan taken on the transformed samples.
- **RTI_RoutingServiceTransformation_UpdateFcn transformation_update**
  Handles the update of the transformation configuration.
- void * **user_object**
  A place for transformation implementors to keep a pointer to data that may be needed by the implementation.

6.12.1 Detailed Description

Transformation plugin.

This structure contains the plugin implementation as a set of function pointers.

6.12.2 Field Documentation

6.12.2.1 plugin_version

struct **RTI_RoutingServiceVersion** RTI_RoutingServiceTransformationPlugin::plugin_version

The version of this transformation plugin.

6.12.2.2 transformation_plugin_delete

RTI_RoutingServiceTransformationPlugin_DeleteFcn RTI_RoutingServiceTransformationPlugin::transformation_plugin_delete

Handles the deletion of the transformation plugin.

6.12.2.3 transformation_plugin_create_transformation

RTI_RoutingServiceTransformationPlugin_CreateTransformationFcn RTI_RoutingServiceTransformationPlugin::transformation_plugin_create_transformation

Handles the creation of transformations.

6.12.2.4 transformation_plugin_delete_transformation

RTI_RoutingServiceTransformationPlugin_DeleteTransformationFcn RTI_RoutingServiceTransformationPlugin::transformation_plugin_delete_transformation

Handles the deletion of transformations.

6.12.2.5 transformation_transform

RTI_RoutingServiceTransformation_TransformFcn RTI_RoutingServiceTransformationPlugin::transformation_transform

Handles the transformation of samples.

6.12.2.6 transformation_return_loan

RTI_RoutingServiceTransformation_ReturnLoanFcn RTI_RoutingServiceTransformationPlugin::transformation↩
_return_loan

Handles the return of the loan taken on the transformed samples.

6.12.2.7 transformation_update

RTI_RoutingServiceTransformation_UpdateFcn RTI_RoutingServiceTransformationPlugin::transformation↩
_update

Handles the update of the transformation configuration.

6.12.2.8 user_object

void* RTI_RoutingServiceTransformationPlugin::user_object

A place for transformation implementors to keep a pointer to data that may be needed by the implementation.

6.13 RTI_RoutingServiceTransportConfig Struct Reference

Association between a transport alias and its create function pointer.

#include <routingservice_service.h>

Data Fields

- char * **alias**

  An alias defined in the XML configuration to refer to a transport.
- NDDS_Transport_create_plugin **create_function**

  Pointer to the function to load the transport.

6.13.1 Detailed Description

Association between a transport alias and its create function pointer.

Allows setting the entry point to load a statically linked transport and refer to it in the XML configuration through the participant_qos transport configuration.

If a transport is configured in the XML configuration and its alias matches the one in this structure, it will be loaded by RTI Connext using the specified function pointer.

6.13.2 Field Documentation

6.13.2.1 alias

```
char* RTI_RoutingServiceTransportConfig::alias
```

An alias defined in the XML configuration to refer to a transport.

6.13.2.2 create_function

```
NDDS_Transport_create_plugin RTI_RoutingServiceTransportConfig::create_function
```

Pointer to the function to load the transport.

6.14 RTI_RoutingServiceTypeInfo Struct Reference

Type information.

```
#include <routingservice_infrastructure.h>
```

Data Fields

- char * **type_name**
  The registered type name.
- **RTI_RoutingServiceTypeRepresentationKind type_representation_kind**
  The representation kind.
- **RTI_RoutingServiceTypeRepresentation type_representation**
  The type representation.

6.14.1 Detailed Description

Type information.

6.14.2 Field Documentation

6.14.2.1 type_name

```
char* RTI_RoutingServiceTypeInfo::type_name
```

The registered type name.

6.14.2.2 type_representation_kind

```
RTI_RoutingServiceTypeRepresentationKind RTI_RoutingServiceTypeInfo::type_representation_kind
```

The representation kind.

See also

Standard Type Representation Kinds (p. 75)

6.14.2.3 type_representation

```
RTI_RoutingServiceTypeRepresentation RTI_RoutingServiceTypeInfo::type_representation
```

The type representation.

6.15 RTI_RoutingServiceVersion Struct Reference

Represents the version of a plugin or RTI Routing Service itself.

```
#include <routingservice_infrastructure.h>
```

Data Fields

- int **major**
  Major version number.
- int **minor**
  Minor version number.
- int **release**
  Release version number.
- int **revision**
  Revision of a release.

6.15.1 Detailed Description

Represents the version of a plugin or RTI Routing Service itself.

6.15.2 Field Documentation

6.15.2.1 major

```
int RTI_RoutingServiceVersion::major
```

Major version number.

6.15.2.2 minor

```
int RTI_RoutingServiceVersion::minor
```

Minor version number.

6.15.2.3 release

```
int RTI_RoutingServiceVersion::release
```

Release version number.

6.15.2.4 revision

```
int RTI_RoutingServiceVersion::revision
```

Revision of a release.

Chapter 7

File Documentation

7.1 routingservice_infrastructure.h File Reference

RTI Routing Service Infrastructure.

```
#include "routingservice/routingservice_dll.h"
#include "dds_c/dds_c_infrastructure.h"
```

Data Structures

- struct **RTI_RoutingServiceNameValue**

  Configuration property.
- struct **RTI_RoutingServiceProperties**

  Set of configuration properties.
- struct **RTI_RoutingServiceVersion**

  Represents the version of a plugin or RTI Routing Service itself.
- struct **RTI_RoutingServiceTypeInfo**

  Type information.
- struct **RTI_RoutingServiceStringSeq**

  Definition of a String sequence.
- struct **RTI_RoutingServiceStreamInfo**

  Stream information.

Macros

- #define **RTI_USER_DLL_EXPORT**

  Utility macro that can be used to export symbols in a shared library on Windows platform. For non-windows, the definion of this macro is empty.
- #define **RTI_UNUSED_PARAMETER**(x) (void)(x)

  This can be used by C client and example code to avoid unused parameter warnings in the compiler. This is not strictly necessary in C++, where just removing the parameter name should yield the same result.
- #define **RTI_ROUTING_SERVICE_ERROR**

  Generic, unspecified error.
- #define **RTI_ROUTING_SERVICE_FATAL_ERROR**

  Fatal error. It will cause the routing service to shut down.
- #define **RTI_ROUTING_SERVICE_ERROR_MAX_LENGTH** 1024

  Maximum length of an error message.
- #define **RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_DYNAMIC_TYPE**

  Dynamic type representation.
- #define **RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_XML**

  ***[Not supported]** XML type representation.*
- #define **RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_JAVA_OBJECT**

  ***[Not supported]** Java object type representation.*
- #define **RTI_ROUTING_SERVICE_DATA_REPRESENTATION_DYNAMIC_DATA**

  Dynamic data representation.
- #define **RTI_ROUTING_SERVICE_DATA_REPRESENTATION_XML**

  ***[Not supported]** XML data representation.*
- #define **RTI_ROUTING_SERVICE_DATA_REPRESENTATION_JAVA_OBJECT**

  ***[Not supported]** Java object data representation.*
- #define **RTI_ROUTING_SERVICE_APP_NAME_PROPERTY_NAME** RTI_ROUTING_SERVICE_PROPERTY↩ _PREFIX".app_name"

  Name of the property that provides the RTI Routing Service given application name.
- #define **RTI_ROUTING_SERVICE_GROUP_PROPERTY_NAME** RTI_ROUTING_SERVICE_PROPERTY_↩ PREFIX".group_name"
- #define **RTI_ROUTING_SERVICE_VERSION_PROPERTY_NAME** RTI_ROUTING_SERVICE_PROPERTY_↩ PREFIX".version"

  Name of the property that provides the RTI Routing Service version as string.
- #define **RTI_ROUTING_SERVICE_VERBOSITY_PROPERTY_NAME** RTI_ROUTING_SERVICE_PROPERTY↩ _PREFIX".verbosity"

  Name of the property that provides verbosity in use by RTI Routing Service.
- #define **RTI_ROUTING_SERVICE_ENTITY_RESOURCE_NAME_PROPERTY_NAME** RTI_ROUTING_↩ SERVICE_PROPERTY_PREFIX".entity.resource_name"

  Name of the property that provides the resource name of the entity that owns the adapter entity.

Typedefs

- typedef struct RTI_RoutingServiceEnvironmentImpl **RTI_RoutingServiceEnvironment**

  The environment permits the return of error information in the RTI Routing Service API and information retreival (version and verbosity).
- typedef int **RTI_RoutingServiceTypeRepresentationKind**

  Type representation kind.

- typedef void * **RTI_RoutingServiceTypeRepresentation**

  Type representation.
- typedef int **RTI_RoutingServiceDataRepresentationKind**

  Data representation kind.
- typedef void * **RTI_RoutingServiceSample**

  Stream sample.
- typedef void * **RTI_RoutingServiceSampleInfo**

  Stream sample info. In DDS, this is a DDS_SampleInfo object.

Enumerations

- enum **RTI_RoutingServiceVerbosity** {
  RTI_ROUTING_SERVICE_VERBOSITY_NONE = 0 ,
  RTI_ROUTING_SERVICE_VERBOSITY_EXCEPTION ,
  RTI_ROUTING_SERVICE_VERBOSITY_WARN ,
  RTI_ROUTING_SERVICE_VERBOSITY_INFO ,
  RTI_ROUTING_SERVICE_VERBOSITY_DEBUG }

  Verbosity used by Routing Service.

Functions

- const char * **RTI_RoutingServiceProperties_lookup_property** (const struct **RTI_RoutingServiceProperties** *self, const char *name)

  Searches for a property given its name.
- void **RTI_RoutingServiceEnvironment_set_error_w_params** (**RTI_RoutingServiceEnvironment** *self, DDS_Boolean overwrite, int error_code, int native_error_code, const char *error_format,...)

  Assigns an error into the environment.
- void **RTI_RoutingServiceEnvironment_set_error** (**RTI_RoutingServiceEnvironment** *self, const char *error_format,...)

  Assigns an error into the environment.
- void **RTI_RoutingServiceEnvironment_fatal_error** (**RTI_RoutingServiceEnvironment** *self, const char *error_format,...)

  Assigns a fatal error into the environment.
- void **RTI_RoutingServiceEnvironment_clear_error** (**RTI_RoutingServiceEnvironment** *self)

  Clears an error (if any) set in this environment.
- **RTI_RoutingServiceVerbosity** **RTI_RoutingServiceEnvironment_get_verbosity** (const **RTI_Routing↩ ServiceEnvironment** *self)

  Retrieves the verbosity that Routing Service is using.
- DDS_Boolean **RTI_RoutingServiceEnvironment_error_occurred** (const **RTI_RoutingServiceEnvironment** *self)

  Checks whether an error has been set in this environment.
- const char * **RTI_RoutingServiceEnvironment_get_error_message** (const **RTI_RoutingService↩ Environment** *self)

  Returns the error message this environment contains.
- struct **RTI_RoutingServiceStreamInfo** * **RTI_RoutingServiceStreamInfo_new_discovered** (const char *stream_name, const char *registered_type_name, **RTI_RoutingServiceTypeRepresentationKind** type_↩ representation_kind, **RTI_RoutingServiceTypeRepresentation** type_representation)

Creates a stream info for a newly discovered stream.

- struct **RTI_RoutingServiceStreamInfo** * **RTI_RoutingServiceStreamInfo_new_disposed** (const char *stream_name)

  Creates a stream info for a disposed stream. Disposed streams are no longer available in a data domain.

- void **RTI_RoutingServiceStreamInfo_delete** (struct **RTI_RoutingServiceStreamInfo** *self)

  Destroys a stream info.

7.1.1 Detailed Description

RTI Routing Service Infrastructure.

7.2 routingservice_infrastructure.h

Go to the documentation of this file.

```
1 /*
2  * (c) Copyright, Real-Time Innovations, 2002-2024.
3  * All rights reserved.
4  *
5  * No duplications, whole or partial, manual or electronic, may be made
6  * without express written permission.  Any such copies, or
7  * revisions thereof, must display this notice unaltered.
8  * This code contains trade secrets of Real-Time Innovations, Inc.
9  */
10
11 #ifndef routingservice_infrastructure_h
12 #define routingservice_infrastructure_h
13
14 #include "routingservice/routingservice_dll.h"
15 #include "dds_c/dds_c_infrastructure.h"
16
17 #ifdef __cplusplus
18     extern "C" {
19 #endif
20
21 /*e \file
22   @brief RTI Routing Service Infrastructure
23 */
24
25 /*e
26   @ingroup RTI_RoutingServiceInfrastructureModule
27
28   @brief Utility macro that can be used to export symbols in a shared library
29   on Windows platform.
30   For non-windows, the definion of this macro is empty.
31 */
32
33 #if (defined(RTI_WIN32) || defined(RTI_INTIME))
34   #undef RTI_USER_DLL_EXPORT
35   #define RTI_USER_DLL_EXPORT __declspec(dllexport)
36 #elif defined(RTI_VISIBILITY_HIDDEN)
37   #undef RTI_USER_DLL_EXPORT
38   #define RTI_USER_DLL_EXPORT __attribute__((visibility("default")))
39 #else
40   #define RTI_USER_DLL_EXPORT
41 #endif
42
43 #define RTI_ROUTING_SERVICE_VERSION {7,7,0,0}
44
45 /*e \ingroup RTI_RoutingServiceInfrastructureModule
46  * @brief This can be used by C client and example code to avoid unused
47  * parameter warnings in the compiler.
48  * This is not strictly necessary in C++, where just removing the parameter name
49  * should yield the same result.
50  */
51 #define RTI_UNUSED_PARAMETER(x) (void)(x)
52
53 /*****************************************************************************/
```

```
54 /* Properties                                                                 */
55 /******************************************************************************/
56
57 /*e \ingroup RTI_RoutingServiceInfrastructureModule
58  *
59  * @brief Configuration property.
60  *
61  * A property is a name/value pair.
62  *
63  */
64 struct RTI_RoutingServiceNameValue {
65     /*e @brief Property name */
66     char * name;
67     /*e @brief Property value */
68     void * value;
69 };
70
71 /*e \ingroup RTI_RoutingServiceInfrastructureModule
72  *
73  * @brief Set of configuration properties.
74  *
75  * Configuration properties for adapters and transformations.
76  */
77 struct RTI_RoutingServiceProperties {
78     /*e @brief Array of configuration properties. */
79     struct RTI_RoutingServiceNameValue * properties;
80     /*e @brief Number of properties in the array. */
81     int count;
82     /*e @brief A non-zero value indicates that all the values of the
83         properties are strings (char *) */
84     DDS_Boolean string_values;
85 };
86
87 /*e \ingroup RTI_RoutingServiceInfrastructureModule
88  *
89  * @brief Searches for a property given its name.
90  *
91  * @param_self
92  * @param name \rs_st_in Property name. Cannot be NULL.
93  *
94  * @return On success, the function returns the value of the first property
95  * with the given name. Otherwise, the function returns NULL.
96  */
97 extern ROUTERDllExport
98 const char * RTI_RoutingServiceProperties_lookup_property(
99     const struct RTI_RoutingServiceProperties * self,
100     const char * name);
101
102 extern ROUTERDllExport
103 const char * RTI_RoutingServiceProperties_lookup_property_with_prefix(
104         const struct RTI_RoutingServiceProperties * self,
105         const char *prefix,
106         const char *name);
107
108 extern ROUTERDllExport
109 DDS_Boolean RTI_RoutingServiceProperties_add(
110         struct RTI_RoutingServiceProperties *self,
111         const char *name,
112         const char *value);
113
114 extern ROUTERDllExport
115 DDS_Boolean RTI_RoutingServiceProperties_assert(
116         struct RTI_RoutingServiceProperties *self,
117         const char *name,
118         const char *value);
119
120 extern ROUTERDllExport
121 DDS_Boolean RTI_RoutingServiceProperties_equals(
122         const struct RTI_RoutingServiceProperties *left,
123         const struct RTI_RoutingServiceProperties *right);
124
125 extern ROUTERDllExport
126 const struct RTI_RoutingServiceProperties *
127 RTI_RoutingServiceProperties_copy(
128         struct RTI_RoutingServiceProperties *self,
129         const struct RTI_RoutingServiceProperties *other);
130
131 extern ROUTERDllExport
132 void RTI_RoutingServiceProperties_finalize(
133         struct RTI_RoutingServiceProperties *self);
134
```

```
135 extern ROUTERDllExport
136 DDS_Boolean RTI_RoutingServiceProperties_initialize(
137     struct RTI_RoutingServiceProperties *self);
138 
139 /*****************************************************************************/
140 /* Environment                                                               */
141 /*****************************************************************************/
142 
143 /*e \defgroup RTI_RoutingServiceErrorCodeModule Standard Error Codes
144  * \ingroup RTI_RoutingServiceInfrastructureModule
145  * @brief Standard error codes.
146  */
147 #define RTI_ROUTING_SERVICE_OK 0
148 
149 /*e \ingroup RTI_RoutingServiceErrorCodeModule
150  *
151  * @brief Generic, unspecified error.
152  * @hideinitializer
153  */
154 #define RTI_ROUTING_SERVICE_ERROR 1
155 
156 /*e \ingroup RTI_RoutingServiceErrorCodeModule
157  *
158  * @brief Fatal error. It will cause the routing service to shut down.
159  * @hideinitializer
160  */
161 #define RTI_ROUTING_SERVICE_FATAL_ERROR 2
162 
163 /*e \ingroup RTI_RoutingServiceInfrastructureModule
164  * @brief Maximum length of an error message.
165  *
166  * Error messages longer than this value are truncated.
167  */
168 #define RTI_ROUTING_SERVICE_ERROR_MAX_LENGTH 1024
169 
170 struct RTI_RoutingServiceEnvironmentImpl;
171 
172 /*e \ingroup RTI_RoutingServiceInfrastructureModule
173  *
174  * @brief The environment permits the return of error information in the \product API and
175  *        information retreival (version and verbosity).
176  *
177  * This is the last parameter of each operation.
178  *
179  * \see \ref RTI_RoutingServiceEnvironment_set_error
180  * \see \ref RTI_RoutingServiceEnvironment_get_verbosity
181  *
182  */
183 typedef struct RTI_RoutingServiceEnvironmentImpl RTI_RoutingServiceEnvironment;
184 
185 /*e \ingroup RTI_RoutingServiceInfrastructureModule
186  *
187  * @brief Represents the version of a plugin or \product itself
188  *
189  */
190 struct RTI_RoutingServiceVersion {
191     /*e
192      * @brief Major version number
193      */
194     int major;
195     /*e
196      * @brief Minor version number
197      */
198     int minor;
199     /*e
200      * @brief Release version number
201      */
202     int release;
203     /*e
204      * @brief Revision of a release
205      */
206     int revision;
207 };
208 
209 /*e \ingroup RTI_RoutingServiceInfrastructureModule
210  *
211  * @brief Verbosity used by Routing Service
212  *
213  */
214 typedef enum {
215     /*e
```

```
216      * No logging (-verbosity 0)
217      */
218     RTI_ROUTING_SERVICE_VERBOSITY_NONE = 0,
219     /*e
220      * Exceptions (-verbosity 1)
221      */
222     RTI_ROUTING_SERVICE_VERBOSITY_EXCEPTION,
223     /*e
224      * Warnings (-verbosity 2)
225      */
226     RTI_ROUTING_SERVICE_VERBOSITY_WARN,
227     /*e
228      * Information (-verbosity 3)
229      */
230     RTI_ROUTING_SERVICE_VERBOSITY_INFO,
231     /*e
232      * Debug information (-verbosity 5 and 6)
233      */
234     RTI_ROUTING_SERVICE_VERBOSITY_DEBUG
235 } RTI_RoutingServiceVerbosity;
236
237 /*e \ingroup RTI_RoutingServiceInfrastructureModule
238  *
239  * @brief Assigns an error into the environment.
240  *
241  * Routing Service will consider that a function call failed when an error has been
242  * set into the environment
243  *
244  * @param_self
245  * @param overwrite \rs_st_in If the environment already contains an error, setting
246  * this parameter to DDS_BOOLEAN_TRUE will overwrite it.
247  * @param error_code \rs_st_in One of the \ref RTI_RoutingServiceErrorCodeModule.
248  * @param native_error_code \rs_st_in Native error code (specific to the transformation or
249  * adapter plugin).
250  * @param error_format \rs_st_in String that contains the error text.
251  * It can optionally contain format tags that are substituted by the values
252  * specified in subsequent argument(s). Cannot be NULL.
253  * The format tags are the same tags used by the function printf in the ANSI C standard.
254  *
255  * @see \ref RTI_RoutingServiceEnvironment_set_error
256  */
257 extern ROUTERDllExport void RTI_RoutingServiceEnvironment_set_error_w_params(
258         RTI_RoutingServiceEnvironment *self,
259         DDS_Boolean overwrite,
260         int error_code,
261         int native_error_code,
262         const char *error_format,
263         ...);
264
265 /*e \ingroup RTI_RoutingServiceInfrastructureModule
266  *
267  * @brief Assigns an error into the environment.
268  *
269  * Routing Service will consider that a function call failed when an error has been
270  * set into the environment
271  *
272  * This function sets the error code to \ref RTI_ROUTING_SERVICE_ERROR and the
273  * native error code to 0.
274  *
275  * If the environment already contains an error, the error is not overwritten.
276  *
277  * @param_self
278  * @param error_format \rs_st_in String that contains the error text.
279  * It can optionally contain format tags that are substituted by the values
280  * specified in subsequent argument(s). Cannot be NULL.
281  * The format tags are the same tags used by the function printf in the ANSI C standard.
282  *
283  * @see \ref RTI_RoutingServiceEnvironment_set_error_w_params
284  */
285 extern ROUTERDllExport void RTI_RoutingServiceEnvironment_set_error(
286         RTI_RoutingServiceEnvironment *self,
287         const char *error_format,
288         ...);
289
290 /*e \ingroup RTI_RoutingServiceInfrastructureModule
291  *
292  * @brief Assigns a fatal error into the environment.
293  *
294  * Routing Service will consider that a function call failed when an error has been
295  * set into the environment. Additionally, a fatal error will cause the routing
296  * service to call the shutdown hook.
```

```
297  *
298  * This function sets the error code to \ref RTI_ROUTING_SERVICE_FATAL_ERROR and the
299  * native error code to 0. Any previous error in the environment will be overwritten.
300  *
301  * @param_self
302  * @param error_format \rs_st_in String that contains the error text.
303  * It can optionally contain format tags that are substituted by the values
304  * specified in subsequent argument(s). Cannot be NULL.
305  * The format tags are the same tags used by the function printf in the ANSI C standard.
306  *
307  * @see \ref RTI_RoutingServiceEnvironment_set_error_w_params
308  */
309 extern ROUTERDllExport void RTI_RoutingServiceEnvironment_fatal_error(
310         RTI_RoutingServiceEnvironment *self,
311         const char *error_format,
312         ...);
313
314 /*e \ingroup RTI_RoutingServiceInfrastructureModule
315  *
316  * @brief Clears an error (if any) set in this environment
317  *
318  * @param_self
319  */
320 extern ROUTERDllExport void RTI_RoutingServiceEnvironment_clear_error(
321         RTI_RoutingServiceEnvironment *self);
322
323 /*e \ingroup RTI_RoutingServiceInfrastructureModule
324  *
325  * @brief Retrieves the verbosity that Routing Service is using
326  *
327  * @param_self
328  * @return The verbosity
329  */
330 extern ROUTERDllExport RTI_RoutingServiceVerbosity
331 RTI_RoutingServiceEnvironment_get_verbosity(
332         const RTI_RoutingServiceEnvironment *self);
333
334 /*e \ingroup RTI_RoutingServiceInfrastructureModule
335  *
336  * @brief Checks whether an error has been set in this environment
337  *
338  * @param_self
339  *
340  * @return A value different than zero if true
341  */
342 extern ROUTERDllExport DDS_Boolean RTI_RoutingServiceEnvironment_error_occurred(
343         const RTI_RoutingServiceEnvironment *self);
344
345 /*e \ingroup RTI_RoutingServiceInfrastructureModule
346  *
347  * @brief Returns the error message this environment contains
348  *
349  * @param_self
350  *
351  * @return The error message or NULL if no error is set
352  */
353 extern ROUTERDllExport const char *
354 RTI_RoutingServiceEnvironment_get_error_message(
355         const RTI_RoutingServiceEnvironment *self);
356
357 /*i \ingroup RTI_RoutingServiceInfrastructureModule
358  *
359  * @brief Returns the underlying native code associated to the last error
360  *        this environment contains.
361  *
362  * The value is specific to the plug-in implementation that sets the native
363  * code with RTI_RoutingServiceEnvironment_set_error_w_params.
364  *
365  * @param_self
366  *
367  * @return The native error code.
368  */
369 extern ROUTERDllExport int RTI_RoutingServiceEnvironment_get_native_code(
370         const RTI_RoutingServiceEnvironment *self);
371
372 extern ROUTERDllExport
373 int RTI_RoutingServiceVersion_compare(
374         const struct RTI_RoutingServiceVersion *left,
375         const struct RTI_RoutingServiceVersion *right);
376
377 /*****************************************************************************/
```

```
378 /* Type information                                                           */
379 /*****************************************************************************/
380
381 /*e
382  * \dref_DYNAMIC_TYPE_REPRESENTATION
383  */
384 #define RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_DYNAMIC_TYPE 0
385
386 /*e
387  * \dref_XML_TYPE_REPRESENTATION
388  */
389 #define RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_XML 1
390
391 /*e
392  * \dref_JAVA_OBJECT_TYPE_REPRESENTATION
393  */
394 #define RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_JAVA_OBJECT 2
395
396 #define RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_FIRST_CUSTOM_REPRESENTATION 100
397
398 /*e
399  * \dref_DYNAMIC_DATA_REPRESENTATION
400  */
401 #define RTI_ROUTING_SERVICE_DATA_REPRESENTATION_DYNAMIC_DATA 0
402
403 /*e
404  * \dref_XML_DATA_REPRESENTATION
405  */
406 #define RTI_ROUTING_SERVICE_DATA_REPRESENTATION_XML 1
407
408 /*e
409  * \dref_JAVA_OBJECT_DATA_REPRESENTATION
410  */
411 #define RTI_ROUTING_SERVICE_DATA_REPRESENTATION_JAVA_OBJECT 2
412
413 #define RTI_ROUTING_SERVICE_DATA_REPRESENTATION_FIRST_CUSTOM_REPRESENTATION 100
414
415 /*e \ingroup RTI_RoutingServiceInfrastructureModule
416  * @brief Type representation kind.
417  *
418  * The range [0-100] is reserved for RTI use. Within
419  * that range are some predefined type representations (\ref RTI_RoutingServiceTypeRepresentationModule).
420  *
421  */
422 typedef int RTI_RoutingServiceTypeRepresentationKind;
423
424 /*e \ingroup RTI_RoutingServiceInfrastructureModule
425  *
426  * @brief Type representation.
427  *
428  * If the representation kind is \ref RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_DYNAMIC_TYPE,
429  * the representation will be an \ndds TypeCode.
430  *
431  * If the representation kind is \ref RTI_ROUTING_SERVICE_TYPE_REPRESENTATION_XML,
432  * the representation will be an XML string.
433  */
434 typedef void * RTI_RoutingServiceTypeRepresentation;
435
436 /*e \dref_TypeInfo
437  */
438 struct RTI_RoutingServiceTypeInfo {
439     /*e \dref_TypeInfo_type_name */
440     char * type_name;
441
442     /*i
443      * Whether this type can be optimized (avoid CDR deserialization). This is
444      * assumed to be true but the service may determine otherwise by studying
445      * counterpart type information.
446      */
447     DDS_Boolean can_be_optimized;
448
449     /*e \dref_TypeInfo_representation_kind */
450     RTI_RoutingServiceTypeRepresentationKind type_representation_kind;
451
452     /*e \dref_TypeInfo_type_representation */
453     RTI_RoutingServiceTypeRepresentation type_representation;
454 };
455
456 /*e \ingroup RTI_RoutingServiceInfrastructureModule
457  * @brief Data representation kind.
458  *
```

```
459  * The range [0-100] is reserved for RTI use. Within
460  * that range are some predefined data representations (\ref RTI_RoutingServiceDataRepresentationModule).
461  *
462  */
463 typedef int RTI_RoutingServiceDataRepresentationKind;
464
465 /*************************************************************************/
466 /* Sample and sample information                                         */
467 /*************************************************************************/
468
469 /*e \ingroup RTI_RoutingServiceInfrastructureModule
470  *
471  * @brief Stream sample.
472  *
473  * Samples are data messages generated by adapters and transformations.
474  *
475  * If the representation kind is \ref RTI_ROUTING_SERVICE_DATA_REPRESENTATION_DYNAMIC_DATA,
476  * the sample will be a DynamicData object.
477  *
478  * If the representation kind is \ref RTI_ROUTING_SERVICE_DATA_REPRESENTATION_XML,
479  * the sample will be an XML string.
480  *
481  */
482 typedef void * RTI_RoutingServiceSample;
483
484 /*e
485  * \ingroup RTI_RoutingServiceInfrastructureModule
486  * @brief Stream sample info.
487  * In DDS, this is a DDS_SampleInfo object.
488  */
489 typedef void * RTI_RoutingServiceSampleInfo;
490
491 /*e
492  * \ingroup RTI_RoutingServiceInfrastructureModule
493  * @brief Definition of a String sequence
494  */
495 struct RTI_RoutingServiceStringSeq {
496     /*e @brief Array of elements. */
497     char **element_array;
498     /*e @brief Number of elements in the array. */
499     int element_count;
500     /*e @brief maximum capacity of the array. */
501     int element_count_max;
502 };
503
504
505 #define RTI_RoutingServiceStringSeq_INITIALIZER \
506     {NULL, 0, 0}
507
508 extern ROUTERDllExport
509 struct RTI_RoutingServiceStringSeq *
510 RTI_RoutingServiceStringSeq_copy(
511         struct RTI_RoutingServiceStringSeq *self,
512         struct RTI_RoutingServiceStringSeq *other);
513
514 /*************************************************************************/
515 /* Stream information                                                    */
516 /*************************************************************************/
517
518 /*e
519  * \dref_StreamInfo
520  */
521 struct RTI_RoutingServiceStreamInfo {
522     /*e
523      * \dref_StreamInfo_stream_name
524      */
525     char *stream_name;
526     /*e
527      * \dref_StreamInfo_type_info
528      */
529     struct RTI_RoutingServiceTypeInfo type_info;
530     /*i
531      *
532      * @brief Internal use only
533      */
534     struct RTI_RoutingServiceStringSeq partition;
535     /*e
536      * \dref_StreamInfo_disposed
537      */
538     DDS_Boolean disposed;
539 };
```

```
540
541
542 #define RTI_RoutingServiceStreamInfo_INITIALIZER {\
543     NULL, \
544     {NULL, DDS_BOOLEAN_TRUE, 0, NULL},\
545     RTI_RoutingServiceStringSeq_INITIALIZER, \
546     DDS_BOOLEAN_FALSE \
547 }
548
549 extern ROUTERDllExport void RTI_RoutingServiceStreamInfo_finalize(
550         struct RTI_RoutingServiceStreamInfo *self);
551
552 extern ROUTERDllExport DDS_Boolean RTI_RoutingServiceStreamInfo_initialize(
553         struct RTI_RoutingServiceStreamInfo *self,
554         DDS_Boolean is_disposed_stream,
555         const char *stream_name,
556         const char *registered_type_name,
557         int type_dfinition_format,
558         RTI_RoutingServiceTypeRepresentation type_definition);
559
560 /*e
561  * \ingroup RTI_RoutingServiceInfrastructureModule
562  *
563  * @brief Creates a stream info for a newly discovered stream.
564  *
565  * @param stream_name \rs_st_in Stream name. Cannot be NULL.
566  * @param registered_type_name \rs_st_in Type name. Cannot be NULL.
567  * @param type_representation_kind \rs_st_in Type representation kind.
568  * @param type_representation \rs_st_in Type representation. Cannot be NULL.
569  *
570  * @return Newly created stream info, or NULL if there is an error.
571  */
572 extern ROUTERDllExport struct RTI_RoutingServiceStreamInfo *
573 RTI_RoutingServiceStreamInfo_new_discovered(
574         const char *stream_name,
575         const char *registered_type_name,
576         RTI_RoutingServiceTypeRepresentationKind type_representation_kind,
577         RTI_RoutingServiceTypeRepresentation type_representation);
578
579 /*e
580  * \ingroup RTI_RoutingServiceInfrastructureModule
581  * @brief Creates a stream info for a disposed stream.
582  * Disposed streams are no longer available in a data domain.
583  * @param stream_name \rs_st_in Stream name. Cannot be NULL.
584  * @return Newly created stream info, or NULL if there is an error.
585  */
586 extern ROUTERDllExport struct RTI_RoutingServiceStreamInfo *
587 RTI_RoutingServiceStreamInfo_new_disposed(const char *stream_name);
588
589 /*e \ingroup RTI_RoutingServiceInfrastructureModule
590  *
591  * @brief Destroys a stream info.
592  *
593  * @param_self
594  */
595 extern ROUTERDllExport
596 void RTI_RoutingServiceStreamInfo_delete(
597     struct RTI_RoutingServiceStreamInfo * self);
598
599
600 #define RTI_ROUTING_SERVICE_PROPERTY_PREFIX \
601     "rti.routing_service"
602
603 /*e \ingroup RTI_RoutingServiceInfrastructureModule
604  *
605  * @brief Name of the property that provides the @product given application name
606  *
607  */
608 #define  RTI_ROUTING_SERVICE_APP_NAME_PROPERTY_NAME \
609         RTI_ROUTING_SERVICE_PROPERTY_PREFIX".app_name"
610
611 /*e \ingroup RTI_RoutingServiceInfrastructureModule
612  *
613  * Name of the property that provides the configured group name.
614  *
615  */
616 #define  RTI_ROUTING_SERVICE_GROUP_PROPERTY_NAME \
617         RTI_ROUTING_SERVICE_PROPERTY_PREFIX".group_name"
618
619 /*e \ingroup RTI_RoutingServiceInfrastructureModule
620  *
```

```
621  * @brief Name of the property that provides the @product version as string.
622  *
623  */
624 #define  RTI_ROUTING_SERVICE_VERSION_PROPERTY_NAME \
625         RTI_ROUTING_SERVICE_PROPERTY_PREFIX".version"
626
627 /*e \ingroup RTI_RoutingServiceInfrastructureModule
628  *
629  * @brief  Name of the property that provides verbosity in use by @product.
630  *
631  * It can take on of the following stings:
632  *      - NONE
633  *      - EXCEPTION
634  *      - WARN
635  *      - INFO
636  *      - DEBUG.
637  *
638  */
639 #define  RTI_ROUTING_SERVICE_VERBOSITY_PROPERTY_NAME \
640         RTI_ROUTING_SERVICE_PROPERTY_PREFIX".verbosity"
641
642 /*e \ingroup RTI_RoutingServiceInfrastructureModule
643  *
644  * @brief Name of the property that provides the resource name of the entity
645  *        that owns the adapter entity.
646  *
647  */
648 #define  RTI_ROUTING_SERVICE_ENTITY_RESOURCE_NAME_PROPERTY_NAME \
649         RTI_ROUTING_SERVICE_PROPERTY_PREFIX".entity.resource_name"
650
651 #ifdef __cplusplus
652 }       /* extern "C" */
653 #endif
654
655 #endif /* routingservice_infrastructure_h */
```

7.3 routingservice_adapter.h File Reference

RTI Routing Service C Adapter API.

```
#include "routingservice/routingservice_infrastructure.h"
```

Data Structures

- struct **RTI_RoutingServiceStreamReaderListener**

  StreamReader listener used to notify Routing Service that new data is available.
- struct **RTI_RoutingServiceAdapterPlugin**

  Adapter plugin.

Macros

- #define **RTI_RoutingServiceAdapterPlugin_initialize**(adapter)

  Initializes the adapter plugin structure.

Typedefs

- typedef void * **RTI_RoutingServiceStreamWriter**

  StreamWriter.
- typedef int(* **RTI_RoutingServiceStreamWriter_WriteFcn**) (**RTI_RoutingServiceStreamWriter** stream_writer, const **RTI_RoutingServiceSample** *sample_list, const **RTI_RoutingServiceSampleInfo** *info_list, int count, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that writes a collection of data samples to an output stream.
- typedef void * **RTI_RoutingServiceStreamReader**

  StreamReader.
- typedef void(* **RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback**) (**RTI_RoutingServiceStreamReader** stream_reader, void *listener_data)

  Prototype of the callback used to notify of new samples.
- typedef void(* **RTI_RoutingServiceStreamReader_ReadFcn**) (**RTI_RoutingServiceStreamReader** stream_reader, **RTI_RoutingServiceSample** **sample_list, **RTI_RoutingServiceSampleInfo** **info_list, int *count, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that reads a collection of data samples and sample infos from an input stream.
- typedef void(* **RTI_RoutingServiceStreamReader_ReturnLoanFcn**) (**RTI_RoutingServiceStreamReader** stream_reader, **RTI_RoutingServiceSample** *sample_list, **RTI_RoutingServiceSampleInfo** *info_list, int count, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that returns the loan on the read samples and infos.
- typedef void * **RTI_RoutingServiceSession**

  Session.
- typedef void * **RTI_RoutingServiceConnection**

  Connection.
- typedef **RTI_RoutingServiceSession**(* **RTI_RoutingServiceConnection_CreateSessionFcn**) (**RTI_RoutingServiceConnection** connection, const struct **RTI_RoutingServiceProperties** *properties, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that creates a Session.
- typedef void(* **RTI_RoutingServiceConnection_DeleteSessionFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceSession** session, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that deletes a Session.
- typedef **RTI_RoutingServiceStreamReader**(* **RTI_RoutingServiceConnection_CreateStreamReaderFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceSession** session, const struct **RTI_RoutingServiceStreamInfo** *stream_info, const struct **RTI_RoutingServiceProperties** *properties, const struct **RTI_RoutingServiceStreamReaderListener** *listener, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that creates a StreamReader.
- typedef void(* **RTI_RoutingServiceConnection_DeleteStreamReaderFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceStreamReader** stream_reader, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that deletes a StreamReader.
- typedef **RTI_RoutingServiceStreamWriter**(* **RTI_RoutingServiceConnection_CreateStreamWriterFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceSession** session, const struct **RTI_RoutingServiceStreamInfo** *stream_info, const struct **RTI_RoutingServiceProperties** *properties, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that creates a StreamWriter.
- typedef void(* **RTI_RoutingServiceConnection_DeleteStreamWriterFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceStreamWriter** stream_writer, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that deletes a StreamWriter.

- typedef **RTI_RoutingServiceStreamReader**(∗ **RTI_RoutingServiceConnection_GetDiscoveryReaderFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that gets a built-in discovery StreamReader.
- typedef **RTI_RoutingServiceTypeRepresentation**(∗ **RTI_RoutingServiceConnection_CopyType↩ RepresentationFcn**) (**RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceTypeRepresentation↩ Kind** type_representation_kind, **RTI_RoutingServiceTypeRepresentation** type_representation, **RTI_↩ RoutingServiceEnvironment** ∗env)

  Prototype of the function that copies a type representation.
- typedef void(∗ **RTI_RoutingServiceConnection_DeleteTypeRepresentationFcn**) (**RTI_RoutingService↩ Connection** connection, **RTI_RoutingServiceTypeRepresentationKind** type_representation_kind, **RTI_↩ RoutingServiceTypeRepresentation** type_representation, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that deletes a type representation.
- typedef void ∗ **RTI_RoutingServiceAdapterEntity**

  Adapter entity.
- typedef void(∗ **RTI_RoutingServiceAdapterEntity_UpdateFcn**) (**RTI_RoutingServiceAdapterEntity** entity, const struct **RTI_RoutingServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that updates the configuration of an adapter entity.
- typedef void(∗ **RTI_RoutingServiceAdapterPlugin_DeleteConnectionFcn**) (struct **RTI_RoutingService↩ AdapterPlugin** ∗plugin, **RTI_RoutingServiceConnection** connection, **RTI_RoutingServiceEnvironment** ∗env)

  *Prototype of the function that deletes a **RTI_RoutingServiceConnection** (p. 54).*
- typedef **RTI_RoutingServiceConnection**(∗ **RTI_RoutingServiceAdapterPlugin_CreateConnectionFcn**) (struct **RTI_RoutingServiceAdapterPlugin** ∗plugin, const char ∗routing_service_name, const char ∗routing↩ _service_group_name, const struct **RTI_RoutingServiceStreamReaderListener** ∗input_stream_discovery↩ _listener, const struct **RTI_RoutingServiceStreamReaderListener** ∗output_stream_discovery_listener, const struct **RTI_RoutingServiceTypeInfo** ∗∗registeredTypes, int registeredTypeCount, const struct **RTI_Routing↩ ServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  *Prototype of the function that creates a **RTI_RoutingServiceConnection** (p. 54).*
- typedef void(∗ **RTI_RoutingServiceAdapterPlugin_DeleteFcn**) (struct **RTI_RoutingServiceAdapterPlugin** ∗plugin, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that deletes an adapter plugin.
- typedef struct **RTI_RoutingServiceAdapterPlugin** ∗(∗ **RTI_RoutingServiceAdapterPlugin_CreateFcn**) (const struct **RTI_RoutingServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that creates an adapter plugin.

7.3.1 Detailed Description

RTI Routing Service C Adapter API.

7.3.2 Typedef Documentation

7.3.2.1 RTI_RoutingServiceAdapterPlugin_DeleteConnectionFcn

```
typedef void(* RTI_RoutingServiceAdapterPlugin_DeleteConnectionFcn) (struct RTI_RoutingService↩
AdapterPlugin *plugin, RTI_RoutingServiceConnection connection, RTI_RoutingServiceEnvironment
*env)
```

Prototype of the function that deletes a **RTI_RoutingServiceConnection** (p. 54).

Connection objects are deleted when the DomainRoutes that contain them are disabled or RTI Routing Service is closed.

Required: Yes

Parameters

plugin	<<*in*>> Adapter plugin.
connection	<<*in*>> Connection to be deleted
env	<<*inout*>> Environment for error indications.

See also

RTI_RoutingServiceAdapterPlugin_CreateConnectionFcn (p. 121)

7.3.2.2 RTI_RoutingServiceAdapterPlugin_CreateConnectionFcn

```
typedef RTI_RoutingServiceConnection( * RTI_RoutingServiceAdapterPlugin_CreateConnectionFcn)
(struct RTI_RoutingServiceAdapterPlugin *plugin, const char *routing_service_name, const char
*routing_service_group_name, const struct RTI_RoutingServiceStreamReaderListener *input_stream_↩
discovery_listener, const struct RTI_RoutingServiceStreamReaderListener *output_stream_discovery↩
_listener, const struct RTI_RoutingServiceTypeInfo **registeredTypes, int registeredTypeCount,
const struct RTI_RoutingServiceProperties *properties, RTI_RoutingServiceEnvironment *env)
```

Prototype of the function that creates a **RTI_RoutingServiceConnection** (p. 54).

A Connection provides access to a data domain (such as a DDS domain or a JMS network provider).

Connection objects are created when the DomainRoutes that contain them are enabled.

Required: Yes

Parameters

plugin	<<*in*>> Adapter plugin.
routing_service_name	<<*in*>> The routing service execution name.
routing_service_group_name	<<*in*>> The group name of the routing service execution.
input_stream_discovery_listener	<<*in*>> The listener of the built-in StreamReader that notifies the discovery of new input streams.
output_stream_discovery_listener	<<*in*>> The listener of the built-in StreamReader that notifies the discovery of new output streams.

Parameters

registeredTypes	<<*in*>> A list of types that are registered in the corresponding XML connection with the tag <registered_type>
registeredTypeCount	<<*in*>> The number of elements of registeredTypes

Parameters

properties	<<*in*>> Configuration properties for the Connection.
env	<<*inout*>> Environment for error indications.

Returns

New Connection if successful. Otherwise, NULL.

See also

RTI_RoutingServiceAdapterPlugin_DeleteConnectionFcn (p. 120)

7.4 routingservice_adapter.h

Go to the documentation of this file.

```
1 /*
2  * (c) Copyright, Real-Time Innovations, 2002-2024.
3  * All rights reserved.
4  *
5  * No duplications, whole or partial, manual or electronic, may be made
6  * without express written permission.  Any such copies, or
7  * revisions thereof, must display this notice unaltered.
8  * This code contains trade secrets of Real-Time Innovations, Inc.
9  */
10
11 #ifndef routingservice_adapter_h
12 #define routingservice_adapter_h
13
14 #include "routingservice/routingservice_infrastructure.h"
15
16 #ifdef __cplusplus
17     extern "C" {
18 #endif
19
20 /*e \file
21   @brief RTI Routing Service C Adapter API
22 */
23
24 struct RTI_RoutingServiceAdapterPlugin;
25
26 /*****************************************************************************/
27 /* StreamWriter                                                              */
28 /*****************************************************************************/
29
30 /*e
31  * \dref_StreamWriter
32  */
33 typedef void *RTI_RoutingServiceStreamWriter;
34
35 /*e
36  * \dref_StreamWriter_write
37  */
38 typedef int (*RTI_RoutingServiceStreamWriter_WriteFcn)(
39         RTI_RoutingServiceStreamWriter stream_writer,
```

```
40         const RTI_RoutingServiceSample *sample_list,
41         const RTI_RoutingServiceSampleInfo *info_list,
42         int count,
43         RTI_RoutingServiceEnvironment *env);
44
45 /*************************************************************************/
46 /* StreamReader                                                          */
47 /*************************************************************************/
48
49 /*e
50  * \dref_StreamReader
51  */
52 typedef void *RTI_RoutingServiceStreamReader;
53
54 /*e
55  * \dref_StreamReaderListener_on_data_available
56  */
57 typedef void (*RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback)(
58         RTI_RoutingServiceStreamReader stream_reader,
59         void *listener_data);
60
61 /*e
62  * \dref_StreamReaderListener
63  */
64 struct RTI_RoutingServiceStreamReaderListener {
65     /*e
66      * \dref_StreamReaderListener_listener_data
67      */
68     void *listener_data;
69     /*e
70      * \dref_StreamReaderListener_on_data_available
71      */
72     RTI_RoutingServiceStreamReaderListener_OnDataAvailableCallback
73             on_data_available;
74 };
75
76 /*e
77  * \dref_StreamReader_read
78  */
79 typedef void (*RTI_RoutingServiceStreamReader_ReadFcn)(
80         RTI_RoutingServiceStreamReader stream_reader,
81         RTI_RoutingServiceSample **sample_list,
82         RTI_RoutingServiceSampleInfo **info_list,
83         int *count,
84         RTI_RoutingServiceEnvironment *env);
85
86 /*e
87  * \dref_StreamReader_return_loan
88  */
89 typedef void (*RTI_RoutingServiceStreamReader_ReturnLoanFcn)(
90         RTI_RoutingServiceStreamReader stream_reader,
91         RTI_RoutingServiceSample *sample_list,
92         RTI_RoutingServiceSampleInfo *info_list,
93         int count,
94         RTI_RoutingServiceEnvironment *env);
95
96 /*************************************************************************/
97 /* Session                                                               */
98 /*************************************************************************/
99
100 /*e
101  * \dref_Session
102  */
103 typedef void *RTI_RoutingServiceSession;
104
105 /*************************************************************************/
106 /* Connection                                                            */
107 /*************************************************************************/
108
109 /*e
110  * \dref_Connection
111  */
112 typedef void *RTI_RoutingServiceConnection;
113
114 /*i
115  */
116 typedef const char *(*RTI_RoutingServiceConnection_ToStringFcn)(
117         RTI_RoutingServiceConnection connection,
118         RTI_RoutingServiceEnvironment *env);
119
120 /*e
```

```
121  * \dref_Connection_create_session
122  */
123 typedef RTI_RoutingServiceSession (
124         *RTI_RoutingServiceConnection_CreateSessionFcn)(
125         RTI_RoutingServiceConnection connection,
126         const struct RTI_RoutingServiceProperties *properties,
127         RTI_RoutingServiceEnvironment *env);
128
129 /*e
130  * \dref_Connection_delete_session
131  */
132 typedef void (*RTI_RoutingServiceConnection_DeleteSessionFcn)(
133         RTI_RoutingServiceConnection connection,
134         RTI_RoutingServiceSession session,
135         RTI_RoutingServiceEnvironment *env);
136
137 /*e
138  * \dref_Connection_create_stream_reader
139  */
140 typedef RTI_RoutingServiceStreamReader (
141         *RTI_RoutingServiceConnection_CreateStreamReaderFcn)(
142         RTI_RoutingServiceConnection connection,
143         RTI_RoutingServiceSession session,
144         const struct RTI_RoutingServiceStreamInfo *stream_info,
145         const struct RTI_RoutingServiceProperties *properties,
146         const struct RTI_RoutingServiceStreamReaderListener *listener,
147         RTI_RoutingServiceEnvironment *env);
148
149 /*e
150  * \dref_Connection_delete_stream_reader
151  */
152 typedef void (*RTI_RoutingServiceConnection_DeleteStreamReaderFcn)(
153         RTI_RoutingServiceConnection connection,
154         RTI_RoutingServiceStreamReader stream_reader,
155         RTI_RoutingServiceEnvironment *env);
156
157 /*e
158  * \dref_Connection_create_stream_writer
159  */
160 typedef RTI_RoutingServiceStreamWriter (
161         *RTI_RoutingServiceConnection_CreateStreamWriterFcn)(
162         RTI_RoutingServiceConnection connection,
163         RTI_RoutingServiceSession session,
164         const struct RTI_RoutingServiceStreamInfo *stream_info,
165         const struct RTI_RoutingServiceProperties *properties,
166         RTI_RoutingServiceEnvironment *env);
167
168 /*e
169  * \dref_Connection_delete_stream_writer
170  */
171 typedef void (*RTI_RoutingServiceConnection_DeleteStreamWriterFcn)(
172         RTI_RoutingServiceConnection connection,
173         RTI_RoutingServiceStreamWriter stream_writer,
174         RTI_RoutingServiceEnvironment *env);
175
176 /*e
177  * \ingroup RTI_RoutingServiceAdapterModule
178  *
179  * \brief Prototype of the function that gets a built-in discovery StreamReader.
180  *
181  * There are two built-in discovery StreamReaders:
182  *
183  * The first StreamReader provides information about output streams. An output
184  * stream is a stream to which StreamWriters can write data. Disposed scenarios,
185  * where an output streams dissapears, are also notified using this
186  * StreamReader.
187  *
188  * The second StreamReader provides information about input streams. An input
189  * stream is a stream from which StreamReaders can read data. Disposed
190  * scenarios, where an output streams dissapears, are also notified using this
191  * StreamReader.
192  *
193  * The StreamReaderListeners associated with the built-in discovery
194  * StreamReaders are provided as parameters to \ref
195  * RTI_RoutingServiceAdapterPlugin_CreateConnectionFcn.
196  *
197  * <b>Required:</b> No
198  *
199  * The implementation of this function is optional. However, if none of the
200  * adapters in a domain route implement the discovery API, the routes' types
201  * must be declared in the XML configuration file.
```

```
202  *
203  * @param connection \rs_st_in Connection.
204  * @param env \rs_st_inout Environment for error indications.
205  *
206  * @return Built-in dicovery StreamReader if successful. Otherwise, NULL.
207  */
208 typedef RTI_RoutingServiceStreamReader (
209         *RTI_RoutingServiceConnection_GetDiscoveryReaderFcn)(
210         RTI_RoutingServiceConnection connection,
211         RTI_RoutingServiceEnvironment *env);
212
213 /*e
214  * \dref_Connection_copy_type_representation
215  */
216 typedef RTI_RoutingServiceTypeRepresentation (
217         *RTI_RoutingServiceConnection_CopyTypeRepresentationFcn)(
218         RTI_RoutingServiceConnection connection,
219         RTI_RoutingServiceTypeRepresentationKind type_representation_kind,
220         RTI_RoutingServiceTypeRepresentation type_representation,
221         RTI_RoutingServiceEnvironment *env);
222
223 /*e
224  * \dref_Connection_delete_type_representation
225  */
226 typedef void (*RTI_RoutingServiceConnection_DeleteTypeRepresentationFcn)(
227         RTI_RoutingServiceConnection connection,
228         RTI_RoutingServiceTypeRepresentationKind type_representation_kind,
229         RTI_RoutingServiceTypeRepresentation type_representation,
230         RTI_RoutingServiceEnvironment *env);
231
232 /*****************************************************************************/
233 /* Entity                                                                    */
234 /*****************************************************************************/
235
236 /*e
237  * \dref_Entity
238  */
239 typedef void *RTI_RoutingServiceAdapterEntity;
240
241 /*e
242  * \dref_Entity_update
243  */
244 typedef void (*RTI_RoutingServiceAdapterEntity_UpdateFcn)(
245         RTI_RoutingServiceAdapterEntity entity,
246         const struct RTI_RoutingServiceProperties *properties,
247         RTI_RoutingServiceEnvironment *env);
248
249 /*****************************************************************************/
250 /* Adapter                                                                   */
251 /*****************************************************************************/
252
253 /*e
254  * \dref_Adapter_delete_connection
255  */
256 typedef void (*RTI_RoutingServiceAdapterPlugin_DeleteConnectionFcn)(
257         struct RTI_RoutingServiceAdapterPlugin *plugin,
258         RTI_RoutingServiceConnection connection,
259         RTI_RoutingServiceEnvironment *env);
260
261 /*e
262  * \dref_Adapter_create_connection
263  */
264 typedef RTI_RoutingServiceConnection (
265         *RTI_RoutingServiceAdapterPlugin_CreateConnectionFcn)(
266         struct RTI_RoutingServiceAdapterPlugin *plugin,
267         const char *routing_service_name,
268         const char *routing_service_group_name,
269         const struct RTI_RoutingServiceStreamReaderListener
270                 *input_stream_discovery_listener,
271         const struct RTI_RoutingServiceStreamReaderListener
272                 *output_stream_discovery_listener,
273         const struct RTI_RoutingServiceTypeInfo **registeredTypes,
274         int registeredTypeCount,
275         const struct RTI_RoutingServiceProperties *properties,
276         RTI_RoutingServiceEnvironment *env);
277
278 /*e
279  * \ingroup RTI_RoutingServiceAdapterModule
280  * \brief Prototype of the function that deletes an adapter plugin.
281  *
282  * Adapter plugins are deleted when \product is closed.
```

```
283  *
284  * <b>Required:</b> yes
285  *
286  * @param plugin \rs_st_in Adapter plugin to be deleted.
287  * @param env \rs_st_inout Environment for error indications.
288  *
289  * @see \ref RTI_RoutingServiceAdapterPlugin_DeleteFcn
290  */
291 typedef void (*RTI_RoutingServiceAdapterPlugin_DeleteFcn)(
292         struct RTI_RoutingServiceAdapterPlugin *plugin,
293         RTI_RoutingServiceEnvironment *env);
294
295 /*e \ingroup RTI_RoutingServiceAdapterModule
296  \brief Adapter plugin.
297
298  This structure contains the plugin implementation as a set of function pointers.
299
300  C adapters are registered with \product using the XML tag &lt;adapter_plugin&gt;
301
302  For example:
303
304  \verbatim
305  <dds>
306      ...
307      <plugin_library name="MyAdapterLib">
308          <adapter_plugin name="MyAdapterPlugin">
309             <dll>mycadapter</dll>
310             <create_function>
311                 MyAdapterPlugin_create
312             </create_function>
313          </adapter_plugin>
314          ...
315      </plugin_library>
316      ...
317      <routing_service>
318      ...
319      </routing_service>
320      ...
321  </dds>
322  \endverbatim
323
324  In the previous example, MyAdapterPlugin_create is the function that creates
325  and instance of the adapter plugin.
326
327  This function must have the following prototype:
328
329  @see \ref RTI_RoutingServiceAdapterPlugin_CreateFcn
330 */
331 struct RTI_RoutingServiceAdapterPlugin {
332
333     /* Adapter interface */
334     int _init;
335     struct RTI_RoutingServiceVersion _rs_version;
336
337     /*e \brief The version of this adapter plugin */
338     struct RTI_RoutingServiceVersion plugin_version;
339
340     /*e \brief Handles the deletion of the adapter plugin (required). */
341     RTI_RoutingServiceAdapterPlugin_DeleteFcn adapter_plugin_delete;
342
343     /*e @brief Handles the creation of connections (required). */
344     RTI_RoutingServiceAdapterPlugin_CreateConnectionFcn adapter_plugin_create_connection;
345     /*e @brief Handles the deletion of connections (required). */
346     RTI_RoutingServiceAdapterPlugin_DeleteConnectionFcn adapter_plugin_delete_connection;
347
348     /* Connection interface */
349
350     /*e @brief Handles the creation of sessions (optional). */
351     RTI_RoutingServiceConnection_CreateSessionFcn connection_create_session;
352     /*e @brief Handles the deletion of sessions (optional). */
353     RTI_RoutingServiceConnection_DeleteSessionFcn connection_delete_session;
354     /*e @brief Handles the creation of stream readers (optional for write-only adapters). */
355     RTI_RoutingServiceConnection_CreateStreamReaderFcn connection_create_stream_reader;
356     /*e @brief Handles the deletion of stream readers (optional for write-only adapters). */
357     RTI_RoutingServiceConnection_DeleteStreamReaderFcn connection_delete_stream_reader;
358     /*e @brief Handles the creation of stream writers (optional for read-only adapters). */
359     RTI_RoutingServiceConnection_CreateStreamWriterFcn connection_create_stream_writer;
360     /*e @brief Handles the deletion of stream writers (optional for read-only adapters). */
361     RTI_RoutingServiceConnection_DeleteStreamWriterFcn connection_delete_stream_writer;
362     /*e
363      * @brief Gets the input stream discovery reader (optional).
```

```
364      */
365     RTI_RoutingServiceConnection_GetDiscoveryReaderFcn connection_get_input_stream_discovery_reader;
366     /*e
367      * @brief Gets the output stream discovery reader (optional).
368      */
369     RTI_RoutingServiceConnection_GetDiscoveryReaderFcn connection_get_output_stream_discovery_reader;
370     /*e @brief Handles the copy of type representations (optional). */
371     RTI_RoutingServiceConnection_CopyTypeRepresentationFcn connection_copy_type_representation;
372     /*e @brief Handles the deletion of type representations (optional). */
373     RTI_RoutingServiceConnection_DeleteTypeRepresentationFcn connection_delete_type_representation;
374     /*e @brief Returns the string representation of a connection for logging purposes (optional). */
375     RTI_RoutingServiceConnection_ToStringFcn connection_to_string;
376     /*e @brief Handles configuration changes in a connection (optional). */
377     RTI_RoutingServiceAdapterEntity_UpdateFcn connection_update;
378
379     /* Session interface */
380
381     /*e @brief \not_supported Handles configuration changes in a session (optional). */
382     RTI_RoutingServiceAdapterEntity_UpdateFcn session_update;
383
384     /* StreamReader interface */
385
386     /*e @brief Reads from an input stream (optional for write-only adapters). */
387     RTI_RoutingServiceStreamReader_ReadFcn stream_reader_read;
388     /*e @brief Returns the loan on the read samples and infos (optional for write-only adapters). */
389     RTI_RoutingServiceStreamReader_ReturnLoanFcn stream_reader_return_loan;
390     /*e @brief \not_supported Handles configuration changes in a stream reader (optional). */
391     RTI_RoutingServiceAdapterEntity_UpdateFcn stream_reader_update;
392
393     /* StreamWriter interface */
394
395     /*e @brief Writes to an output stream (optional for read-only adapters). */
396     RTI_RoutingServiceStreamWriter_WriteFcn stream_writer_write;
397     /*e @brief \not_supported Handles configuration changes in a stream writer (optional). */
398     RTI_RoutingServiceAdapterEntity_UpdateFcn stream_writer_update;
399
400     /*e @brief A place for adapter implementors to keep a pointer to data that
401         may be needed by the implementation. */
402     void * user_object;
403 };
404
405 /*e \ingroup RTI_RoutingServiceAdapterModule
406  @brief Prototype of the function that creates an adapter plugin.
407
408  The name of the function that implements this prototype
409  must be provided to \product using the tag &lt;create_function&gt;
410  when the adapter plugin is registered. For example:
411
412  \verbatim
413  <dds>
414      ...
415      <plugin_library name="MyAdapterLib">
416          <adapter_plugin name="MyAdapterPlugin">
417             <dll>mycadapter</dll>
418             <create_function>
419                 MyAdapterPlugin_create
420             </create_function>
421          </adapter_plugin>
422          ...
423      </plugin_library>
424      ...
425      <routing_service>
426      ...
427      </routing_service>
428      ...
429  </dds>
430  \endverbatim
431
432  <b>Required:</b> yes
433
434  @param properties Configuration properties for the adapter.
435  @param env \rs_st_inout Environment for error indications.
436
437  @return New plugin instance if successful. Otherwise, NULL.
438
439  @see \ref RTI_RoutingServiceAdapterPlugin_DeleteFcn
440  @see \ref RTI_RoutingServiceAdapterPlugin_initialize
441 */
442 typedef struct RTI_RoutingServiceAdapterPlugin *(
443         *RTI_RoutingServiceAdapterPlugin_CreateFcn)(
444         const struct RTI_RoutingServiceProperties *properties,
```

```
445         RTI_RoutingServiceEnvironment *env);
446
447
448 #define RTI_ROUTING_SERVICE_ADAPTER_PLUGIN_INIT_NUMBER (7654321)
449 /*e \ingroup RTI_RoutingServiceAdapterModule
450  \hideinitializer
451  @brief Initializes the adapter plugin structure.
452
453  This macro must be called to initialize the
454  return value of RTI_RoutingServiceAdapterPlugin_CreateFcn
455
456  @param adapter Pointer to the adapter plugin structure
457
458  @see \ref RTI_RoutingServiceAdapterPlugin_CreateFcn
459 */
460 #define RTI_RoutingServiceAdapterPlugin_initialize(adapter)                    \
461 {                                                                              \
462     struct RTI_RoutingServiceVersion rsVersion = RTI_ROUTING_SERVICE_VERSION; \
463     struct RTI_RoutingServiceVersion adapterVersion = {0,0,0,0};              \
464     (adapter)->_init = RTI_ROUTING_SERVICE_ADAPTER_PLUGIN_INIT_NUMBER;        \
465     (adapter)->_rs_version = rsVersion;                                       \
466     (adapter)->plugin_version = adapterVersion;                               \
467     (adapter)->adapter_plugin_delete = 0;                                     \
468     (adapter)->adapter_plugin_create_connection = 0;                          \
469     (adapter)->adapter_plugin_delete_connection = 0;                          \
470     (adapter)->connection_create_session = 0;                                 \
471     (adapter)->connection_delete_session = 0;                                 \
472     (adapter)->connection_create_stream_reader = 0;                           \
473     (adapter)->connection_delete_stream_reader = 0;                           \
474     (adapter)->connection_create_stream_writer = 0;                           \
475     (adapter)->connection_delete_stream_writer = 0;                           \
476     (adapter)->connection_get_input_stream_discovery_reader = 0;              \
477     (adapter)->connection_get_output_stream_discovery_reader = 0;             \
478     (adapter)->connection_copy_type_representation = 0;                       \
479     (adapter)->connection_delete_type_representation = 0;                     \
480     (adapter)->connection_to_string = 0;                                      \
481     (adapter)->connection_update = 0;                                         \
482     (adapter)->session_update = 0;                                            \
483     (adapter)->stream_reader_read = 0;                                        \
484     (adapter)->stream_reader_return_loan = 0;                                 \
485     (adapter)->stream_reader_update = 0;                                      \
486     (adapter)->stream_writer_write = 0;                                       \
487     (adapter)->stream_writer_update = 0;                                      \
488     (adapter)->user_object = 0;                                               \
489 }
490
491 #ifdef __cplusplus
492 }       /* extern "C" */
493 #endif
494
495 #endif /* routingservice_adapter_h */
```

7.5 routingservice_processor.h File Reference

RTI Routing Service Processor API.

```
#include "routingservice/routingservice_infrastructure.h"
#include "routingservice/routingservice_adapter_new.h"
```

Data Structures

- struct **RTI_RoutingServiceLoanedSamples**

  A pair of sample and sample info arrays. They are assumed to be of the same length.
- struct **RTI_RoutingServiceProcessor**

  Main Processor class.
- struct **RTI_RoutingServiceProcessorPlugin**

  ProcessorPlugin class.

Macros

- #define **RTI_RoutingServiceProcessorPlugin_initialize**(plugin)

  Utility macro to initialize a ProcessorPlugin object.

Typedefs

- typedef void(∗ **RTI_RoutingServiceProcessor_OnRouteEventFcn**) (void ∗processor_data, RTI_Routing↩ ServiceRouteEvent ∗route_event, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that processes an event that occurred in the Route where the Processor is installed.
- typedef void(∗ **RTI_RoutingServiceProcessor_UpdateFcn**) (void ∗processor_data, const struct **RTI_↩ RoutingServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that updates a Processor configuration.
- typedef struct **RTI_RoutingServiceProcessorPlugin** ∗(∗ **RTI_RoutingServiceProcessorPlugin_CreateFcn**) (const struct **RTI_RoutingServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that creates a ProcessorPlugin.
- typedef void(∗ **RTI_RoutingServiceProcessorPlugin_DeleteFcn**) (struct **RTI_RoutingServiceProcessor↩ Plugin** ∗plugin, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that deletes a ProcessorPlugin.
- typedef struct **RTI_RoutingServiceProcessor** ∗(∗ **RTI_RoutingServiceProcessorPlugin_Create↩ ProcessorFcn**) (void ∗processor_plugin_data, RTI_RoutingServiceRoute ∗route, const struct **RTI_Routing↩ ServiceProperties** ∗properties, **RTI_RoutingServiceEnvironment** ∗env)

  Prototype of the function that creates a route Processor.

Enumerations

- enum **RTI_RoutingServiceRouteEventKind**

  Representation of the different event kinds triggered for a Route.

Functions

- const char ∗ **RTI_RoutingServiceInput_get_name** (RTI_RoutingServiceInput ∗self)

  Returns the name of the specified input.
- const struct **RTI_RoutingServiceStreamInfo** ∗ **RTI_RoutingServiceInput_get_stream_info** (RTI_Routing↩ ServiceInput ∗self)
- DDS_Boolean **RTI_RoutingServiceInput_is_active** (RTI_RoutingServiceInput ∗self)

  Checks whether an Input is active.
- DDS_Boolean **RTI_RoutingServiceInput_take** (RTI_RoutingServiceInput ∗self, struct **RTI_RoutingService↩ LoanedSamples** ∗loaned_samples)

  Takes all the available samples in this Input into the specified loaned samples instance.
- DDS_Boolean **RTI_RoutingServiceInput_take_w_selector** (RTI_RoutingServiceInput ∗self, struct **RTI_↩ RoutingServiceLoanedSamples** ∗loaned_samples, const struct RTI_RoutingServiceSelectorState ∗selector)

  Takes all the available samples in this Input into the specified loaned samples instance, filtering samples based on the given data selector.
- DDS_Boolean **RTI_RoutingServiceInput_read** (RTI_RoutingServiceInput ∗self, struct **RTI_RoutingService↩ LoanedSamples** ∗loaned_samples)

  Reads all the available samples in this Input into the specified loaned samples instance.

- DDS_Boolean **RTI_RoutingServiceInput_read_w_selector** (RTI_RoutingServiceInput *self, struct **RTI_RoutingServiceLoanedSamples** *loaned_samples, const struct RTI_RoutingServiceSelectorState *selector)

  Reads all the available samples in this Input into the specified loaned samples instance, filtering samples based on the given data selector. Cannot be null.
- DDS_Boolean **RTI_RoutingServiceInput_return_loan** (RTI_RoutingServiceInput *self, struct **RTI_RoutingServiceLoanedSamples** *loaned_samples)

  Returns a loan on the read or taken samples.
- RTI_RoutingServiceSelectorStateQueryData **RTI_RoutingServiceInput_create_content_query** (RTI_RoutingServiceInput *self, RTI_RoutingServiceSelectorStateQueryData old_query_data, const struct RTI_RoutingServiceSelectorContent *content)

  Creates a query object that selects the data with the specified filter.
- DDS_Boolean **RTI_RoutingServiceInput_delete_content_query** (RTI_RoutingServiceInput *self, RTI_RoutingServiceSelectorStateQueryData query_data)

  Deletes a content query created from this Input.
- const char * **RTI_RoutingServiceOutput_get_name** (RTI_RoutingServiceOutput *self)

  Returns the name of the specified output.
- const struct **RTI_RoutingServiceStreamInfo** * **RTI_RoutingServiceOutput_get_stream_info** (RTI_RoutingServiceOutput *self)

  Returns the type information associated with the specified Output. The type information refers to the format of the stream that this Output's write operation expects to receive from the processor of the route.
- DDS_Boolean **RTI_RoutingServiceOutput_write** (RTI_RoutingServiceOutput *self, const **RTI_RoutingServiceSample** *sample_array, const **RTI_RoutingServiceSampleInfo** *sample_info_array, int *array_length)

  Writes a list of data and information samples.
- DDS_Boolean **RTI_RoutingServiceOutput_write_sample** (RTI_RoutingServiceOutput *self, const **RTI_RoutingServiceSample** data, const **RTI_RoutingServiceSampleInfo** info)

  Writes a single sample, optionally with metatada indicated by an info sample.
- int **RTI_RoutingServiceRoute_get_input_count** (RTI_RoutingServiceRoute *self)

  Returns the number of inputs associated with this Route.
- DDS_Boolean **RTI_RoutingServiceRoute_is_input_enabled** (RTI_RoutingServiceRoute *self, int index)

  Checks whether the Input with the specified index is enabled.
- RTI_RoutingServiceInput * **RTI_RoutingServiceRoute_get_input_at** (RTI_RoutingServiceRoute *self, int index)

  Returns the Input at the specified index.
- RTI_RoutingServiceInput * **RTI_RoutingServiceRoute_lookup_input_by_name** (RTI_RoutingServiceRoute *self, const char *input_name)

  Looks up the specified Input by its configuration name.
- int **RTI_RoutingServiceRoute_get_output_count** (RTI_RoutingServiceRoute *self)

  Returns the number of outputs associated with this Route.
- DDS_Boolean **RTI_RoutingServiceRoute_is_output_enabled** (RTI_RoutingServiceRoute *self, int index)

  Checks whether the Output at the specified index is enabled.
- RTI_RoutingServiceOutput * **RTI_RoutingServiceRoute_get_output_at** (RTI_RoutingServiceRoute *self, int index)

  Returns the Output at the specified index.
- RTI_RoutingServiceOutput * **RTI_RoutingServiceRoute_lookup_output_by_name** (RTI_RoutingServiceRoute *self, const char *output_name)

  Looks up the specified Output by its configuration name.
- void **RTI_RoutingServiceRoute_wakeup_route** (RTI_RoutingServiceRoute *route)

  Sends a wakeup event to the specified Route. This operation can be safely called from any thread context.
- void **RTI_RoutingServiceRoute_get_period** (RTI_RoutingServiceRoute *self, struct DDS_Duration_t *period)

Get the current period of the periodic event for this Route.

- void **RTI_RoutingServiceRoute_set_period** (RTI_RoutingServiceRoute *self, const struct DDS_Duration_t *period)

  Changes the periodic event period for this route.

- RTI_RoutingServiceInput * **RTI_RoutingServiceRoute_get_first_input** (RTI_RoutingServiceRoute *self)

  Returns the first enabled Input in this route.

- RTI_RoutingServiceInput * **RTI_RoutingServiceRoute_get_next_input** (RTI_RoutingServiceRoute *self, RTI_RoutingServiceInput *prev_input)

  Returns the next enabled Input sibling to the specified input.

- RTI_RoutingServiceOutput * **RTI_RoutingServiceRoute_get_first_output** (RTI_RoutingServiceRoute *self)

  Returns the first enabled output in this route.

- RTI_RoutingServiceOutput * **RTI_RoutingServiceRoute_get_next_output** (RTI_RoutingServiceRoute *self, RTI_RoutingServiceOutput *prev_output)

  Returns the next enabled Output sibling to the specified output.

- const char * **RTI_RoutingServiceRoute_get_full_name** (RTI_RoutingServiceRoute *self)

  Returns the fully-qualified name of this Route, derived from the XML configuration.

- **RTI_RoutingServiceRouteEventKind** **RTI_RoutingServiceRouteEvent_get_kind** (RTI_RoutingServiceRouteEvent *self)

  Returns the kind of this RouteEvent.

- RTI_RoutingServiceRoute * **RTI_RoutingServiceRouteEvent_get_route** (RTI_RoutingServiceRouteEvent *self)

  Returns the Route that triggered the event.

- void * **RTI_RoutingServiceRouteEvent_get_event_data** (RTI_RoutingServiceRouteEvent *self)

  Returns the data associated with the event.

- void * **RTI_RoutingServiceRouteEvent_get_affected_entity** (RTI_RoutingServiceRouteEvent *self)

  Returns the entity that is directly affected by the event.

7.5.1 Detailed Description

RTI Routing Service Processor API.

7.6 routingservice_processor.h

Go to the documentation of this file.

```
1 /*
2  * (c) Copyright, Real-Time Innovations, 2002-2024.
3  * All rights reserved.
4  *
5  * No duplications, whole or partial, manual or electronic, may be made
6  * without express written permission.  Any such copies, or
7  * revisions thereof, must display this notice unaltered.
8  * This code contains trade secrets of Real-Time Innovations, Inc.
9  */
10
11 #ifndef routingservice_processor_h
12 #define routingservice_processor_h
13
14 #include "routingservice/routingservice_infrastructure.h"
15 #include "routingservice/routingservice_adapter_new.h"
16
17 #ifdef __cplusplus
18     extern "C" {
19 #endif
```

```
20
21 /*e \file
22   @brief RTI Routing Service Processor API
23 */
24
25 /*************************************************************************/
26 /* Route                                                                 */
27 /*************************************************************************/
28
29 /*e \defgroup RTI_RoutingServiceProcessorModule RTI Routing Service Processor API
30  *
31  * \ingroup ROUTER
32  *
33  * @brief Entity responsible for administering inputs and outputs
34  *
35  */
36 struct RTI_RoutingServiceRouteImpl;
37
38 typedef struct RTI_RoutingServiceRouteImpl RTI_RoutingServiceRoute;
39
40 /*
41  * --- RTI_RoutingServiceInput -------------------------------------------
42  */
43
44 /*e
45  * \ingroup RTI_RoutingServiceProcessorModule
46  *
47  * @brief A pair of sample and sample info arrays. They are assumed to be of the
48  *        same length.
49  *
50  * @see RTI_RoutingServiceSample
51  * @see RTI_RoutingServiceSampleInfo
52  */
53 struct RTI_RoutingServiceLoanedSamples {
54     RTI_RoutingServiceSample *data_array;
55     RTI_RoutingServiceSampleInfo *info_array;
56     int length;
57 };
58
59 #define RTI_RoutingServiceLoanedSamples_INITIALIZER {\
60     NULL,\
61     NULL,\
62     0 \
63 }
64
65 struct RTI_RoutingServiceInputImpl;
66
67 typedef struct RTI_RoutingServiceInputImpl RTI_RoutingServiceInput;
68
69 /*e \ingroup RTI_RoutingServiceProcessorModule
70  *
71  * @brief Returns the name of the specified input.
72  *
73  * The input name is given by the attribute of the corresponding XML tag
74  * in the configuration.
75  *
76  * @param self \b In. Input for which the name is returned.
77  *
78  */
79 extern ROUTERDllExport
80 const char* RTI_RoutingServiceInput_get_name(RTI_RoutingServiceInput *self);
81
82 /*e \ingroup RTI_RoutingServiceProcessorModule
83  *
84  * Returns the type information associated with the specified Input.
85  * The TypeInformation refers to the format of the stream that the specified
86  * Input's read operation expects to receive.
87  *
88  * @param self \b In. Input for which the type information is returned.
89  */
90 extern ROUTERDllExport
91 const struct RTI_RoutingServiceStreamInfo *
92 RTI_RoutingServiceInput_get_stream_info(RTI_RoutingServiceInput *self);
93
94 /*e \ingroup RTI_RoutingServiceProcessorModule
95  *
96  * @brief Checks whether an Input is active
97  *
98  * An Input is active when it has data available but not yet read. That is,
99  * the data available notification has been received but a call to read or take the
100  * samples has not been made.
```

```
101  *
102  * @param self \b In. Input to be checked. Cannot be null.
103  *
104  * @return \c DDS_BOOLEAN_FALSE if the Input is inactive, otherwise,
105  *         \c DDS_BOOLEAN_TRUE.
106  */
107 extern ROUTERDllExport
108 DDS_Boolean RTI_RoutingServiceInput_is_active(RTI_RoutingServiceInput *self);
109
110 /*e \ingroup RTI_RoutingServiceProcessorModule
111  *
112  * @brief Takes all the available samples in this Input into the specified
113  *        loaned samples instance.
114  *
115  * @pre Input is enabled
116  *
117  * This operation will call \c take() on the underlying
118  * RTI_RoutingServiceStreamReader, and the samples will be loaned into the
119  * given loaned samples collection. Samples loaned have to be returned by
120  * calling RTI_RoutingServiceInput_return_loan().
121  *
122  * @param self \b In. Input to take the samples from. Cannot be null.
123  * @param loaned_samples \b Out. Samples taken will be stored here. Cannot be
124  *                null.
125  *
126  * @return \c DDS_BOOLEAN_FALSE if the samples could not be taken,
127  *         \c DDS_BOOLEAN_TRUE otherwise.
128  *
129  * @see RTI_RoutingServiceLoanedSamples
130  */
131 extern ROUTERDllExport
132 DDS_Boolean RTI_RoutingServiceInput_take(
133         RTI_RoutingServiceInput *self,
134         struct RTI_RoutingServiceLoanedSamples *loaned_samples);
135
136 /*e \ingroup RTI_RoutingServiceProcessorModule
137  *
138  * @brief Takes all the available samples in this Input into the specified
139  *        loaned samples instance, filtering samples based on the given
140  *        data selector.
141  *
142  * @pre Input is enabled
143  *
144  * This operation will call \c take() on the underlying
145  * RTI_RoutingServiceStreamReader, and the samples will be loaned into the
146  * given loaned samples collection. Samples loaned have to be returned by
147  * calling RTI_RoutingServiceInput_return_loan().
148  *
149  * @param self \b In. Input to take the samples from.
150  * @param loaned_samples \b Out. Samples taken will be stored here.
151  * @param selector \b In. Selection criteria for the samples to be taken.
152  *
153  * @return \c DDS_BOOLEAN_FALSE if the samples could not be taken,
154  *         \c DDS_BOOLEAN_TRUE otherwise.
155  *
156  * @see RTI_RoutingServiceLoanedSamples
157  * @see RTI_RoutingServiceSelectorState
158  */
159 extern ROUTERDllExport
160 DDS_Boolean RTI_RoutingServiceInput_take_w_selector(
161         RTI_RoutingServiceInput *self,
162         struct RTI_RoutingServiceLoanedSamples *loaned_samples,
163         const struct RTI_RoutingServiceSelectorState *selector);
164
165 /*e \ingroup RTI_RoutingServiceProcessorModule
166  *
167  * @brief Reads all the available samples in this Input into the specified
168  *        loaned samples instance.
169  *
170  * @pre Input is enabled
171  *
172  * This operation will call \c read() on the underlying
173  * RTI_RoutingServiceStreamReader, and the samples will be loaned into the
174  * given loaned samples collection. Samples loaned have to be returned by
175  * calling RTI_RoutingServiceInput_return_loan().
176  *
177  * @param self \b In. Input to read the samples from.
178  * @param loaned_samples \b Out. Samples read will be stored here.
179  *
180  * @return \c DDS_BOOLEAN_FALSE) if the samples could not be read,
181  *         \c DDS_BOOLEAN_TRUE otherwise.
```

```
182  * @see RTI_RoutingServiceLoanedSamples
183  */
184 extern ROUTERDllExport
185 DDS_Boolean RTI_RoutingServiceInput_read(
186         RTI_RoutingServiceInput *self,
187         struct RTI_RoutingServiceLoanedSamples *loaned_samples);
188
189 /*e \ingroup RTI_RoutingServiceProcessorModule
190  *
191  * @brief Reads all the available samples in this Input into the
192  *        specified loaned samples instance, filtering samples based on the
193  *        given data selector. Cannot be null.
194  *
195  * @pre Input is enabled
196  *
197  * This operation will call \c read() on the underlying
198  * RTI_RoutingServiceStreamReader, and the samples will be loaned into the
199  * given loaned samples collection. Samples loaned have to be returned by
200  * calling RTI_RoutingServiceInput_return_loan().
201  *
202  * @param self \b In. Input to read the samples from. Cannot be null.
203  * @param loaned_samples \b Out. Samples taken will be stored here. Cannot be
204  *                null.
205  * @param selector \b In. Selection criteria for the samples to be taken.
206  *
207  * @return \c DDS_BOOLEAN_FALSE if the samples could not be read,
208  *         \c DDS_BOOLEAN_TRUE otherwise.
209  *
210  * @see RTI_RoutingServiceLoanedSamples
211  * @see RTI_RoutingServiceSelectorState
212  * @see RTI_RoutingServiceInput_return_loan
213  */
214 extern ROUTERDllExport
215 DDS_Boolean RTI_RoutingServiceInput_read_w_selector(
216         RTI_RoutingServiceInput *self,
217         struct RTI_RoutingServiceLoanedSamples *loaned_samples,
218         const struct RTI_RoutingServiceSelectorState *selector);
219
220 /*e \ingroup RTI_RoutingServiceProcessorModule
221  *
222  * @brief Returns a loan on the read or taken samples.
223  *
224  * Call this method to indicate that you're done accessing
225  * the collection of samples obtained by an earlier
226  * invocation of RTI_RoutingServiceInput_take(),
227  * RTI_RoutingServiceInput_take_w_selector(), RTI_RoutingServiceInput_read() or
228  * RTI_RoutingServiceInput_read_w_selector().
229  *
230  * @param self \b In. Input the samples were taken/read from. Cannot be null.
231  * @param loaned_samples \b Inout. The loaned samples to be returned. Cannot be
232  *                null.
233  *
234  * @return \c DDS_BOOLEAN_FALSE if the samples could not be returned successfully,
235  *         \c DDS_BOOLEAN_TRUE otherwise.
236  *
237  * @see RTI_RoutingServiceLoanedSamples
238  * */
239 extern ROUTERDllExport
240 DDS_Boolean RTI_RoutingServiceInput_return_loan(
241         RTI_RoutingServiceInput *self,
242         struct RTI_RoutingServiceLoanedSamples *loaned_samples);
243
244 /*e \ingroup RTI_RoutingServiceProcessorModule
245  *
246  * @brief Creates a query object that selects the data with the specified
247  * filter.
248  *
249  * @pre Input is enabled.
250  *
251  * This operation allows reading data with a SelectorState that contains
252  * a query object returned by this operation.
253  *
254  * A query object type is implementation-dependent and guaranteed to
255  * to be used only within the same StreamReader that created it. Because
256  * a query object may be an expensive resource, this operation allows
257  * receiving a previously created query for a potential reuse and update of
258  * its filter.
259  *
260  * @param self \b In. Input used to create the content query. Cannot be null.
261  * @param old_query_data \b Inout. A previously created query object that
262  *            is provided for potential reuse and update of its filter.
```

```
263  * @param content \b In. The filter to be applied. Cannot be null.
264  *
265  * @return The new or updated query object.
266  *
267  * @see RTI_RoutingServiceSelectorStateQueryData
268  * @see RTI_RoutingServiceSelectorContent
269  */
270 extern ROUTERDllExport
271 RTI_RoutingServiceSelectorStateQueryData
272 RTI_RoutingServiceInput_create_content_query(
273         RTI_RoutingServiceInput *self,
274         RTI_RoutingServiceSelectorStateQueryData old_query_data,
275         const struct RTI_RoutingServiceSelectorContent *content);
276
277 /*e \ingroup RTI_RoutingServiceProcessorModule
278  *
279  * @brief Deletes a content query created from this Input.
280  *
281  * @pre Input is enabled
282  *
283  * @param self \b In. Input used to create the content query. Cannot be null.
284  * @param query_data \b In. The query object to be deleted. Cannot be null.
285  *
286  * @return \c DDS_BOOLEAN_FALSE if the operation fails, \c DDS_BOOLEAN_TRUE
287  *         otherwise.
288  */
289 extern ROUTERDllExport
290 DDS_Boolean RTI_RoutingServiceInput_delete_content_query(
291         RTI_RoutingServiceInput *self,
292         RTI_RoutingServiceSelectorStateQueryData query_data);
293
294 /*
295  * --- RTI_RoutingServiceOutput ------------------------------------------------
296  */
297
298 struct RTI_RoutingServiceOutputImpl;
299
300 typedef struct RTI_RoutingServiceOutputImpl RTI_RoutingServiceOutput;
301
302 /*e \ingroup RTI_RoutingServiceProcessorModule
303  *
304  * @brief Returns the name of the specified output.
305  *
306  * The output name is given by the attribute of the corresponding XML tag
307  * in the configuration.
308  *
309  * @param self \b In. Output for which the name is required.
310  *
311  */
312 extern ROUTERDllExport
313 const char* RTI_RoutingServiceOutput_get_name(RTI_RoutingServiceOutput *self);
314
315 /*e \ingroup RTI_RoutingServiceProcessorModule
316  *
317  * @brief Returns the type information associated with the specified Output.
318  * The type information refers to the format of the stream that this Output's
319  * write operation expects to receive from the processor of the route.
320  *
321  * The TypeInformation may be the one coming from the Transformation if this
322  * Output has one. In this case, the type information is the one corresponding
323  * to the Output side of the Transformation.
324  *
325  * @param self \b In. Output for which the type information is required. Cannot
326  *             be null.
327  */
328 extern ROUTERDllExport
329 const struct RTI_RoutingServiceStreamInfo *
330 RTI_RoutingServiceOutput_get_stream_info(RTI_RoutingServiceOutput *self);
331
332 /*e \ingroup RTI_RoutingServiceProcessorModule
333  *
334  * @brief Writes a list of data and information samples.
335  *
336  * @pre Output enabled
337  *
338  * @param self \b In. The Output where to write the samples. Cannot be null.
339  * @param sample_array \b In. Array of samples to write. Cannot be null.
340  * @param sample_info_array \b In. Array of information samples to write.
341  * @param array_length \b Inout. The length of the sample and info arrays.
342  *             Output is the number of samples written. Cannot be null.
343  *
```

```
344  * @return \c DDS_BOOLEAN_FALSE if the write operation was not successful.
345  *         \c DDS_BOOLEAN_TRUE otherwise.
346  */
347 extern ROUTERDllExport
348 DDS_Boolean RTI_RoutingServiceOutput_write(
349         RTI_RoutingServiceOutput *self,
350         const RTI_RoutingServiceSample *sample_array,
351         const RTI_RoutingServiceSampleInfo *sample_info_array,
352         int *array_length);
353
354 /*e \ingroup RTI_RoutingServiceProcessorModule
355  *
356  * @brief Writes a single sample, optionally with metatada indicated by an info
357  * sample
358  *
359  * @pre Output enabled
360  *
361  * @param self \b In. The Output where to write the samples. Cannot be null.
362  * @param data \b In. Data sample to be written. Cannot be null.
363  * @param info \b In. Associated information sample.
364  *
365  * @return \c DDS_BOOLEAN_FALSE if the sample could not be written,
366  *         \c DDS_BOOLEAN_TRUE otherwise.
367  */
368 extern ROUTERDllExport
369 DDS_Boolean RTI_RoutingServiceOutput_write_sample(
370         RTI_RoutingServiceOutput *self,
371         const RTI_RoutingServiceSample data,
372         const RTI_RoutingServiceSampleInfo info);
373
374 /*
375  * --- RTI_RoutingServiceRoute -------------------------------------------------
376  */
377
378 /*e \ingroup RTI_RoutingServiceProcessorModule
379  *
380  * @brief Returns the number of inputs associated with this Route.
381  *
382  */
383 extern ROUTERDllExport
384 int RTI_RoutingServiceRoute_get_input_count(RTI_RoutingServiceRoute *self);
385
386 /*e \ingroup RTI_RoutingServiceProcessorModule
387  *
388  * @brief Checks whether the Input with the specified index is enabled.
389  *
390  * @pre index >= 0 and index < RTI_RoutingServiceRoute_get_input_count(self)
391  *
392  * Note: This function will return \c DDS_BOOLEAN_FALSE both when all the parameters
393  * are correct and the input is disabled, or when any of the parameters are
394  * incorrect. It's the responsibility of the caller to ensure the parameters are
395  * valid, so that the returned value is, too.
396  *
397  * @param self \b In. The Route for which to check the Input status. Cannot be
398  *             null.
399  * @param index \b In. The index, starting at 0, of the Input for which the
400  *             status is required.
401  */
402 extern ROUTERDllExport
403 DDS_Boolean RTI_RoutingServiceRoute_is_input_enabled(
404         RTI_RoutingServiceRoute *self,
405         int index);
406
407 /*e \ingroup RTI_RoutingServiceProcessorModule
408  *
409  * @brief Returns the Input at the specified index.
410  *
411  * @pre index >= 0 and index < RTI_RoutingServiceRoute_get_input_count(self)
412  *
413  * Inputs are ordered according to the order of input declarations in the XML
414  * configuration. Then indexes are assigned increasingly starting at zero from
415  * the first declaration.
416  *
417  * @return The Input at the specified index, or \c NULL if the Input is not
418  *         enabled or the index is out of bounds.
419  */
420 extern ROUTERDllExport
421 RTI_RoutingServiceInput * RTI_RoutingServiceRoute_get_input_at(
422         RTI_RoutingServiceRoute *self,
423         int index);
424
```

```
425 /*e \ingroup RTI_RoutingServiceProcessorModule
426  *
427  * @brief Looks up the specified Input by its configuration name.
428  *
429  * The Input configuration name is the value of the name attribute specified
430  * within the &lt;input&gt; tag in the XML configuration.
431  *
432  * \b NOTE: The condition that the input must be enabled will be removed
433  * in a future version. This is being tracked with issue ID ROUTING-1131.
434  *
435  * @param self \b In. The Route for which to obtain the Input. Cannot be null.
436  * @param input_name \b In. Name of the input configuration. Cannot be null.
437  *
438  * @return The Input corresponding to the specified configuration name or
439  * NULL if it could not be found or if the input is not enabled.
440  */
441 extern ROUTERDllExport
442 RTI_RoutingServiceInput * RTI_RoutingServiceRoute_lookup_input_by_name(
443         RTI_RoutingServiceRoute *self,
444         const char *input_name);
445
446 /*e \ingroup RTI_RoutingServiceProcessorModule
447  *
448  * @brief Returns the number of outputs associated with this Route.
449  *
450  */
451 extern ROUTERDllExport
452 int RTI_RoutingServiceRoute_get_output_count(RTI_RoutingServiceRoute *self);
453
454 /*e \ingroup RTI_RoutingServiceProcessorModule
455  *
456  * @brief Checks whether the Output at the specified index is enabled.
457  *
458  * @pre index >= 0 and index < RTI_RoutingServiceRoute_get_output_count(self)
459  *
460  * Note: This function will return \c DDS_BOOLEAN_FALSE both when all the parameters
461  * are correct and the output is disabled, or when any of the parameters are
462  * incorrect. It's the responsibility of the caller to ensure the parameters are
463  * valid, so that the returned value is, too.
464  *
465  * @param self \b In. The Route for which to check the Output status. Cannot be
466  *             null.
467  * @param index \b In. The index, starting at 0, of the Output for which the
468  *             status is required.
469  */
470 extern ROUTERDllExport
471 DDS_Boolean RTI_RoutingServiceRoute_is_output_enabled(
472         RTI_RoutingServiceRoute *self,
473         int index);
474
475 /*e \ingroup RTI_RoutingServiceProcessorModule
476  *
477  * @brief Returns the Output at the specified index.
478  *
479  * @pre index >= 0 and index < RTI_RoutingServiceRoute_get_output_count(self)
480  *
481  * Outputs are ordered according to the order of output declarations in the XML
482  * configuration. Then indexes are assigned increasingly starting at zero from
483  * the first declaration.
484  *
485  * @return The Output at the specified index, or \c NULL if the Output is not
486  *         enabled or the index is out of bounds.
487  */
488 extern ROUTERDllExport
489 RTI_RoutingServiceOutput * RTI_RoutingServiceRoute_get_output_at(
490         RTI_RoutingServiceRoute *self,
491         int index);
492
493 /*e \ingroup RTI_RoutingServiceProcessorModule
494  *
495  * @brief Looks up the specified Output by its configuration name.
496  *
497  * The Output configuration name is the value of the name attribute specified
498  * within the &lt;output&gt; tag in the XML configuration.
499  *
500  * \b NOTE: The condition that the input must be enabled will be removed
501  * in a future version. This is being tracked with issue ID ROUTING-1131.
502  *
503  * @param self \b In. The Route for which to obtain the Output. Cannot be null.
504  * @param output_name \b In. Name of the output configuration. Cannot be null.
505  *
```

```
506  * @return The Output corresponding to the specified configuration name or
507  * NULL if it could not be found or if the output is not enabled.
508  */
509 extern ROUTERDllExport
510 RTI_RoutingServiceOutput * RTI_RoutingServiceRoute_lookup_output_by_name(
511         RTI_RoutingServiceRoute *self,
512         const char * output_name);
513
514 /*e \ingroup RTI_RoutingServiceProcessorModule
515  *
516  * @brief Sends a wakeup event to the specified Route.
517  * This operation can be safely called from any thread context.
518  */
519 extern ROUTERDllExport
520 void RTI_RoutingServiceRoute_wakeup_route(RTI_RoutingServiceRoute *route);
521
522 /*e \ingroup RTI_RoutingServiceProcessorModule
523  *
524  * @brief Get the current period of the periodic event for this Route.
525  *
526  * @param self \b In. The Route for which to obtain the period. Cannot be null.
527  * @param period \b Out. The current periodic event period.
528  */
529 extern ROUTERDllExport
530 void RTI_RoutingServiceRoute_get_period(
531         RTI_RoutingServiceRoute *self,
532         struct DDS_Duration_t *period);
533
534 /*e \ingroup RTI_RoutingServiceProcessorModule
535  *
536  * @brief Changes the periodic event period for this route.
537  *
538  * @pre not DDS_Duration_is_zero(period)
539  * @pre not DDS_Duration_is_infinite(period)
540  *
541  * This operation can be called many times within the same periodic event, in
542  * which only the last call will make effect.
543  *
544  * The operation has no effect if the periodic event is not enabled in the
545  * route.
546  *
547  * @param self \b In. The Route for which to set the period. Cannot be null.
548  * @param period \b In. New session period for periodic events.
549  *
550  */
551 extern ROUTERDllExport
552 void RTI_RoutingServiceRoute_set_period(
553         RTI_RoutingServiceRoute *self,
554         const struct DDS_Duration_t *period);
555
556 /*e \ingroup RTI_RoutingServiceProcessorModule
557  *
558  * @brief Returns the first enabled Input in this route.
559  *
560  * This function, along with RTI_RoutingServiceRoute_get_next_input(), provides
561  * iterator-based capabilities to iterate through a Route's enabled Inputs.
562  *
563  */
564 extern ROUTERDllExport
565 RTI_RoutingServiceInput * RTI_RoutingServiceRoute_get_first_input(
566         RTI_RoutingServiceRoute *self);
567
568 /*e \ingroup RTI_RoutingServiceProcessorModule
569  *
570  * @brief Returns the next enabled Input sibling to the specified input.
571  *
572  * This function, along with RTI_RoutingServiceRoute_get_first_input(), provides
573  * iterator-based capabilities to iterate through a Route's enabled Inputs.
574  */
575 extern ROUTERDllExport
576 RTI_RoutingServiceInput * RTI_RoutingServiceRoute_get_next_input(
577         RTI_RoutingServiceRoute *self,
578         RTI_RoutingServiceInput *prev_input);
579
580 /*e \ingroup RTI_RoutingServiceProcessorModule
581  *
582  * @brief Returns the first enabled output in this route
583  *
584  * This function, along with RTI_RoutingServiceRoute_get_next_output(), provides
585  * iterator-based capabilities to iterate through a Route's enabled Outputs.
586  */
```

```
587 extern ROUTERDllExport
588 RTI_RoutingServiceOutput * RTI_RoutingServiceRoute_get_first_output(
589         RTI_RoutingServiceRoute *self);
590
591 /*e \ingroup RTI_RoutingServiceProcessorModule
592  *
593  * @brief Returns the next enabled Output sibling to the specified output.
594  *
595  * This function, along with RTI_RoutingServiceRoute_get_first_output(), provides
596  * iterator-based capabilities to iterate through a Route's enabled Outputs.
597  */
598 extern ROUTERDllExport
599 RTI_RoutingServiceOutput * RTI_RoutingServiceRoute_get_next_output(
600         RTI_RoutingServiceRoute *self,
601         RTI_RoutingServiceOutput *prev_output);
602
603 /*e \ingroup RTI_RoutingServiceProcessorModule
604  *
605  * @brief Returns the fully-qualified name of this Route, derived from the
606  * XML configuration.
607  *
608  */
609 extern ROUTERDllExport
610 const char* RTI_RoutingServiceRoute_get_full_name(
611         RTI_RoutingServiceRoute *self);
612
613 /*****************************************************************************/
614 /* RouteEvent                                                                */
615 /*****************************************************************************/
616
617 /*e \ingroup RTI_RoutingServiceProcessorModule
618  *
619  * @brief Representation of the different event kinds triggered for a Route.
620  *
621  */
622 typedef enum {
623
624     RTI_ROUTING_SERVICE_ROUTE_EVENT_NONE = 0,
625     RTI_ROUTING_SERVICE_ROUTE_EVENT_INPUT_ENABLED,
626     RTI_ROUTING_SERVICE_ROUTE_EVENT_INPUT_DISABLED,
627     RTI_ROUTING_SERVICE_ROUTE_EVENT_OUTPUT_ENABLED,
628     RTI_ROUTING_SERVICE_ROUTE_EVENT_OUTPUT_DISABLED,
629     RTI_ROUTING_SERVICE_ROUTE_EVENT_ROUTE_STARTED,
630     RTI_ROUTING_SERVICE_ROUTE_EVENT_ROUTE_STOPPED,
631     RTI_ROUTING_SERVICE_ROUTE_EVENT_ROUTE_RUNNING,
632     RTI_ROUTING_SERVICE_ROUTE_EVENT_ROUTE_PAUSED,
633     RTI_ROUTING_SERVICE_ROUTE_EVENT_DATA_ON_INPUTS,
634     RTI_ROUTING_SERVICE_ROUTE_EVENT_PERIODIC_ACTION,
635     RTI_ROUTING_SERVICE_ROUTE_EVENT_OUTPUT_STREAM_REQUEST,
636     RTI_ROUTING_SERVICE_ROUTE_EVENT_WAKEUP,
637     RTI_RoutingServiceRouteEventKind_COUNT,
638     RTI_ROUTING_SERVICE_ROUTE_EVENT_ALL = 0xffffffff,
639
640 } RTI_RoutingServiceRouteEventKind;
641
642
643 /*i \ingroup RTI_RoutingServiceProcessorModule
644  *
645  */
646 struct RTI_RoutingServiceRouteEventImpl;
647
648 /*i \ingroup RTI_RoutingServiceProcessorModule
649  *
650  * @brief An event which indicates that an action occurred in a Route.
651  *
652  */
653 typedef struct RTI_RoutingServiceRouteEventImpl RTI_RoutingServiceRouteEvent;
654
655 /*e
656  * \ingroup RTI_RoutingServiceProcessorModule
657  *
658  * @brief Returns the kind of this RouteEvent.
659  *
660  * @return The Route that triggered the event, or
661  *         \c RTI_ROUTING_SERVICE_ROUTE_EVENT_NONE if the parameter passed is
662  *         \c null.
663  */
664 extern ROUTERDllExport
665 RTI_RoutingServiceRouteEventKind RTI_RoutingServiceRouteEvent_get_kind(
666         RTI_RoutingServiceRouteEvent *self);
667
```

```
668 /*e
669  * \ingroup RTI_RoutingServiceProcessorModule
670  *
671  * @brief Returns the Route that triggered the event.
672  *
673  * @return The Route that triggered the event, or \c NULL if the parameter
674  *         passed is null.
675  */
676 extern ROUTERDllExport
677 RTI_RoutingServiceRoute * RTI_RoutingServiceRouteEvent_get_route(
678         RTI_RoutingServiceRouteEvent *self);
679
680 /*e
681  * \ingroup RTI_RoutingServiceProcessorModule
682  *
683  * @brief Returns the data associated with the event.
684  *
685  * The event data depends on the kind of events. See documentation of individual
686  * events for the type of event data they provide.
687  *
688  * @return The data associated with the event, or \c NULL if the parameter
689  *         passed is null.
690  */
691 extern ROUTERDllExport
692 void * RTI_RoutingServiceRouteEvent_get_event_data(
693         RTI_RoutingServiceRouteEvent *self);
694
695 /*e
696  * \ingroup RTI_RoutingServiceProcessorModule
697  *
698  * @brief Returns the entity that is directly affected by the event.
699  *
700  * Each kind of event can be affected by different types of entities.
701  * See documentation of individual events for the type of affected entity.
702  *
703  * @return The entity affected by the event, or \c NULL if the parameter
704  *         passed is null.
705  */
706 extern ROUTERDllExport
707 void * RTI_RoutingServiceRouteEvent_get_affected_entity(
708         RTI_RoutingServiceRouteEvent *self);
709
710 /*
711 ------------------------------------------------------------------------------
712  Processor
713 ------------------------------------------------------------------------------
714 */
715
716 struct RTI_RoutingServiceProcessor;
717
718 /*e \ingroup RTI_RoutingServiceProcessorModule
719  *
720  * @brief Prototype of the function that processes an event that occurred in the
721  * Route where the Processor is installed.
722  *
723  * @param processor_data \b In. Implementation data of the Processor associated
724  *             with the Route that triggered the event.
725  * @param route_event \b In. Event that was triggered within the Route.
726  * @param env \b Out. Environment for error indications.
727  *
728 */
729 typedef void (*RTI_RoutingServiceProcessor_OnRouteEventFcn)(
730         void * processor_data,
731         RTI_RoutingServiceRouteEvent * route_event,
732         RTI_RoutingServiceEnvironment * env);
733
734
735 /*e \ingroup RTI_RoutingServiceProcessorModule
736  *
737  * @brief Prototype of the function that updates a Processor configuration.
738  *
739  * @param processor_data \b In. Implementation data.
740  * @param properties \b In. New configuration properties.
741  * @param env \b Out. Environment for error indications.
742  *
743  */
744 typedef void (*RTI_RoutingServiceProcessor_UpdateFcn)(
745         void * processor_data,
746         const struct RTI_RoutingServiceProperties * properties,
747         RTI_RoutingServiceEnvironment * env);
748
```

```
749
750 /*e \ingroup RTI_RoutingServiceProcessorModule
751  *
752  * @brief Main Processor class.
753  *
754  * A Route can notify a Processor about different events related to inputs
755  * and outputs.
756  *
757  * @see RTI_RoutingServiceRouteEvent
758  */
759 struct RTI_RoutingServiceProcessor{
760
761     /*e @brief Handles event processing (\b required). */
762     RTI_RoutingServiceProcessor_OnRouteEventFcn on_route_event;
763
764     /*e @brief Handles configuration changes in a Processor (\b optional). */
765     RTI_RoutingServiceProcessor_UpdateFcn update;
766
767     /*e
768      * @brief Implementation data. Provided by implementors, and passed back
769      *         in API calls.
770      */
771     void * processor_data;
772 };
773
774 /*e \ingroup RTI_RoutingServiceProcessorModule
775  *
776  * @brief Entry point for creating and deleting Processor objects.
777  *
778  */
779 struct RTI_RoutingServiceProcessorPlugin;
780
781
782 /*e \ingroup RTI_RoutingServiceProcessorModule
783  *
784  * @brief Prototype of the function that creates a ProcessorPlugin.
785  *
786  * @param properties \b In. Configuration properties for the ProcessorPlugin.
787  * @param env \b Out. Environment for error indications.
788  *
789  * @return New ProcessorPlugin instance if successful. Otherwise, \c NULL.
790  *
791  * @see RTI_RoutingServiceProcessorPlugin_DeleteFcn
792  */
793 typedef struct RTI_RoutingServiceProcessorPlugin *
794 (*RTI_RoutingServiceProcessorPlugin_CreateFcn)(
795         const struct RTI_RoutingServiceProperties * properties,
796         RTI_RoutingServiceEnvironment * env);
797
798 /*e \ingroup RTI_RoutingServiceProcessorModule
799  *
800  * @brief Prototype of the function that deletes a ProcessorPlugin.
801  *
802  * @param plugin \b In. ProcessorPlugin to be deleted.
803  * @param env \b Out. Environment for error indications.
804  *
805  * @see RTI_RoutingServiceProcessorPlugin_CreateFcn
806  */
807 typedef void (*RTI_RoutingServiceProcessorPlugin_DeleteFcn)(
808         struct RTI_RoutingServiceProcessorPlugin * plugin,
809         RTI_RoutingServiceEnvironment * env);
810
811
812 /*e \ingroup RTI_RoutingServiceProcessorModule
813  *
814  * @brief Prototype of the function that creates a route Processor.
815  *
816  * This function is called when a Processor is created.
817  *
818  * @param processor_plugin_data implementation data of the plugin from which the
819  *             Processor is created
820  * @param route Processor to be deleted.
821  * @param properties Configuration properties for the Processor.
822  *             These properties correspond to the properties specified
823  *             within the tag &lt;processor&gt;.
824  * @param env Environment for error indications.
825  *
826  * @return New processor if successful. Otherwise, error.
827  *
828  * @see RTI_RoutingServiceProcessorPlugin#DeleteProcessorFcn
829 */
```

```
830 typedef struct RTI_RoutingServiceProcessor *
831 (*RTI_RoutingServiceProcessorPlugin_CreateProcessorFcn)(
832         void * processor_plugin_data,
833         RTI_RoutingServiceRoute * route,
834         const struct RTI_RoutingServiceProperties * properties,
835         RTI_RoutingServiceEnvironment * env);
836
837 /*i \ingroup RTI_RoutingServiceProcessorModule
838  *
839  * @brief Prototype of the function that deletes a Processor.
840  *
841  * This function is called when a Processor is deleted
842  *
843  * @param processor_plugin_data implementation data of the plugin from which the
844  *            Processor is deleted.
845  * @param processor Processor to be deleted.
846  * @param env Environment for error indications.
847  *
848  * @see RTI_RoutingServiceProcessorPlugin_CreateProcessorFcn
849 */
850 typedef void (*RTI_RoutingServiceProcessorPlugin_DeleteProcessorFcn)(
851         void * processor_plugin_data,
852         struct RTI_RoutingServiceProcessor * processor,
853         RTI_RoutingServiceRoute * route,
854         RTI_RoutingServiceEnvironment * env);
855
856 /*e \ingroup RTI_RoutingServiceProcessorModule
857  *
858  * @brief ProcessorPlugin class.
859  *
860  * ProcessorPlugins are the entry points to the Processor API. They provide a
861  * factory for Processors.
862 */
863 struct RTI_RoutingServiceProcessorPlugin {
864     int _init;
865     struct RTI_RoutingServiceVersion _rs_version;
866
867     /*e
868      * @brief The version of this ProcessorPlugin.
869      */
870     struct RTI_RoutingServiceVersion plugin_version;
871
872     /*e
873      * @brief Handles the deletion of the ProcessorPlugin.
874      */
875     RTI_RoutingServiceProcessorPlugin_DeleteFcn plugin_delete;
876
877     /*e
878      * @brief Handles the creation of Processors.
879      */
880     RTI_RoutingServiceProcessorPlugin_CreateProcessorFcn create_processor;
881
882     /*e
883      * @brief Handles the deletion of Processors.
884      */
885     RTI_RoutingServiceProcessorPlugin_DeleteProcessorFcn delete_processor;
886
887     /*e
888      * @brief A placeholder for Processor implementors to keep a pointer to data
889      *        that may be needed by the implementation.
890      */
891     void * processor_plugin_data;
892 };
893
894 #define RTI_ROUTING_SERVICE_PROCESSOR_PLUGIN_INIT_NUMBER (8765432)
895
896 /*e \ingroup RTI_RoutingServiceProcessorModule
897  *
898  * @brief Utility macro to initialize a ProcessorPlugin object.
899  *
900  * This macro must be called to initialize the value returned by
901  * RTI_RoutingServiceProcessorPlugin_CreateFcn.
902  *
903  * @param plugin Pointer to the ProcessorPlugin object.
904  *
905  * @see RTI_RoutingServiceProcessorPlugin_CreateFcn
906 */
907 #define RTI_RoutingServiceProcessorPlugin_initialize(plugin)                  \
908 {                                                                             \
909     struct RTI_RoutingServiceVersion rsVersion = RTI_ROUTING_SERVICE_VERSION; \
910     struct RTI_RoutingServiceVersion processorVersion = {0,0,0,0};            \
```

```
911     (plugin)->_init = RTI_ROUTING_SERVICE_PROCESSOR_PLUGIN_INIT_NUMBER;       \
912     (plugin)->_rs_version = rsVersion;                                        \
913     (plugin)->plugin_version = processorVersion;                              \
914     (plugin)->plugin_delete = 0;                                              \
915     (plugin)->create_processor = 0;                                           \
916     (plugin)->delete_processor = 0;                                           \
917     (plugin)->processor_plugin_data = 0;                                      \
918 }
919
920 #ifdef __cplusplus
921 }       /* extern "C" */
922 #endif
923
924 #endif /* routingservice_processor_h */
```

7.7 routingservice_service.h File Reference

RTI Routing Library API.

```
#include "stdarg.h"
#include "routingservice/routingservice_dll.h"
#include "ndds/ndds_transport_c.h"
#include "routingservice/routingservice_adapter.h"
#include "routingservice/routingservice_transformation.h"
#include "routingservice/routingservice_processor.h"
#include "ndds/ndds_config_c.h"
#include "routingservice/ServiceAdmin.h"
#include "routingservice/ServiceAdminPlugin.h"
#include "routingservice/ServiceAdminSupport.h"
```

Data Structures

- struct **RTI_RoutingService**

  RTI Routing Service.
- struct **RTI_RoutingServiceTransportConfig**

  Association between a transport alias and its create function pointer.
- struct **RTI_RoutingServiceProperty**

  Configuration of RTI Routing Service.

Macros

- #define **RTI_RoutingServiceProperty_INITIALIZER**

  *The initial values for an **RTI_RoutingServiceProperty** (p. 91) instance.*

Functions

- struct **RTI_RoutingService** * **RTI_RoutingService_new** (const struct **RTI_RoutingServiceProperty** *property)

  Create a new RTI Routing Service instance.
- void **RTI_RoutingService_delete** (struct **RTI_RoutingService** *self)

  Stop and delete an RTI Routing Service instance.
- DDS_Boolean **RTI_RoutingService_start** (struct **RTI_RoutingService** *self)

  Start RTI Routing Service.
- DDS_Boolean **RTI_RoutingService_stop** (struct **RTI_RoutingService** *self)

  Stop RTI Routing Service.
- DDS_Boolean **RTI_RoutingService_attach_adapter_plugin** (struct **RTI_RoutingService** *self, void *adapter, const char *plugin_name)

  Attach an adapter to be used by routing service when it is started.
- DDS_Boolean **RTI_RoutingService_attach_transformation_plugin** (struct **RTI_RoutingService** *self, struct **RTI_RoutingServiceTransformationPlugin** *transformation_plugin, const char *plugin_name)

  Attach a transformation plugin to be used by Routing Service when it is started.
- DDS_Boolean **RTI_RoutingService_attach_processor_plugin** (struct **RTI_RoutingService** *self, void *processor_plugin, const char *plugin_name)

  Attach a processor to be used by Routing Service when it is started.
- DDS_Boolean **RTI_RoutingService_set_remote_shutdown_hook** (struct **RTI_RoutingService** *self, const struct RTI_RoutingServiceRemoteShutdownHook *shutdown_hook)

  Set the remote shutdown hook in this Routing Service instance.
- void **RTI_RoutingService_execute_command** (struct **RTI_RoutingService** *self, struct RTI_Service_Admin↩ _CommandReply **reply, const struct RTI_Service_Admin_CommandRequest *request)

  Executes an Administration command on this service.
- void **RTI_RoutingService_return_reply** (struct **RTI_RoutingService** *self, struct RTI_Service_Admin_↩ CommandReply *reply)

  Returns the reply object that is obtained as result of executing a command.
- DDS_Boolean **RTI_RoutingService_initialize_globals** (void)
- DDS_Boolean **RTI_RoutingService_finalize_globals** (void)
- const char * **RTI_RoutingService_get_build_number_string** (void)

  Return the build ID of this library.
- DDS_Boolean **RTI_RoutingService_is_started** (struct **RTI_RoutingService** *self)

  Query whether this Routing Service is currently started.
- void **RTI_RoutingServiceLogger_log** (NDDS_Config_LogLevel log_level, const char *format,...)

  Logs as message with the specified level.

Variables

- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_DEBUG**

  Verbosity level: exceptions + warnings + info + periodic + content.
- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_ALL**

  Verbosity level: exceptions + warnings + info + periodic.
- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_INFO**

  Verbosity level: exceptions + warnings + info.
- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_WARNINGS**

  Verbosity level: exceptions + warnings.

- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS**

  Verbosity level: exceptions.
- const int **RTI_ROUTING_SERVICE_LOG_VERBOSITY_SILENT**

  Verbosity level: silent.

7.7.1 Detailed Description

RTI Routing Library API.

7.8 routingservice_service.h

Go to the documentation of this file.

```
1 /*
2  * (c) Copyright, Real-Time Innovations, 2002-2024.
3  * All rights reserved.
4  *
5  * No duplications, whole or partial, manual or electronic, may be made
6  * without express written permission.  Any such copies, or
7  * revisions thereof, must display this notice unaltered.
8  * This code contains trade secrets of Real-Time Innovations, Inc.
9  */
10
11 #ifndef routingservice_service_h
12 #define routingservice_service_h
13
14 #include "stdarg.h"
15 #include "routingservice/routingservice_dll.h"
16 #include "ndds/ndds_transport_c.h"
17 #include "routingservice/routingservice_adapter.h"
18 #include "routingservice/routingservice_transformation.h"
19 #include "routingservice/routingservice_processor.h"
20 #include "ndds/ndds_config_c.h"
21
22 #include "routingservice/ServiceAdmin.h"
23 #include "routingservice/ServiceAdminPlugin.h"
24 #include "routingservice/ServiceAdminSupport.h"
25
26 #ifdef __cplusplus
27     extern "C" {
28 #endif
29
30 /*e \file
31   @brief RTI Routing Library API
32 */
33
34 /*e \defgroup RTI_RoutingServiceLibModule RTI Routing Library API
35  *
36  * \ingroup ROUTER
37  *
38  * @brief Prototype of the function that gets called upon reception of the
39  * shutdown command.
40  */
41 typedef void (*RTI_RoutingServiceRemoteShutdownHook_OnShutdownFunction)(
42         void *shutdownHookData);
43
44 /*e \defgroup RTI_RoutingServiceLibModule RTI Routing Library API
45  *
46  * \ingroup ROUTER
47  * @brief Definition of the interface that handles the remote shutdown.
48  */
49 struct RTI_RoutingServiceRemoteShutdownHook {
50     /* @brief a place holder for implementors */
51     void *shutdown_hook_data;
52     /* @brief handles the remote shutdown command */
53     RTI_RoutingServiceRemoteShutdownHook_OnShutdownFunction on_shutdown;
54 };
55
56 #define RTI_RoutingServiceRemoteShutdownHook_INITIALIZER {NULL, NULL}
```

```
57
58 /*e \defgroup RTI_RoutingServiceLibModule RTI Routing Library API
59  * \ingroup ROUTER
60  * @brief Routing Service can be deployed as a C library linked into your application on select
       architectures.
61  *
62  * This API allows you to create, configure and start RTI Routing Service instances from your application.
63  *
64  * The following code shows the typical use of the API:
65  *
66  * \code
67  *
68  * struct RTI_RoutingServiceProperty property = RTI_RoutingServiceProperty_INITIALIZER;
69  * struct RTI_RoutingService * service = NULL;
70  *
71  * property.cfg_file = "my_routing_service_cfg.xml";
72  * property.service_name = "my_routing_service";
73  * ...
74  *
75  * service = RTI_RoutingService_new(&property);
76  * if(service == NULL) {
77  *    printf("Error...");
78  *    return -1;
79  * }
80  *
81  * if(!RTI_RoutingService_start(service)) {
82  *    printf("Error...");
83  *    RTI_RoutingService_delete(service);
84  *    return -1;
85  * }
86  *
87  * while(keep_running) {
88  *     sleep();
89  *     ...
90  * }
91  *
92  * RTI_RoutingService_delete(service);
93  *
94  * return 0;
95  *
96  * \endcode
97  *
98  * Instead of a file, you can use XML strings to configure RTI Routing Service.
99  * See \ref RTI_RoutingServiceProperty for more information.
100  * <p>
101  * To build your application you need to link with the RTI Routing Service library
102  * in <b> &lt;RTI Routing Service home&gt;/bin/&lt;architecture&gt;/ </b>
103  *
104  * If you are using the C API on a Windows, Linux, MAC or INTEGRITY platform:
105  * See the example in
106  * <b> &lt;RTI Routing Service home&gt;/example/wrapperApp </b>
107  * Example makefiles and project files for several architecures are provided.
108  * Also see the README.txt file in the wrapperApp/src directory.
109  *
110  * ### Development Requirements
111  *
112  * |                 | Linux/macOS Systems                     | Windows Systems           |
113  * | --------------: | :---------------:                       | :-------------:           |
114  * | Shared Libraries| librtirsinfrastructure.so               | rtirsinfrastructure.dll   |
115  * | ^               | librtidlc.so                            | rtidlc.dll                |
116  * | ^               | librtiapputilsc.so                      | rtiapputilsc.dll          |
117  * | ^               | libnddsmetp.so                          | nddsmetp.dll              |
118  * | ^               | librticonnextmsgc.so                    | rticonnextmsgc.dll        |
119  * | ^               | libnddsc.so                             | nddsc.dll                 |
120  * | ^               | librtixml2.so                           | rtixml2.dll               |
121  * | ^               | libnddscore.so                          | nddscore.dll              |
122  * | Headers         | routingservice/routingservice_service.h ||
123  *
124  */
125
126 /*****************************************************************************/
127
128 #ifdef DOXYGEN_DOCUMENTATION_ONLY
129 /*e \ingroup RTI_RoutingServiceLibModule
130  *
131  * @brief RTI Routing Service
132  *
133  */
134 struct RTI_RoutingService {};
135 #endif
136
```

```
137 struct RTI_RoutingService;
138
139 /*****************************************************************************/
140
141 /*e \ingroup RTI_RoutingServiceLibModule
142  *
143  * @brief Association between a transport alias and its create function pointer
144  *
145  * Allows setting the entry point to load a statically linked transport and
146  * refer to it in the XML configuration through the participant_qos transport configuration.
147  *
148  * If a transport is configured in the XML configuration and its alias
149  * matches the one in this structure, it will be loaded
150  * by \ndds using the specified function pointer.
151  */
152 struct RTI_RoutingServiceTransportConfig {
153     /*e
154      * @brief An alias defined in the XML configuration to refer to a transport
155      */
156     char * alias;
157     /*e
158      * @brief Pointer to the function to load the transport
159      */
160     NDDS_Transport_create_plugin create_function;
161 };
162
163 /*e \ingroup RTI_RoutingServiceLibModule
164  *
165  * @brief Configuration of RTI Routing Service
166  *
167  * This structure must be initialized
168  * with \ref RTI_RoutingServiceProperty_INITIALIZER.
169  */
170 struct RTI_RoutingServiceProperty {
171     /*e
172      *
173      * @brief Path to an RTI Routing Service configuration file
174      *
175      * If not \c NULL, this file is loaded; otherwise,
176      * if \c cfg_strings is not \c NULL, that XML code is loaded.
177      *
178      * \default NULL.
179      */
180     char * cfg_file;
181
182     /*e
183      * @brief XML configuration represented as strings
184      *
185      * An array of strings that altogether make up an
186      * XML document to configure RTI Routing Service. This parameter is used
187      * only if \ref cfg_file is \c NULL.
188      *
189      * For example:
190      *
191      * \code
192      * int MY_ROUTING_SERVICE_CFG_SIZE = 3;
193      * const char * MY_ROUTING_SERVICE_CFG[MY_ROUTING_SERVICE_CFG_SIZE] =
194      *                           {"<dds><routing_service>",
195      *                            "<domain_route><participant><domai",
196      *                            "n_id>0...</dds>"};
197      *
198      * property.cfg_strings = MY_ROUTING_SERVICE_CFG;
199      * property.cfg_strings_count = MY_ROUTING_SERVICE_CFG_SIZE;
200      * \endcode
201      *
202      * The reason for using an array instead of one single string is to
203      * get around the limited size of literal strings in C.
204      * In general, if you create the XML string dynamically
205      * using one single string in the array,
206      * setting \ref cfg_strings_count to 1 is enough:
207      *
208      * \code
209      * property.cfg_strings = malloc(sizeof(char *));
210      * property.cfg_strings[0] = "<dds><routing_service>...</dds>";
211      * property.cfg_strings_count = 1;
212      * \endcode
213      *
214      * If your target system doesn't support a file system, you can use
215      * XML strings to configure the service. To ease this process, a
216      * utility is shipped to generate a C string array from a text file.
217      * You can find this utility in
```

```
218      * [RTI Routing Service installation directory]/resource/perl/cStringifyFile.pl
219      *
220      * \default NULL.
221      */
222     const char ** cfg_strings;
223
224     /*e
225      * @brief Size of the array \ref cfg_strings
226      *
227      * \default 0.
228      */
229     int cfg_strings_count;
230
231     /*e
232      * @brief The name of the Routing Service instance to run
233      *
234      * This is the name used to find the &lt;routing_service&gt;
235      * XML tag in the configuration file; the name that will
236      * be used to refer to this execution in remote administration and
237      * monitoring.
238      *
239      * \default NULL (use RTI_RoutingService)
240      */
241     char * service_name;
242
243     /*e
244      * @brief Assigns a name to the execution of the RTI Routing Service.
245      *
246      * Remote commands and status information will refer to the routing
247      * service using this name.
248      * In addition, the name of DomainParticipants created by RTI
249      * Routing Service will be based on this name.
250      *
251      * \default service_name if this value is different than NULL. Otherwise,
252      * "RTI_Routing_Service".
253      */
254     char * application_name;
255
256      /*e
257      *
258      * @brief Controls whether the service applies XSD validation to the loaded
259      * configuration.
260      *
261      * \default DDS_BOOLEAN_TRUE
262      */
263     DDS_Boolean enforce_xsd_validation;
264
265     /*e
266      * @brief The verbosity of the service
267      *
268      * Values:
269      *  <ul>
270      *  <li> \ref RTI_ROUTING_SERVICE_LOG_VERBOSITY_SILENT
271      *  <li> \ref RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS
272      *  <li> \ref RTI_ROUTING_SERVICE_LOG_VERBOSITY_WARNINGS
273      *  <li> \ref RTI_ROUTING_SERVICE_LOG_VERBOSITY_INFO
274      *  </ul>
275      *
276      * \default RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS
277      */
278     int service_verbosity;
279     /*e
280      * @brief The verbosity of \ndds core libraries
281      *
282      * Values:
283      *  <ul>
284      *  <li> \ref RTI_ROUTING_SERVICE_LOG_VERBOSITY_SILENT
285      *  <li> \ref RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS
286      *  <li> \ref RTI_ROUTING_SERVICE_LOG_VERBOSITY_WARNINGS
287      *  <li> \ref RTI_ROUTING_SERVICE_LOG_VERBOSITY_INFO
288      *  </ul>
289      *
290      * \default RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS
291      */
292     int dds_verbosity;
293
294     /*e
295      * @brief Value that is added to the domain IDs of the
296      *        domain routes in the XML configuration.
297      *
298      * By using this, an XML file can use relative domain IDs.
```

```
299      *
300      * \default 0
301      */
302     int domain_id_base;
303
304     /*e
305      * @brief This path is used to look for plug-in libraries.
306      *
307      * If the plug-in class libraries specified in the XML file don't
308      * contain a full path, RTI Routing Service looks for them in this
309      * directory. If not present, it will rely on the system library path.
310      *
311      * \default NULL (current directory)
312      */
313     char * plugin_search_path;
314
315     /*e
316      * @brief Set this to true to if you do not want RTI Routing Service enabled when
317      *        \ref RTI_RoutingService_start is called.
318      *
319      * RTI Routing Service can be enabled afterwards through
320      * remote administration.
321      *
322      * \default DDS_BOOLEAN_FALSE
323      */
324     DDS_Boolean dont_start_service;
325
326     /*e
327      * @brief Set this to true to enable remote administration
328      * or false to disable it.
329      *
330      * \default DDS_BOOLEAN_FALSE
331      */
332     DDS_Boolean enable_administration;
333
334     /*e
335      * @brief If \ref enable_administration is true, this is
336      * the domain ID to use for remote administration.
337      *
338      * Takes precedence over the XML configuration.
339      * If \ref enable_administration is false, this value is not used
340      * even if remote administration is enabled in the XML configuration.
341      *
342      * \default 0
343      */
344     int administration_domain_id;
345
346     /*e
347      * @brief Set it to true to enable remote monitoring
348      * or false to disable it.
349      *
350      * \default DDS_BOOLEAN_FALSE
351      */
352     DDS_Boolean enable_monitoring;
353
354     /*e
355      * @brief If \ref enable_monitoring is true, this is
356      * the domain ID to use for remote monitoring.
357      *
358      * Takes precedence over the XML configuration.
359      * If \ref enable_monitoring is false, this value is not used,
360      * even if remote monitoring is enabled in the XML configuration.
361      *
362      * \default 0
363      */
364     int monitoring_domain_id;
365
366     /*e
367      * @brief Clock value that is set as Routing Service internal clock.
368      * The clock must be defined as a comma-delimited list of clocks,
369      * in the order of preference.
370      * Valid clock names are "realtime," "system," and "monotonic." |br|
371      *
372      * By using this, Routing Service can use different clocks.
373      *
374      * \default realtime
375      */
376     char * internal_clock;
377
378     /*e
379      * @brief Set it to true to avoid loading the
```

```
380      * standard files usually loaded by RTI Routing Service.
381      *
382      * Only the configuration in \ref cfg_file or \ref cfg_strings will be loaded.
383      *
384      * \default DDS_BOOLEAN_FALSE
385      */
386     DDS_Boolean skip_default_files;
387
388     /*e
389      * @brief Set this to true to append the host name and process ID
390      * to the RTI Routing Service execution name.
391      *
392      * Used to get unique names for remote administration and monitoring.
393      *
394      * \default DDS_BOOLEAN_FALSE
395      */
396     DDS_Boolean identify_execution;
397
398     /*e
399      * @brief Transports to be loaded by \ndds.
400      *
401      * @see \ref registered_transports_count
402      *
403      * @pre Maximum 8 transports
404      */
405     struct RTI_RoutingServiceTransportConfig registered_transports[8];
406
407     /*e
408      * @brief Number of transports configured in \ref registered_transports.
409      *
410      * @pre Minimum 0 (no custom transport to load), maximum 8.
411      *
412      * \default 0
413      */
414     int registered_transports_count;
415
416     /*e
417      *
418      * @brief Path to an RTI Routing Service license file
419      *
420      * If not \c NULL, this file is checked for a valid license; otherwise,
421      * default location will be used. This parameter is only used if your
422      * installation requires a license file.
423      *
424      * \default NULL.
425      */
426     char * license_file_name;
427
428     /*e
429      * @brief Dictionary of user variables.
430      * The dictionary provides a parallel way to expand XML configuration
431      * variables in the form $(my_var), when they are not defined in the
432      * environment.
433      *
434      * \default empty
435      */
436     struct RTI_RoutingServiceProperties user_environment;
437
438 };
439
440 /*e \ingroup RTI_RoutingServiceLibModule
441 * @brief The initial values for an \ref RTI_RoutingServiceProperty instance
442 */
443 #define RTI_RoutingServiceProperty_INITIALIZER {\
444     NULL,                           /* cfg_file */                      \
445     NULL,                           /* cfg_strings */                   \
446     0,                              /* cfg_strings_count */             \
447     NULL,                           /* service_name */                  \
448     NULL,                           /* application_name */              \
449     DDS_BOOLEAN_TRUE,               /* enforce_xsd_validation */        \
450     RTI_LOG_BIT_FATAL_ERROR | RTI_LOG_BIT_EXCEPTION,   /* service_verbosity */ \
451     RTI_LOG_BIT_FATAL_ERROR | RTI_LOG_BIT_EXCEPTION,   /* dds_verbosity */     \
452     0,                              /* domain_id_base */                \
453     NULL,                           /* plugin_search_path */            \
454     RTI_FALSE,                      /* dont_start_service */            \
455     RTI_FALSE,                      /* enable_administration */         \
456     0,                              /* administration_domain_id */      \
457     RTI_FALSE,                      /* enable_monitoring */             \
458     0,                              /* monitoring_domain_id */          \
459     NULL,                           /* internal_clock */                \
460     RTI_FALSE,                      /* skip_default_files */            \
```

```
461     RTI_FALSE,                      /* identify_execution */            \
462     {                               /* registered_transports */         \
463         {NULL,NULL}, \
464         {NULL,NULL}, \
465         {NULL,NULL}, \
466         {NULL,NULL}, \
467         {NULL,NULL}, \
468         {NULL,NULL}, \
469         {NULL,NULL}, \
470         {NULL,NULL}  \
471     },\
472     0,                              /* registered_transport_count */    \
473     NULL,                           /* license_file_name */             \
474     {NULL, 0, 1}                    /* user_environment */              \
475 }
476
477 extern ROUTERDllVariable
478 const struct RTI_RoutingServiceProperty  RTI_ROUTING_SERVICE_PROPERTY_DEFAULT;
479
480 extern ROUTERDllExport
481 DDS_Boolean RTI_RoutingServiceProperty_initialize(
482         struct RTI_RoutingServiceProperty *self);
483
484 extern ROUTERDllExport
485 struct RTI_RoutingServiceProperty * RTI_RoutingServiceProperty_copy(
486         struct RTI_RoutingServiceProperty *to,
487         const struct RTI_RoutingServiceProperty *from);
488
489 extern ROUTERDllExport
490 void RTI_RoutingServiceProperty_finalize(
491         struct RTI_RoutingServiceProperty *self);
492
493 /*e \ingroup RTI_RoutingServiceLibModule
494 * @brief Verbosity level: exceptions + warnings + info + periodic + content
495 */
496 extern ROUTERDllVariable
497 const int RTI_ROUTING_SERVICE_LOG_VERBOSITY_DEBUG;
498
499 /*e \ingroup RTI_RoutingServiceLibModule
500 * @brief Verbosity level: exceptions + warnings + info + periodic
501 */
502 extern ROUTERDllVariable
503 const int  RTI_ROUTING_SERVICE_LOG_VERBOSITY_ALL;
504
505 /*e \ingroup RTI_RoutingServiceLibModule
506 * @brief Verbosity level: exceptions + warnings + info
507 */
508 extern ROUTERDllVariable
509 const int  RTI_ROUTING_SERVICE_LOG_VERBOSITY_INFO;
510
511 /*e \ingroup RTI_RoutingServiceLibModule
512 * @brief Verbosity level: exceptions + warnings
513 */
514 extern ROUTERDllVariable
515 const int  RTI_ROUTING_SERVICE_LOG_VERBOSITY_WARNINGS;
516
517 /*e \ingroup RTI_RoutingServiceLibModule
518 * @brief Verbosity level: exceptions
519 */
520 extern ROUTERDllVariable
521 const int  RTI_ROUTING_SERVICE_LOG_VERBOSITY_EXCEPTIONS;
522
523 /*e \ingroup RTI_RoutingServiceLibModule
524 * @brief Verbosity level: silent
525 */
526 extern ROUTERDllVariable
527 const int  RTI_ROUTING_SERVICE_LOG_VERBOSITY_SILENT;
528
529 /*************************************************************************/
530
531 /*e \ingroup RTI_RoutingServiceLibModule
532  *
533  * @brief Create a new RTI Routing Service instance
534  *
535  * @param property The properties to configure RTI Routing Service.
536  *                 This parameter is copied internally, so the user is responsible
537  *                 for releasing any memory allocated inside this structure.
538  *
539  *  @mtsafety On non-Linux, non-Windows systems (i.e. VxWorks):
540  *  UNSAFE for multiple threads to simultaneously make the FIRST
541  *  call to RTI_RoutingService_new(). Subsequent calls are thread safe.
```

```
542  *  On Windows and Linux systems, these calls are thread-safe.
543  */
544 extern ROUTERDllExport
545 struct RTI_RoutingService * RTI_RoutingService_new(
546         const struct RTI_RoutingServiceProperty * property);
547
548 /*e \ingroup RTI_RoutingServiceLibModule
549  *
550  * @brief Stop and delete an RTI Routing Service instance.
551  *
552  * @see \ref RTI_RoutingService_stop
553  *
554  * @param self An \ref RTI_RoutingService instance created with
555  * \ref RTI_RoutingService_new
556  *
557  * @mtsafety
558  * This method is not thread-safe. Calling this method from different threads
559  * for the same Routing Service instance may result in undefined behavior.
560  *
561  *
562  */
563 extern ROUTERDllExport
564 void RTI_RoutingService_delete(struct RTI_RoutingService * self);
565
566 /*e \ingroup RTI_RoutingServiceLibModule
567  *
568  * @brief Start RTI Routing Service.
569  *
570  * This is a non-blocking operation. RTI Routing Service will create its own set
571  * of threads to perform its tasks.
572  *
573  * @param self An \ref RTI_RoutingService instance created with
574  * \ref RTI_RoutingService_new
575  *
576  */
577 extern ROUTERDllExport
578 DDS_Boolean RTI_RoutingService_start(struct RTI_RoutingService * self);
579
580 /*e \ingroup RTI_RoutingServiceLibModule
581  *
582  * @brief Stop RTI Routing Service.
583  *
584  * This function won't return the execution control until the instance is
585  * fully stopped.
586  *
587  * @param self An \ref RTI_RoutingService instance created with
588  * \ref RTI_RoutingService_new
589  *
590  * @mtsafety
591  * This method is not thread-safe. Calling this method from different threads
592  * for the same Routing Service instance may result in undefined behavior.
593  *
594  */
595 extern ROUTERDllExport
596 DDS_Boolean RTI_RoutingService_stop(struct RTI_RoutingService * self);
597
598 /*e \ingroup RTI_RoutingServiceLibModule
599  *
600  * @brief Attach an adapter to be used by routing service when it is started.
601  *
602  * By using this function, an adapter can be statically compiled, created
603  * in your application and have routing service load it,
604  * instead of registering a shared library and a create function
605  * in the configuration. The name passed into this function is the name that has
606  * to be used in the configuration to instantiate connections from the plugin.
607  *
608  * Example:
609  *
610  * \code
611  *
612  * service = RTI_RoutingService_new(&property);
613  * myAdapter = MyAdapter_create();
614  *
615  * RTI_RoutingService_attach_adapter_plugin(service, myAdapter, "MyAdapter");
616  *
617  * RTI_RoutingService_start(service);
618  * ...
619  *
620  * \endcode
621  *
622  * And our configuration would look like this:
```

```
623  *
624  * \code
625  * <dds>
626  *  <!-- No need to register the plugin in
627  *       <plugin_library><adapter_plugin>
628  *  -->
629  *  <routing_service name="example">
630  *   <domain_route name="myadapter_to_dds">
631  *
632  *     <connection name="MyConnection" plugin_name="MyAdapter">
633  *        ...
634  *     </connection>
635  *
636  *     ...
637  *   </domain_route>
638  *  </routing_service>
639  * </dds>
640  *
641  * \endcode
642  *
643  * This function can be called as many times as desired to attach several plugins.
644  *
645  * Note: The RTI Routing Service Adapter SDK is required.
646  *
647  * @pre Routing Service must not be started.
648  *
649  * @param self An \ref RTI_RoutingService instance not started yet (or stopped)
650  * @param adapter The adapter plugin to be attached
651  * @param plugin_name The name used for this plugin in the &lt;connection&gt;
652  *                    tags in the configuration
653  *
654  */
655 extern ROUTERDllExport
656 DDS_Boolean RTI_RoutingService_attach_adapter_plugin(
657     struct RTI_RoutingService * self,
658     void * adapter,
659     const char * plugin_name);
660
661 /*e \ingroup RTI_RoutingServiceLibModule
662  *
663  * @brief Attach a transformation plugin to be used by Routing Service when
664  * it is started.
665  *
666  * @see RTI_RoutingService_attach_adapter_plugin
667  */
668 extern ROUTERDllExport
669 DDS_Boolean RTI_RoutingService_attach_transformation_plugin(
670     struct RTI_RoutingService *self,
671     struct RTI_RoutingServiceTransformationPlugin *transformation_plugin,
672     const char *plugin_name);
673
681 extern ROUTERDllExport
682 DDS_Boolean RTI_RoutingService_attach_processor_plugin(
683         struct RTI_RoutingService * self,
684         void * processor_plugin,
685         const char * plugin_name);
686
696 extern ROUTERDllExport
697 DDS_Boolean RTI_RoutingService_set_remote_shutdown_hook(
698         struct RTI_RoutingService * self,
699         const struct RTI_RoutingServiceRemoteShutdownHook * shutdown_hook);
700
718 extern ROUTERDllExport
719 void RTI_RoutingService_execute_command(
720         struct RTI_RoutingService * self,
721         struct RTI_Service_Admin_CommandReply ** reply,
722         const struct RTI_Service_Admin_CommandRequest * request);
723
735 extern ROUTERDllExport
736 void RTI_RoutingService_return_reply(
737         struct RTI_RoutingService * self,
738         struct RTI_Service_Admin_CommandReply * reply);
739
740
748 extern ROUTERDllExport
749 DDS_Boolean RTI_RoutingService_initialize_globals(void);
750
758 extern ROUTERDllExport
759 DDS_Boolean RTI_RoutingService_finalize_globals(void);
760
768 extern ROUTERDllExport
```

```
769 const char* RTI_RoutingService_get_build_number_string(void);
770
775 extern ROUTERDllExport
776 DDS_Boolean RTI_RoutingService_is_started(
777         struct RTI_RoutingService * self);
778
779 /*e \ingroup RTI_RoutingServiceInfrastructureModule
780  *
781  * @brief Logs as message with the specified level
782  *
783  * The message is specified with a format and a format parameter, in a similar
784  * fashion to the standard C printf() operation.
785  *
786  * The generated log will be part of logging stream of the running
787  * RoutingService, if the \p log_level is part of the configured verbosity.
788  * The result log message may include additional information according to the
789  * Connext logging configuration, such as Advlog Context, thread ID, line number,
790  * etc. Additionally, the result log message will contain a newline character
791  * at the end, so the format does not need to contain it.
792  *
793  * @param[in] log_level Log level associated to the message
794  * @param[in] format message format specification
795  * @param[in] ... variable-length argument (stdarg)
796  */
797 extern ROUTERDllExport
798 void RTI_RoutingServiceLogger_log(
799         NDDS_Config_LogLevel log_level,
800         const char *format,
801         ...);
802
803 #ifdef __cplusplus
804 }       /* extern "C" */
805 #endif
806
807 #endif /* routingservice_service_h */
```

7.9 routingservice_transformation.h File Reference

RTI Routing Service Transformation API.

```
#include "routingservice/routingservice_infrastructure.h"
```

Data Structures

- struct **RTI_RoutingServiceTransformationPlugin**

  Transformation plugin.

Macros

- #define **RTI_RoutingServiceTransformationPlugin_initialize**(transf)

  Initializes the plugin structure.

Typedefs

- typedef void * **RTI_RoutingServiceTransformation**

  Transformation.
- typedef struct **RTI_RoutingServiceTransformationPlugin** *(* **RTI_RoutingServiceTransformationPlugin_CreateFcn**) (const struct **RTI_RoutingServiceProperties** *properties, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that creates a transformation plugin.
- typedef void(* **RTI_RoutingServiceTransformationPlugin_DeleteFcn**) (struct **RTI_RoutingServiceTransformationPlugin** *plugin, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that deletes a transformation plugin.
- typedef **RTI_RoutingServiceTransformation**(* **RTI_RoutingServiceTransformationPlugin_CreateTransformationFcn**) (struct **RTI_RoutingServiceTransformationPlugin** *plugin, const struct **RTI_RoutingServiceTypeInfo** *input_type_info, const struct **RTI_RoutingServiceTypeInfo** *output_type_info, const struct **RTI_RoutingServiceProperties** *properties, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that creates a route transformation.
- typedef void(* **RTI_RoutingServiceTransformationPlugin_DeleteTransformationFcn**) (struct **RTI_RoutingServiceTransformationPlugin** *plugin, **RTI_RoutingServiceTransformation** transformation, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that deletes a transformation.
- typedef void(* **RTI_RoutingServiceTransformation_TransformFcn**) (**RTI_RoutingServiceTransformation** transformation, **RTI_RoutingServiceSample** **out_sample_lst, **RTI_RoutingServiceSampleInfo** **out_info_lst, int *out_count, **RTI_RoutingServiceSample** *in_sample_lst, **RTI_RoutingServiceSampleInfo** *in_info_lst, int in_count, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that transforms a sequence of input samples into a sequence of output samples.
- typedef void(* **RTI_RoutingServiceTransformation_ReturnLoanFcn**) (**RTI_RoutingServiceTransformation** transformation, **RTI_RoutingServiceSample** *sample_lst, **RTI_RoutingServiceSampleInfo** *info_lst, int count, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that returns the loan on the output samples and infos.
- typedef void(* **RTI_RoutingServiceTransformation_UpdateFcn**) (**RTI_RoutingServiceTransformation** transformation, const struct **RTI_RoutingServiceProperties** *properties, **RTI_RoutingServiceEnvironment** *env)

  Prototype of the function that updates a transformation configuration.

7.9.1 Detailed Description

RTI Routing Service Transformation API.

7.10 routingservice_transformation.h

Go to the documentation of this file.

```
1 /*
2  * (c) Copyright, Real-Time Innovations, 2002-2024.
3  * All rights reserved.
4  *
5  * No duplications, whole or partial, manual or electronic, may be made
6  * without express written permission.  Any such copies, or
7  * revisions thereof, must display this notice unaltered.
8  * This code contains trade secrets of Real-Time Innovations, Inc.
9  */
10
11 #ifndef routingservice_transformation_h
```

```
12 #define routingservice_transformation_h
13
14 #include "routingservice/routingservice_infrastructure.h"
15
16 #ifdef __cplusplus
17     extern "C" {
18 #endif
19
20 /*e \file
21   @brief RTI Routing Service Transformation API
22 */
23
24 /*e \defgroup RTI_RoutingServiceTransformationModule RTI Routing Service Transformation API
25  \ingroup ROUTER
26  @brief This module describes the Transformation API.
27
28  An \product route transforms the incoming data using a \em transformation, which is an
29  object created by a transformation plugin.<br>
30
31  Transformation plugins implement the transformation API described in this module and
32  must be provided as shared libraries that RTI \product will load dynamically.
33
34  To register a transformation plugin with \product, you must use the tag
35  &lt;transformation_plugin&gt; within &lt;transformation_library&gt;. For example:
36
37  \verbatim
38  <dds>
39      ...
40      <transformation_library name="MyTransfLib">
41          <transformation_plugin name="MyTransfPlugin">
42              <dll>mytransformation</dll>
43              <create_function>
44                 MyTransfPlugin_create
45             </create_function>
46         </transformation_plugin>
47         ...
48      </transformation_library>
49      ...
50      <routing_service>
51      ...
52      </routing_service>
53      ...
54  </dds>
55  \endverbatim
56
57  Once a transformation plugin is registered, an Output can use it to create a data transformation.
58  For example:
59
60  \verbatim
61  <topic_route name="SquareSwitchCoord">
62      <input participant="1">
63          <topic_name>Square</topic_name>
64          <registered_type_name>ShapeType</registered_type_name>
65      </input>
66      <output>
67          <topic_name>Square</topic_name>
68          <registered_type_name>ShapeType</registered_type_name>
69      </output>
70      <transformation plugin_name="MyTransfLib::MyTransPlugin">
71          <property>
72              <value>
73                  <element>
74                      <name>X</name>
75                      <value>Y</value>
76                  </element>
77                  <element>
78                      <name>Y</name>
79                      <value>X</value>
80                  </element>
81              </value>
82          </property>
83      </transformation>
84  </topic_route>
85  \endverbatim
86
87  For additional information on configuring transformations, see the \ref_url_routing_service_users_manual.
88
89  \section Requirements Development Requirements
90
91  <table border="1" cellspacing="1">
92     <tr>
```

```
93          <td></td>
94          <td>Linux or macOS Systems</td>
95          <td>Windows Systems</td>
96      </tr>
97      <tr>
98          <td><b>Shared Library</b></td>
99          <td>librtirsinfrastructure.so<br>
100             <br>
101             If the transformation must work with adapters providing DDS_DynamicData or DDS_SampleInfo (such
        as the built-in DDS adapter):<br>
102             libnddsc.so and libnddscore.so
103         </td>
104         <td>rtirsinfrastructure.dll<br>
105             <br>
106             If the transformation must work with adapters providing DDS_DynamicData or DDS_SampleInfo (such
        as the built-in DDS adapter):<br>
107             nddsc.dll and nddscore.dll
108         </td>
109     </tr>
110     <tr>
111         <td><b>Header</b></td>
112         <td colspan="2">
113             routingservice_transformation.h<br>
114             <br>
115             If the transformation must work with adapters providing DDS_DynamicData or DDS_SampleInfo (such
        as the built-in DDS adapter):
116             ndds_c.h
117         </td>
118     </tr>
119 </table>
120 */
121 struct RTI_RoutingServiceTransformationPlugin;
122
123 /*e \ingroup RTI_RoutingServiceTransformationModule
124   \brief Transformation.
125
126   A route can transform the incoming data using transformation objects.
127
128   The transformation objects are created by transformation plugins.
129 */
130 typedef void * RTI_RoutingServiceTransformation;
131
132 /*e \ingroup RTI_RoutingServiceTransformationModule
133  \brief Prototype of the function that creates a transformation plugin.
134
135  The name of the function that implements this prototype
136  must be provided to \product using the tag &lt;create_function&gt;
137  when the transformation plugin is registered.
138
139  For example:
140
141  \verbatim
142  <plugin_library name="MyTransfLib">
143      <transformation_plugin name="MyTransfPlugin">
144          <dll>mytransformation</dll>
145          <create_function>
146             MyTransfPlugin_create
147          </create_function>
148      </transformation_plugin>
149      ...
150  </plugin_library>
151  \endverbatim
152
153  This is the only function that is not part of
154  \ref RTI_RoutingServiceTransformationPlugin.
155
156  <b>Required:</b> yes
157
158  @param properties \rs_st_in Configuration properties for the transformation.
159
160  @param env \rs_st_inout Environment for error indications.
161
162  @return New plugin instance if successful. Otherwise, NULL.
163
164  @see \ref RTI_RoutingServiceTransformationPlugin_DeleteFcn
165 */
166 typedef struct RTI_RoutingServiceTransformationPlugin *(
167         *RTI_RoutingServiceTransformationPlugin_CreateFcn)(
168         const struct RTI_RoutingServiceProperties *properties,
169         RTI_RoutingServiceEnvironment *env);
170
```

```
171 /*e \ingroup RTI_RoutingServiceTransformationModule
172  \brief Prototype of the function that deletes a transformation plugin.
173
174  Transformation plugins are deleted when \product is closed.
175
176  <b>Required:</b> yes
177
178  @param plugin \rs_st_in Transformation plugin to be deleted.
179  @param env \rs_st_inout Environment for error indications.
180
181  @see \ref RTI_RoutingServiceTransformationPlugin_CreateFcn
182 */
183 typedef void (*RTI_RoutingServiceTransformationPlugin_DeleteFcn)(
184         struct RTI_RoutingServiceTransformationPlugin *plugin,
185         RTI_RoutingServiceEnvironment *env);
186
187 /*e \ingroup RTI_RoutingServiceTransformationModule
188  \brief Prototype of the function that creates a route transformation.
189
190  This function is called when the route containing the transformation
191  is ready to forward data.
192
193  The format associated with the input and output types depends on
194  the format provided by the route adapters.
195
196  For the built-in DDS adapter, the format of the types is DDS_TypeCode.
197
198  <b>Required:</b> yes
199
200  @param plugin \rs_st_in Transformation plugin that will be used to create the transformation.
201  @param input_type_info \rs_st_in Type information associated with the input samples.
202  @param output_type_info \rs_st_in Type information associated with the output samples.
203  @param properties \rs_st_in Configuration properties for the transformation.
204  These properties corresponds to the properties specified within the tag &lt;transformation&gt;.
205  @param env \rs_st_inout Environment for error indications.
206
207  @return New transformation if successful. Otherwise, error.
208
209  @see \ref RTI_RoutingServiceTransformationPlugin_DeleteTransformationFcn
210 */
211 typedef RTI_RoutingServiceTransformation (
212         *RTI_RoutingServiceTransformationPlugin_CreateTransformationFcn)(
213         struct RTI_RoutingServiceTransformationPlugin *plugin,
214         const struct RTI_RoutingServiceTypeInfo *input_type_info,
215         const struct RTI_RoutingServiceTypeInfo *output_type_info,
216         const struct RTI_RoutingServiceProperties *properties,
217         RTI_RoutingServiceEnvironment *env);
218
219 /*e \ingroup RTI_RoutingServiceTransformationModule
220  \brief Prototype of the function that deletes a transformation.
221
222  This function is called when the route containing the transformation is disabled.
223
224  @param plugin \rs_st_in Transformation plugin that will be used to delete the transformation.
225  @param transformation \rs_st_in Transformation to be deleted.
226  @param env \rs_st_inout Environment for error indications.
227
228  @see \ref RTI_RoutingServiceTransformationPlugin_CreateTransformationFcn
229 */
230 typedef void (*RTI_RoutingServiceTransformationPlugin_DeleteTransformationFcn)(
231         struct RTI_RoutingServiceTransformationPlugin *plugin,
232         RTI_RoutingServiceTransformation transformation,
233         RTI_RoutingServiceEnvironment *env);
234
235 /*e \ingroup RTI_RoutingServiceTransformationModule
236
237  \brief Prototype of the function that transforms a sequence of input
238  samples into a sequence of output samples.
239
240  When \product is done using the output samples, it will 'return the loan'
241  to the transformation by calling \ref RTI_RoutingServiceTransformation_ReturnLoanFcn.
242
243  The number of output samples can be different than the number of input samples.
244
245  The memory used by the output samples and sample infos is managed by the transformation.
246  The transformation must allocate the memory for the output samples and sample infos
247  and free it when \product calls \ref RTI_RoutingServiceTransformation_ReturnLoanFcn.
248
249  The format associated with the input and output samples and sample infos depends on
250  the format provided and consumed by the route StreamReader and StreamWriter.
251
```

```
252  For the built-in DDS adapter, the format of the samples is DDS_DynamicData and the format
253  of the sample info is DDS_SampleInfo.
254
255  <b>Required:</b> yes
256
257  @param transformation \rs_st_in Transformation that will transform the samples.
258  @param out_sample_lst \rs_st_out Array that will hold the output samples.
259  This array will be provided by the transformation. *Note*: A transformation
260  cannot return the input sample array as the output sample array;
261  otherwise \product will fail. In general, the memory used for the output samples
262  should be managed by the transformation.
263  @param out_info_lst \rs_st_out Array that will hold the output sample infos.
264  This array will be provided by the transformation. *Note*: A transformation
265  cannot return the input sample info array as the output info array;
266  otherwise \product will fail. In general, the memory used for the
267  output sample infos should be managed by the transformation and freed when
268  \product calls \ref RTI_RoutingServiceTransformation_ReturnLoanFcn.
269  @param out_count \rs_st_out Number of output samples. The value must be greater
270  than or equal to zero.
271  @param in_sample_lst \rs_st_in Array of input samples.
272  @param in_info_lst \rs_st_in Array of input sample infos.
273  @param in_count \rs_st_in Number of input samples.
274  @param env \rs_st_inout Environment for error indications.
275
276  @see \ref RTI_RoutingServiceTransformation_ReturnLoanFcn
277 */
278 typedef void (*RTI_RoutingServiceTransformation_TransformFcn)(
279         RTI_RoutingServiceTransformation transformation,
280         RTI_RoutingServiceSample **out_sample_lst,
281         RTI_RoutingServiceSampleInfo **out_info_lst,
282         int *out_count,
283         RTI_RoutingServiceSample *in_sample_lst,
284         RTI_RoutingServiceSampleInfo *in_info_lst,
285         int in_count,
286         RTI_RoutingServiceEnvironment *env);
287
288 /*e \ingroup RTI_RoutingServiceTransformationModule
289  \brief Prototype of the function that returns the loan on the output samples
290  and infos.
291
292  This function is called by \product to indicate that it is done accessing the
293  array of data samples obtained by an earlier invocation of
294  \ref RTI_RoutingServiceTransformation_TransformFcn.
295
296  The memory used by the output samples and sample infos is managed by the
297  transformation and should be freed in this call if it was allocated when
298  \product calls \ref RTI_RoutingServiceTransformation_ReturnLoanFcn.
299
300  <b>Required:</b> yes
301
302  @param transformation \rs_st_in Transformation that owns the samples and sample infos.
303  @param sample_lst \rs_st_in Array of samples.
304  @param info_lst \rs_st_in Array of sample infos.
305  @param count \rs_st_in Number of samples in the array.
306  @param env \rs_st_inout Environment for error indications.
307 */
308 typedef void (*RTI_RoutingServiceTransformation_ReturnLoanFcn)(
309         RTI_RoutingServiceTransformation transformation,
310         RTI_RoutingServiceSample *sample_lst,
311         RTI_RoutingServiceSampleInfo *info_lst,
312         int count,
313         RTI_RoutingServiceEnvironment *env);
314
315 /*e \ingroup RTI_RoutingServiceTransformationModule
316  \brief Prototype of the function that updates a transformation configuration.
317
318  @param transformation \rs_st_in Transformation.
319  @param properties \rs_st_in New configuration properties.
320  @param env \rs_st_inout Environment for error indications.
321 */
322 typedef void (*RTI_RoutingServiceTransformation_UpdateFcn)(
323         RTI_RoutingServiceTransformation transformation,
324         const struct RTI_RoutingServiceProperties *properties,
325         RTI_RoutingServiceEnvironment *env);
326
327 /*e \ingroup RTI_RoutingServiceTransformationModule
328   \brief Transformation plugin.
329
330   This structure contains the plugin implementation as a set of function pointers.
331 */
332 struct RTI_RoutingServiceTransformationPlugin {
```

```
333     int _init;
334     struct RTI_RoutingServiceVersion _rs_version;
335
336     /*e \brief The version of this transformation plugin */
337     struct RTI_RoutingServiceVersion plugin_version;
338
339     /*e \brief Handles the deletion of the transformation plugin. */
340     RTI_RoutingServiceTransformationPlugin_DeleteFcn
341         transformation_plugin_delete;
342     /*e \brief Handles the creation of transformations. */
343     RTI_RoutingServiceTransformationPlugin_CreateTransformationFcn
344         transformation_plugin_create_transformation;
345     /*e \brief Handles the deletion of transformations. */
346     RTI_RoutingServiceTransformationPlugin_DeleteTransformationFcn
347         transformation_plugin_delete_transformation;
348
349     /*e \brief Handles the transformation of samples. */
350     RTI_RoutingServiceTransformation_TransformFcn transformation_transform;
351     /*e \brief Handles the return of the loan taken on the transformed samples. */
352     RTI_RoutingServiceTransformation_ReturnLoanFcn transformation_return_loan;
353     /*e \brief Handles the update of the transformation configuration. */
354     RTI_RoutingServiceTransformation_UpdateFcn transformation_update;
355
356     /*e @brief A place for transformation implementors to keep a pointer to data that
357         may be needed by the implementation. */
358     void * user_object;
359 };
360
361
362
363 #define RTI_ROUTING_SERVICE_TRANSFORMATION_PLUGIN_INIT_NUMBER (9876543)
364 /*e \ingroup RTI_RoutingServiceTransformationModule
365  \hideinitializer
366  @brief Initializes the plugin structure.
367
368  This macro must be called to initialize the
369  return value of RTI_RoutingServiceTransformationPlugin_CreateFcn
370
371  @param transf Pointer to the transformation plugin structure
372
373  @see \ref RTI_RoutingServiceTransformationPlugin_CreateFcn
374 */
375 #define RTI_RoutingServiceTransformationPlugin_initialize(transf)             \
376 {                                                                             \
377     struct RTI_RoutingServiceVersion rsVersion = RTI_ROUTING_SERVICE_VERSION; \
378     struct RTI_RoutingServiceVersion transfVersion = {0,0,0,0};               \
379     (transf)->_init = RTI_ROUTING_SERVICE_TRANSFORMATION_PLUGIN_INIT_NUMBER;  \
380     (transf)->_rs_version = rsVersion;                                        \
381     (transf)->plugin_version = transfVersion;                                 \
382     (transf)->transformation_plugin_delete = 0;                               \
383     (transf)->transformation_plugin_create_transformation = 0;                \
384     (transf)->transformation_plugin_delete_transformation = 0;                \
385     (transf)->transformation_transform = 0;                                   \
386     (transf)->transformation_return_loan = 0;                                 \
387     (transf)->transformation_update = 0;                                      \
388     (transf)->user_object = 0;                                                \
389 }
390
391 #ifdef __cplusplus
392 }       /* extern "C" */
393 #endif
394
395 #endif /* routingservice_transformation_h */
```

Index