```
// serializeToJson.h:
#include <isSerializableToJson.h>  // general trait variable template

template <typename T>
JsonObject serializeToJson(const T& object)  // serialization function template
{
    static_assert(isSerializableToJson<T>,
                  "T must support serialization to JSON.");

    // ...

    return { /*...*/ };
}
```

Notice that we have used the C++11 **`static_assert`** feature to ensure that any type used to instantiate this function will have specialized the general variable template associated with the specific type to be **`true`**; see the next code snippet.

Now imagine that we have a type, `CompanyData`, that we would like to advertise at compile time as being serializable to JSON. Like other templates, variable templates can be specialized explicitly:

```
// companyData.h:
#include <isSerializableToJson.h>  // general trait variable template

struct CompanyData { /*...*/ };    // type to be JSON serialized

template <>
constexpr bool isSerializableToJson<CompanyData> = true;
    // Let anyone who needs to know that this type is JSON serializable.
```

Finally, our `client` function incorporates all of the above and attempts to serialize both a `CompanyData` object and an `std::map<int, char>`:

```
// client.h:
#include <isSerializableToJson.h>  // general trait template
#include <companyData.h>           // JSON serializable type
#include <serializeToJson.h>       // serialization function
#include <map>                     // std::map (not JSON serializable)
```