7 Interfaces

Interfaces provide the ability to name and parameterize object types and to compose existing named object types into new ones.

Interfaces have no run-time representation—they are purely a compile-time construct. Interfaces are particularly useful for documenting and validating the required shape of properties, objects passed as parameters, and objects returned from functions.

Because TypeScript has a structural type system, an interface type with a particular set of members is considered identical to, and can be substituted for, another interface type or object type literal with an identical set of members (see section 3.8.2).

Class declarations may reference interfaces in their implements clause to validate that they provide an implementation of the interfaces.

7.1 Interface Declarations

An interface declaration declares a new named type (section 3.5) by introducing a type name in the containing module.

InterfaceDeclaration:
`interface` *Identifier* *TypeParameters*$_{opt}$ *InterfaceExtendsClause*$_{opt}$ *ObjectType*

InterfaceExtendsClause:
`extends` *ClassOrInterfaceTypeList*

ClassOrInterfaceTypeList:
ClassOrInterfaceType
ClassOrInterfaceTypeList `,` *ClassOrInterfaceType*

ClassOrInterfaceType:
TypeReference

The *Identifier* of an interface declaration may not be one of the predefined type names (section 3.6.1).

An interface may optionally have type parameters (section 3.4.1) that serve as placeholders for actual types to be provided when the interface is referenced in type references. An interface with type parameters is called a ***generic interface***. The type parameters of a generic interface declaration are in scope in the entire declaration and may be referenced in the *InterfaceExtendsClause* and *ObjectType* body.

An interface can inherit from zero or more ***base types*** which are specified in the *InterfaceExtendsClause*. The base types must be type references to class or interface types.