Bundesamt für Sicherheit in der Informationstechnik

Deutschland Digital•Sicher•BSI•

BSI TR-03151-2

Technical Guideline BSI TR-03151 Secure Element API (SE API)

Part 2: Interface Mapping

Version 1.1.0
2023-02-13

Version history

Version	Date	Description
1.1.0	2023-02-13	Initial version

Table 1 Version history

Federal Office for Information Security
Post Box 20 03 63
D-53133 Bonn

E-Mail: registrierkassen@bsi.bund.de
Internet: https://www.bsi.bund.de

Table of Contents

1 Introduction

The Technical Guideline BSI TR-03151 is split into two distinct parts:

- Part 1, the document [BSI TR-03151-1], describes the Secure Element API (SE API)
- Part 2, this document, describes the mapping of the SE API to different programming and description languages.

Part 1 describes the Secure Element API by defining functions. The function definitions include, besides function names and descriptions of the functional behaviour, input and output parameters of each function.

To allow a (programming) language-independent definition of data types used within functions, the [BSI TR-03151-1] utilizes the definition language OMG IDL, defined in [OMG2017a].

This document, part 2, provides an introduction into certain constructs that have been used in the context of the API definition as well as the data type definition.

Part of this document are appendices (in the form of further documents) that specify the mapping of these constructs to specific (programming) languages. Examples for such appendices to this Technical Guideline would be the mapping to ANSI C [TR-03151-2: ANSI C] and Java [TR-03151-2: Java].

Note: The mapping established and presented in this document and its appendices can be utilized to map any interface definition to the programming and interface languages supported by the appendix document, as long as the interface definition concept of [BSI TR-03151-1] is used to define the API.

2 Mapping of an API

The following sub-chapters describe the core concepts used for the definition of the API, including data types, exceptions, and function parameters.

2.1 Data types

Data types are used to store values and are usually manipulated during the execution of functions. There are basic types, that usually only contain one value at a time and more complex ones which may contain multiple values. The data types used in this mapping context are the types defined within [OMG2017a]. This definition is independent from any programming language but can be mapped to different languages for the implementation. Note: (Programming) language specific mappings of the OMG IDL data types mentioned here can be found in the respective appendix documents of this document.

2.1.1 Basic types

Table 2 contains information regarding the basic data types that are used. This table has been specified under consideration of the following aspects:

- OMG IDL standard definition for the syntax of basic types in [OMG2017a], Chap. 7.4.1.4.4.1.1, p. 25 f.
- The value ranges for the integer types in OMG IDL as defined in [OMG2017a], Chap. 7.4.1.4.4.1.1.1, p. 26.

OMG IDL name	**Value range / comment**
short	(-2^{15} ... 2^{15}- 1)
long	(-2^{31} ... 2^{31}- 1)
long long	(-2^{63} ... 2^{63}- 1)
unsigned short	(0 ... 2^{16}-1)
unsigned long	(0 ... 2^{32}-1)
unsigned long long	(0 ... 2^{64} -1)
octet	“The octet type is an opaque 8-bit quantity” (see [OMG2017a], Chap. 7.4.1.4.4.1.1.6, p. 27)
boolean	The boolean data type can only take the values TRUE and FALSE. (see [OMG2017a], Chap. 7.4.1.4.4.1.1.5, p. 27)
Date/Time values (non-OMG IDL name)	Represented by the native type DateTime. (see [OMG2017a], Chap. 7.4.1.4.4.4, p. 32 f).

Table 2 Mapping of data types

2.1.2 Complex types

2.1.2.1 Strings

Regarding the definition of strings, the OMG IDL distinguishes between non-wide strings and wide strings. As no wide strings have been used in this mapping context, the following discussion refers only to non-wide strings.

In the context of the OMG IDL, non-wide strings are represented by the type string (see [OMG2017a], Chap. 7.4.1.4.4.1.2.2, p. 27). Optionally, it is possible to define the maximum size of a string. The size of a string is represented by a positive integer value that is surrounded by the signs < and >.

In contrast to the OMG IDL definition of strings, in this mapping context an API may require the return of an empty string. See the appendices to this document for further information on how to handle empty strings in different (programming) languages.

Text 1 shows different examples of OMG IDL strings.

OMG IDL

```
string<100> textWithBounds
string textWithoutBounds
string emptyText = ""
```

Text 1: Example for OMG IDL strings

2.1.2.2 Enumerations

The OMG IDL provides the language construct enumeration (see [OMG2017a], Chap. 7.4.1.4.4.2.3) that enables the definition of data types with a finite set of values. Enumerations are defined by the keyword enum that is followed by the name of the enumeration. The set of values is contained in braces. The values are separated by commas.

Text 2 shows an example for OMG IDL enumerations.

OMG IDL

```
enum Color {red, green, blue, orange};
enum Fruit {apple, orange};
```

Text 2: Example for OMG IDL enumerations

2.1.2.3 Arrays

The OMG IDL provides the language construct array (see [OMG2017a], Chap. 7.4.1.4.4.3) that represents the data structure array. These arrays can be one-dimensional or multi-dimensional. In deviation from [OMG2017a] the use of arrays of variable length is allowed.

In this mapping context, only one-dimensional arrays are used as types of function parameters. At this, an array is represented by its type, its name and a following opening and closing squared bracket. If the array is of fixed length, the squared brackets contain the definition for the size of the array in form of a positive integer value. If an array has no fixed size, the squared brackets contain no integer number. This notation for the use of arrays with no fixed size represents an extension to [OMG2017a].

Text 3 shows the difference between an OMG IDL array named “example” with a fixed length of 20 and the API definition of the byte array value called “variableExample” with a variable length.

OMG IDL

```
octet example [20]
```

OMG IDL as used in this mapping context

```
octet variableExample[]
```

Text 3: Example for the difference between a classic OMG IDL array and one used in this API mapping context

2.1.2.4 Definition context

The way particular API functions are implemented depends on the (programming) language. Refer to the appendices to this document for further informations (e.g. [TR-03151-2: ANSI C], [TR-03151-2: Java]).

2.2 Exceptions

Exceptions are a means of error handling. In this mapping context API exceptions are part of the defined function to indicate certain errors.

There are two basic ways to implement the exceptions that have been specified for the functions:

- Some programming languages support exceptions as an explicit construct of the language. In erroneous situations, a function raises an exception at the point where an error is detected. Here, the program flow of the function is interrupted immediately and the exception is caught in the program code or by the code of the calling application. Therefore, the function does not return a return value.
- If no specialized language constructs are provided, exception handling is implemented by error codes. Here, the function exits its program flow when an error is detected by returning an appropriate error code as the return value.

For more information on how different (programming) languages implement the exceptions defined by the API, refer to the appendices to this document (e.g. [TR-03151-2: ANSI C], [TR-03151-2: Java]).

2.2.1 Exceptions with additional information

The API definitions, e.g. in [BSI TR-03151-1], occasionally require additional information to be passed alongside the exception raised. This is mentioned in the detailed description of a function, usually highlighted by a variation of the phrase “[...] provide additional information on XYZ alongside the exception”. Instead of XYZ, the detailed description contains a concrete value on which the additional information shall be provided.

For more information on how different (programming) languages implement exceptions with additional information as defined by the API, refer to the appendices to this document (e.g. [TR-03151-2: ANSI C], [TR-03151-2: Java]).

2.3 Function parameters

2.3.1 Function input parameters

An input parameter is a special kind of variable which is used to pass data to a function upon which the processing of the function depends. Within this API mapping context every input parameter is associated with an OMG IDL data type. The types are mapped according to chapter 2.1 for both basic and complex types. See for example [TR-03151-2: Java] and [TR-03151-2: ANSI C] for further information on how to handle input parameters in Java and ANSI C.

2.3.2 Function output parameters

In contrast to input parameters an output parameter is used to pass data from a function back to the caller. Since every (programming) language handles output parameters differently or might not even support them directly, the handling of them is delegated to the appendices of this Technical Guideline (e.g. [TR-03151-2: ANSI C], [TR-03151-2: Java]).

2.3.3 Optional and conditional function parameters

OMG IDL does not provide any construct for representing optional and conditional input and output parameters of a function. Nevertheless, functions defined might have input parameters which may be omitted generally (optional input parameters) or based on the presence or value of another input parameters (conditional input parameters). The same applies to output parameters which might be missing depending on certain conditions (conditional output parmeters). Informations about optional or conditional parameters can be found in the respective API defining document, e.g. Technical Guideline [BSI TR-03151-1].

For more information on how different (programming) languages implement optional and conditional parameters defined by the API, refer to the appendices to this document (e.g. [TR-03151-2: ANSI C], [TR-03151-2: Java]).

2.3.4 Large data function parameters

For some functions in the API definition it might be necessary to return a large amount of data. This might for example apply to functions with the purpose of exporting large amounts of logs. Please refer to the appendices to this document for further information on how to handle large amounts of data in different (programming) languages.

References

BSI TR-03151-1	BSI: Technical Guideline BSI TR-03151-1: Secure Element API (SE API) – Part 1: Interface Definition
TR-03151-2: Java	BSI: Technical Guideline BSI TR-03151-2: Secure Element API (SE API) – Part 2: Interface Mapping – Appendix Java: Mapping of APIdefinitions to Java
TR-03151-2: ANSI C	BSI: Technical Guideline BSI TR-03151-2: Secure Element API (SE API) – Part 2: Interface Mapping – Appendix ANSI C: Mapping of API definitions to ANSI C
OMG2017a	OMG: Interface Definition Language, Version 4.1, 2017