

{{lastupdated_at}} by {{lastupdated_by}}

Exceptions

GammaLib implements error handling using exceptions. The exceptions are implemented by the GException class. GammaLib distinguishes between **logic exceptions** and **runtime exceptions**.

Logic exceptions

Logic exceptions occur in situations that a client could have tested. They comprise:

- `invalid_value`: a value is invalid
- `invalid_argument`: an argument passed to a method or a function is invalid
- `out_of_range`: a value is outside its valid range
- `fits_error`: an error occurred in a cfitsio routine

Runtime exceptions

Runtime exceptions occur in situations that a client can not test. They comprise:

- `underflow_error`
- `overflow_error`
- `feature_not_implemented`: the requested feature is not yet implemented in GammaLib

Example code

Below an example code that illustrates how exceptions should be implemented:

```
if (num != m_cube.nmaps() ) {
    std::string msg = "Number of energies in 'ENERGIES' extension"
        " (" +gammalib::str(num)+") does not match the"
        " number of maps (" +gammalib::str(m_cube.nmaps())+" "
        " in the map cube.\n"
        "The 'ENERGIES' extension table shall provide"
        " one enegy value for each map in the cube.";
    throw GException::invalid_value(G_LOAD, msg);
}
```

An error message should be composed using a `std::string`. This message is then passed to `invalid_value`, `invalid_argument` or any of the other standard exceptions. The `G_LOAD` macro defines the name of the method that actually throws the exception (and is defined in the header of the `.cpp` file):

```
#define G_LOAD          "GModelSpatialDiffuseCube::load(std::string&)"
```

Note that arguments are given in this definition without the `const` declaration and without the parameter name. If more than a single parameter exists, the parameter should be separated by a blank character:

```
#define G_MC            "GModelSpatialDiffuseCube::mc(GEnergy&, GTime&, GRan&)"
```