EndpointDocParser (in Tools)

How to use the EndpointDocParser

- The EndpointDocParser is found under OT\OpenTwin\Tools.
- Endpoints can be documented directly in the respective services according to the documentation schema listed below.
- When the EndpointDocParser is executed, the documented endpoints for each service are written to a Sphinx file, which is stored under OT\OpenTwin\Documentation\Developer\documented_endpoints. An index file named documented_endpoints.rst is also dynamically generated in Sphinx, which references the generated Service.rst files in its table of contents.
- No Sphinx documentation is generated for incorrectly documented endpoints. Instead, they are written to a text file called parseErrors.txt, which is stored in the EndpointDocParser project directory. They are also logged as errors.
- To keep track of the status of the documentation, the EndpointDocParser tracks endpoints to be documented and logs undocumented endpoints with a warning.

Endpoint documentation schema

Endpoint action:
//api @action OT_ACTION_CMD_MyAction
- Each endpoint must have an endpoint action

Security (message type):
//api @security TLS / mTLS
- Each endpoint should have a message type; TLS and mTLS are accepted
- Otherwise, mTLS is set by default and a warning is logged

Brief description:
//api @brief A short brief description.
- Each endpoint must have a brief description

Parameter:
//api @param OT_ACTION_PARAM_MyParam DataType Parameter description.

Response parameter:
//api @rparam OT_ACTION_PARAM_MyRParam DataType Response parameter description.
- None, one, or multiple parameters / response parameters are possible
- Data types could be Boolean, Char, Integer, Float, Double, String, Array, Object, Enum, and Unsigned Integer 64

Response description:
//api @return Any information about the response.

Detailed description:
//api Here is a more detailed description.
- Detailed descriptions can be added to a brief / response description and param / rparam descriptions

Paragraph:
//api
- Paragraphs can be added in the detailed description and in note and warning blocks

Warning-Block:
//api @warning This is a warning.
//api Second line of the warning.
//api @endwarning
- Warning blocks can be added to detailed descriptions

Note-Block:
//api @note This is a note.
//api Second line of the note.
//api @endnote

- Note blocks can be added to detailed descriptions

Documentation of the most important functions

importActionTypes()

- Parses the ActionTypes.h file of the OTCommunication library
- Searches for the prefix “#define” to identify the relevant lines
- Fills the map m_actionMacros with entries consisting of Macro : Definition
- Takes various cases into account:
  - OT_ACTION_PASSWORD_SUBTEXT "Password"
  - OT_ACTION_PARAM_SESSIONTYPE_STUDIOSUITE "CST Studio Suite"
  - OT_ACTION_RETURN_UnknownError OT_ACTION_RETURN_INDICATOR_Error "Unknown error"
  - OT_PARAM_AUTH_LOGGED_IN_USER_PASSWORD "LoggedInUser" OT_ACTION_PASSWORD_SUBTEXT
  - OT_PARAM_AUTH_PASSWORD OT_ACTION_PASSWORD_SUBTEXT

searchForServices()

- Goes through all include and src directory files of the OpenTwin services
- Searches line by line for:
  - Prefix „//api“:
    - indicates the beginning of an API documentation block
    - or indicates that you are in an API documentation block if the “inApiBlock” flag is also set
    - recognizes the end of an API documentation block if the prefix is missing but the flag is still set
  - Prefix „connectAction“
    - as an indicator for endpoints to be documented
- Parses an API documentation block and adds an error-free documented endpoint to the list of endpoints „m_endpoints“ in the service
- Adds a service that has documented endpoints to the list of services „m_services“
- Collects incorrectly documented endpoints in the m_parseErrors list
- Collects endpoints to be documented in the m_endpointsToBeDocumented list
- Uses an exitCode to evaluate whether there have been any serious errors (unable to open the file, etc.)

generateDocumentation()

- Creates the following for each service from the list of services:
  - A syntactically correct Sphinx documentation
  - And writes its content to an .rst file carrying the name of the service
- Then creates the index file documented_endpoints.rst and fills the table of contents with references to all .rst files created for the services
- Uses an exitCode to evaluate whether there were any serious errors (file could not be opened, etc.)

documentParseErrors()

- writes all incorrectly documented endpoints collected in m_parseErrors, if any, to a file named parseErrors.txt and stores it in the EnpointDocParser directory.

reportEndpointsToBeDocumented()

- Uses the endpoint action to compare whether the endpoints to be documented, which are contained in the „m_endpointsToBeDocumented“ list, are also present in the list of documented endpoints „m_endpoints“ in the relevant service
- If not present, an OT_LOG_W is thrown for undocumented endpoints and the number of undocumented endpoints is output as OT_LOG_D

What still needs to be done

Sphinx Documentation:

- Remove documented endpoints from OpenTwin.net under List of Services; instead, add them under Documented Endpoints or select a different path for the parser to automatically store the files

EndpointDocParser:

- Prepare service names for headings in Sphinx documentation (AuthorisationService → Authorisation Service)
- Unit tests
- Extend documentation schema:
  - Paramter:
    - (see also https://opentwin.net/doc/how_to/document_the_endpoints.html)
    - specify the structure of the nested object
    - add a reference to the class documentation, maybe by using the following syntax: //api @ref ot::ClassName Text to display for the hyperlink label.
  - Response:
    - „A String containing an ot::ReturnMessage.“ (ot::ReturnMessage should be a link)
    - „In case of error will return a String with prefix “Error: “.“
    - If not providing any return text a default text indicating that this enpoint has no return value should be generated.
  - If no parameters/Rparameters are available, insert a sentence in the Sphinx documentation to indicate that none exist.
- Consider special cases when searching for endpoints to be documented
  - AuthorisationService (dispatchAction)
  - OT_HANDLER
  - connectActions…
    - without an endpoint action, e.g.: connectAction(c_setProjectEntitySelectedAction, this, &Application::handleSetProjectEntitySelected);
    - with more than one endpoint action, e.g.: connectAction({ OT_ACTION_CMD_PYTHON_EXECUTE_Command, OT_ACTION_CMD_PYTHON_EXECUTE_Scripts }, this, &Application::handleForwardToSubprocess);
    - These endpoints are detected by the parser, but there is no check to see whether they have been documented.