# Parsing and exceptions

When the input is not valid JSON, an exception of type `parse_error` is thrown. This exception contains the position in the input where the error occurred, together with a diagnostic message and the last read input token. The exceptions page contains a list of examples for parse error exceptions. In case you process untrusted input, always enclose your code with a `try` / `catch` block, like

```
json j;
try
{
    j = json::parse(my_input);
}
catch (json::exception::parse_error& ex)
{
    std::cerr << "parse error at byte " << ex.byte << std::endl;
}
```

In case exceptions are undesired or not supported by the environment, there are different ways to proceed:

## Switch off exceptions

The `parse()` function accepts as last parameter a `bool` variable `allow_exceptions` which controls whether an exception is thrown when a parse error occurs ( `true`, default) or whether a discarded value should be returned ( `false`).

```
json j = json::parse(my_input, nullptr, false);
if (j.is_discarded())
{
    std::cerr << "parse error" << std::endl;
}
```

Note there is no diagnostic information available in this scenario.

## Use accept() function

Alternatively, function `accept()` can be used which does not return a `json` value, but a `bool`

indicating whether the input is valid JSON.

```
if (!json::accept(my_input))
{
    std::cerr << "parse error" << std::endl;
}
```

Again, there is no diagnostic information available.

## User-defined SAX interface

Finally, you can implement the SAX interface and decide what should happen in case of a parse error.

This function has the following interface:

```
bool parse_error(std::size_t position,
                 const std::string& last_token,
                 const json::exception& ex);
```

The return value indicates whether the parsing should continue, so the function should usually return `false`.

**Example**

```
#include <iostream>
#include "json.hpp"

using json = nlohmann::json;

class sax_no_exception : public nlohmann::detail::json_sax_dom_parser<json>
{
  public:
    sax_no_exception(json& j)
      : nlohmann::detail::json_sax_dom_parser<json>(j, false)
    {}

    bool parse_error(std::size_t position,
                     const std::string& last_token,
                     const json::exception& ex)
    {
        std::cerr << "parse error at input byte " << position << "\n"
                  << ex.what() << "\n"
                  << "last read: \"" << last_token << "\""
                  << std::endl;
        return false;
    }
};

int main()
{
    std::string myinput = "[1,2,3,]";

    json result;
    sax_no_exception sax(result);

    bool parse_result = json::sax_parse(myinput, &sax);
    if (!parse_result)
    {
        std::cerr << "parsing unsuccessful!" << std::endl;
    }

    std::cout << "parsed value: " << result << std::endl;
}
```

Output:

```
parse error at input byte 8
[json.exception.parse_error.101] parse error at line 1, column 8: syntax error
while parsing value - unexpected ']'; expected '[', '{', or a literal
last read: "3,]"
parsing unsuccessful!
parsed value: [1,2,3]
```