

Purpose of this example

This example demonstrates the automatic generation of ASN.1 encoders and decoders for messages exchanged between the SDL system and the environment.

What it does

Generate the code with the profile corresponding to your platform. It generates an executable called `sys` in `cgc_<platform_name>` directory. The `main()` of the executable is located `sys_main.c` file. It will first launch the code generated out of the SDL system (`SDL_main()`), then it will send pre-encoded ASN.1 strings for:

- one `begin_line` message,
- 2 `add_point` messages,
- one `add_point_by_coords` message,
- one `end_line` message

to the SDL system with the use of the `RTDS_ASN1_MESSAGE_TO_SYSTEM` macro.

The SDL system will reply via a message that has a complex parameter: a sequence of `PointType` which is a sequence having fields of various types (`INTEGER`, `IA5String`, `SET OF IA5String`). The external C code defines the `RTDS_ASN1_MESSAGE_FROM_SYSTEM` to print the hexadecimal values on the command line.

The output should be as follows:

```
>sys
Starting SDL system...
Sending 'begin_line' message
Sending first point ('add_point' message)
Sending second point ('add_point' message)
Sending third point ('add_point_by_coords' message)
Sending 'end_line' message
Waiting for answer...
```

and then a sequence of hexadecimal digits representing the ASN.1 DER encoding of a value sent by the system (message line).

Organization

The messages are defined in the system '`sys`'; all their parameters are either ASN.1 defined types or SDL base types that have a direct mapping to ASN.1:

- `begin_line` & `end_line` have no parameters;
- `add_point` has a single parameter of type `PointType`, defined in `Types/BasicTypes.asn1`;
- `add_point_by_coords` has two parameters with type `Integer`, which can be mapped to the ASN.1 base type `INTEGER`;
- `line` has a single parameter with the type `PolylineType`, defined in `Types/ComplexTypes.asn1`.

Encoders & decoders generation

The generation of the ASN.1 encoders and decoders are triggered by checking the option 'Generate ASN.1 codecs for env. messages' in the code generation options ('Code gen.' tab, third group in the 'SDL-RT/SDL specific options'). If this option is checked, the following steps are added to the C code generation options:

- An ASN.1 file is generated for the types for the messages themselves. This file is called `RTDS-Asn1Messages.asn1` and contains:
 - One type for each message, named `RTDS-Message-<SDL message name>`, which is either a `SEQUENCE` with one field named `param<i>` for each message parameter if the message has parameters, or a 'syntype' to `NULL` if the message has no parameters;

- One type that is a CHOICE on all the messages to be able to represent a message as one kind of ASN.1 encoded stream. This type is always called `RTDS-AllMessages`.
- An ASN.1 'compiler' is run on this generated ASN.1 file as well as all the ones defining the types for the message parameters. It generates a number of C files in the 'asn1' sub-directory of the code generation directory, which are then automatically integrated in the build.
- The actual encoders and decoders are generated for the ASN.1 types. These convert the internal structures used for the types to the ones used by encoders and decoders generated by the ASN.1 compiler.
- Each message in the system that is sent to the environment is actually sent in the code by using the standard macro `RTDS_ASN1_MESSAGE_FROM_SYSTEM`, which takes as parameters the message number, the number of bytes in the encoded buffer and the buffer itself. The buffer will contain a variable of type `RTDS-AllMessages` as defined in `RTDS-Asn1Messages.asn1`, containing the full message sent from the system.

Note that the message type is already encoded in the buffer, so the message number passed to the macro is there for informational purposes only. It can be used to route the message to the correct receiver without having to decode the full ASN.1 stream.

- A macro called `RTDS_ASN1_MESSAGE_TO_SYSTEM` is generated in `RTDS_messages.h` allowing to send a message to the SDL system. Its parameters are the number of bytes in the encoded buffer as an unsigned long and the buffer itself as an unsigned char*.

The encoding used for all ASN.1 data is BER (or DER which is a subset of BER, so a BER decoder can be used to decode it). If you need to generate your own encoders and decoders for the parts communicating with the SDL system, just include `RTDS-Asn1Messages.asn1` in the set of source ASN.1 files; the data to encode or decode will always have the type `RTDS-AllMessages`.

Implementation in this example

The definition of the macro `RTDS_ASN1_MESSAGE_FROM_SYSTEM` is in the file `ASN1_adaptation.h` in the Adaptation package. This package also contains the file `sys_main.c` defining the main function for the generated executable, which sends messages to the SDL system, then waits for its answer and exits. A 'use' on the 'Adaptation' package in `sys` ensures that the header file will be included in the generated code and the source file compiled and linked with it. To prevent the generated code from including another main function, the option `-DRTDS_MAIN=SDL_main` is used in all sets of generation options. This renames the entry point function for the SDL system to 'SDL_main', which is then called in the actual main function in `sys_main.c`.

Generating ASN.1 encoding for other values

If you want to change the messages sent to the system, there's a message value generator utility in the `value_generator` sub-directory of the example's directory. This utility uses the C code generated by the ASN.1 compiler, so you'll have to generate the code for the SDL system first. Once this is done, refer to the file `compile_command.txt` in `value_generator` to compile the utility. Once it's compiled, you can run:

- `gen_value b`
This prints a hex-encoded string for a `begin_line` message.
- `gen_value e`
This prints a hex-encoded string for an `end_line` message.
- `gen_value p x y tag1,tag2,tag3,... [name]`
This prints a hex-encoded string for an `add_point` message with a point at coordinates (x, y), with tags tag1, tag2, tag3, ..., and with the name 'name' if specified on the command line. For a point with no tags, use - as the tag list.
- `gen_value c x y`
This prints a hex-encoded string for an `add_point_by_coords` message with a point at coordinates (x, y). This point will have no tags and no name.

The hex-encoded values can then be used in `sys_main.c`. Just make sure to include a little pause after each message send to make sure the message is correctly received and processed by the threads in the SDL system.