Compiler Messages

This page lists selected error messages and explanations on how to fix them. It is not a complete list of all compiler messages.

> **Note on message IDs**
>
> Message IDs are not finalized, yet. They can change at short notice.

Table of Contents

- type-unexpected-foreign-keys
- type-unexpected-on-condition
- wildcard-excluding-one

---

anno-duplicate-unrelated-layer

An annotation is assigned multiple times through unrelated layers.

A *layer* can be seen as a group of connected sources, for example CDL files. They form a cyclic connection through their dependencies (for example, `using` in CDL). If there are no cyclic dependencies, a single CDL file is equivalent to a layer.

Example

Erroneous code example using four CDS files:

```cds
// (1) Base.cds: Contains the artifact that should be annotated
entity FooBar {  }

// (2) FooAnnotate.cds: First unrelated layer to Base.cds
using from './Base';
annotate FooBar with @Anno: 'Foo';

// (3) BarAnnotate.cds: Second unrelated layer to Base.cds
using from './Base';
annotate FooBar with @Anno: 'Bar';

// (4) All.cds: Combine all files ❌
using from './FooAnnotate';
using from './BarAnnotate';
```

In (4) the compiler will warn that there are duplicate annotations in unrelated layers. That is because (2) and (3) are unrelated, i.e. they do not have a connection.

Due to these unrelated layers, the compiler can't decide in (4) which annotation should be applied first.

Instead of passing (4) to the compiler, you can also pass (2) and (3) to it. Because there are no cyclic dependencies between the files, each file represents one layer.

How to Fix

Remove one of the duplicate annotations. Chances are, that only one was intended to begin with. For the erroneous example above, remove the annotation from (3).

Alternatively, add an annotation assignment to (4). This annotation has precedence and the error will vanish. For the example above, (4) will look like this:

```cds
// (4) All.cds: Combine all files
using from './FooAnnotate';
using from './BarAnnotate';
// This annotation has precedence.
annotate FooBar with @Anno: 'Bar';
```

You can also make (3) depend on (2) so that they are no longer in unrelated layers and the compiler can determine which annotation to apply.

```cds
// (3) BarAnnotate.cds: Now depends on (2)
using from './FooAnnotate';
annotate FooBar with @Anno: 'Bar';
```

This works because there is now a defined dependency order.

---

anno-missing-rewrite

A propagated annotation containing expressions can't be rewritten and would end up with invalid paths.

While propagating annotations containing expressions such as `@anno: (path)`, the compiler ensures that the path remains valid. If necessary, the paths have to be rewritten, e.g. when being propagated to projections that rename their source's elements. If rewriting is not possible, this error is emitted.

Example

Erroneous code example:

```cds
type T : {
    @anno: (sibling)
    elem: String;
    sibling: String;
};
type TString : T:elem; // ❌ there is no `sibling`
```

The annotating `@anno` would be propagated to `TString`. However, because its path refers to an element that is not reachable at `TString`, the path can't be rewritten and compilation fails.

How to Fix

Explicitly override the annotation. Either remove it by setting its value to `null` or by using another value.

```cds
// (1) direct annotation
@anno: null
type TString : T:elem;

// (2) annotate statement
type TString : T:elem;
annotate TString with @(anno: null);
```

Variant (1) may not always be applicable, e.g. if annotations in a structured type would need to be overridden. In those cases, use variant (2) and assign annotations via the `annotate` statement.

---

check-proper-type-of

An element in a `type of` expression doesn't have proper type information.

The message's severity is `Info` but may be raised to `Error` in the SQL, SAP HANA, and OData backends. These backends require elements to have a type. Otherwise, they aren't able to render elements (for example, to SQL columns).

Example

Erroneous code example:

```cds
entity Foo {
  key id : Integer;
};
view ViewFoo as select from Foo {
  1+1 as calculatedField @(anno)
};
entity Bar {
  // ❌ `e` has no proper type but has the annotation `@anno`.
  e : ViewFoo:calculatedField;
};
```

`ViewFoo:calculatedField` is a calculated field without an explicit type.

`type of` is used in *`E:e`* 's type specification. You would expect the element to have a proper type. However, because the referenced element is calculated, the compiler isn't able to determine the correct type. The element still inherits *`ViewFoo:calculatedField`* 's annotations and other properties but won't have a proper type, which is required by some backends.

How to Fix

Assign an explicit type to *`ViewFoo:calculatedField`* .

```cds
view ViewFoo as select from Foo {
  1+1 as calculatedField @(anno) : Integer
};
```

Related Messages

- *`def-missing-type`*

---

def-duplicate-autoexposed

Two or more entities with the same name can't be auto-exposed in the same service.

Auto-exposure is a compiler feature which makes it easier for developers to write services. Auto-exposure uses the name of the entity to expose it in the service. It ignores the entity's namespace and context. This may lead to name collisions.

The message's severity is `Error` and is raised by the compiler. You need to adapt your model to fix the error.

Example

Erroneous code example:

```cds
// (1)
entity ns.first.Foo {
  key parent : Association to one ns.Base;
};
// (2)
entity ns.second.Foo {
  key parent : Association to one ns.Base;
};
// (3)
entity ns.Base {
  key id    : UUID;
  to_first  : Composition of many  ns.first.Foo;
  to_second : Composition of many ns.second.Foo;
}
service ns.MyService {
  // (4) ❌
  entity BaseView as projection on ns.Base;
};
```

Both (1) and (2) define an entity `Foo`, but in different namespaces. For example, they could be located in different files with a `namespace` statement. (3) contains compositions of both `first.Foo` and `second.Foo`.

In (4), a projection on `Base` is exposed in service `MyService`. Both composition targets are auto-exposed. However, because the namespaces of (2) and (3) are ignored, a name collision happens.

How to Fix

You need to explicitly expose one or more entities under a name that does not exist in the service, yet.

For the erroneous example above, you could add these two lines to the service `ns.MyService` :

```cds
entity first.Foo as projection on ns.first.Foo;   // (5)
entity second.Foo as projection on ns.second.Foo; // (6)
```

Here we reuse the namespaces `first` and `second` . We don't use `ns` because it's the common namespace. But you can choose any other name.

The compiler will pick up both manually exposed entities and will correctly redirect all associations.

Note: For the example, it is sufficient to expose only one entity. If you remove (6), you will get these two projections:

- `ns.MyService.first.Foo` for (5)
- `ns.MyService.Foo` for (6) Where (6) is the name chosen by the compiler.

Notes on auto-exposure

You may wonder why the compiler does not reuse the namespace when auto-exposing entities. The reason is that the resulting auto-exposed names could become *long* names that don't seem natural nor intuitive. We chose to expose the entity name because that's what most developers want to do when they manually expose entities.

Other Notes

This message was called `duplicate-autoexposed` in cds-compiler v3 and earlier.

---

def-missing-type

A type artifact doesn't have proper type information.

The message's severity is `Info` but may be raised to `Error` in the SQL, SAP HANA, and OData backends. These backends require types to have type information. Otherwise,

they aren't able to render elements that use this type (for example, to SQL columns).

Example

Erroneous code example:

```json
{
  "definitions": {
    "MainType": {
      "kind": "type"
    }
  }
}
```

`MainType` is of kind "type" but has not further type-information.

How to Fix

Add explicit type information to `MainType`, for example, add an `elements` property to make a structured type.

```json
{
  "definitions": {
    "MainType": {
      "kind": "type",
      "elements": {
        "id": {
          "type": "cds.String"
        }
      }
    }
  }
}
```

Related Messages

- `check-proper-type-of`

def-upcoming-virtual-change

The behavior of `@sap/cds-compiler` v6 will change for a selected element.

Example

Erroneous code example:

```cds
entity Source {
  key ID : String;
  a : String;
};

entity Proj as projection on Source {
  ID,
  virtual a, // ❌ behavior will change in v6
};
```

In `@sap/cds-compiler` v5 and earlier, element `Proj:a` is a reference to element `Source:a`, which was marked virtual.

In `@sap/cds-compiler` v6 and later, it will instead be a *new* element, without any reference to `Source:a`.

This may or may not affect your runtime coding, hence the warning.

How to Fix

If the v6 behavior works for you, there is nothing you need to do.

However, if you want to keep a reference to `Source:a` in CSN, for example because you use the reference at runtime, then you can keep the old behavior by either:

1. prepending a table alias to the reference
2. adding a column alias

```cds
// (1) prepend a table alias
entity V as projection on E {
  ID,
```

```cds
  virtual E.a, // ok
};

// (2) add an alias
entity V as projection on E {
  ID,
  virtual a as a, // ok
};
```

extend-repeated-intralayer

The order of elements of an artifact may not be stable due to multiple extensions in the same layer (for example in the same file).

A *layer* can be seen as a group of connected sources, for example, CDL files. They form a cyclic connection through their dependencies (for example, `using` in CDL).

Example

Erroneous code example with multiple CDL files:

```cds
// (1) Definition.cds
using from './Extension.cds';
entity FooBar { };
extend FooBar { foo: Integer; }; // ❌

// (2) Extension.cds
using from './Definition.cds';
extend FooBar { bar: Integer; }; // ❌
```

Here we have a cyclic dependency between (1) and (2). Together they form one layer with multiple extensions. Again, the element order isn't stable.

How to Fix

Move extensions for the same artifact into the same extension block:

```cds
// (1) Definition.cds : No extension block
using from './Extension.cds';
entity FooBar { }

// (2) Extension.cds : Now contains both extensions
using from './Definition.cds';
extend FooBar {
  foo : Integer;
  bar : Integer;
}
```

Related Messages

- *extend-unrelated-layer*

---

extend-unrelated-layer

Unstable element order due to extensions for the same artifact in unrelated layers.

A *layer* can be seen as a group of connected sources, for example CDL files. They form a cyclic connection through their dependencies (for example, *using* in CDL).

Example

Erroneous code example using four CDS files:

```cds
// (1) Base.cds: Contains the artifact that should be extended
entity FooBar {  }

// (2) FooExtend.cds: First unrelated layer to Base.cds
using from './Base';
extend FooBar { foo : Integer; }

// (3) BarExtend.cds: Second unrelated layer to Base.cds
using from './Base';
extend FooBar { bar : Integer; }
```

```
// (4) ❌ All.cds: Combine all files
using from './FooExtend';
using from './BarExtend';
```

In (4) the compiler will warn that the element order of *FooBar* is unstable. That is because the extensions in (2) and (3) are in different layers and when used in (4) it can't be ensured which extension is applied first.

Instead of passing (4) to the compiler, you can also pass (2) and (3) to it. Because there are no cyclic dependencies between the files, each file represents one layer.

How to Fix

Move extensions for the same artifact into the same layer, that is, the same file.

For the erroneous example above, remove the extension from (3) and move it to (2):

cds

```
// (2) FooExtend.cds
using from './Base';
extend FooBar {
  foo : Integer;
  bar : Integer;
}
```

Related Messages

- *extend-repeated-intralayer*

---

file-unexpected-case-mismatch

The filename of a *using* statement does not match the file's actual name on disk.

To avoid operating-system dependent issues, the compiler checks if the name of an imported file matches the name of the file in the filesystem / on disk. For example, by default macOS uses a case-insensitive file system. Hence, a file named *model.cds* will also be loaded by *using from './Model.cds'* on such systems.

However, on other filesystems that are case-sensitive, e.g. when building your application in another environment, the file will not be found.

Hence, the `using` statement needs to be adapted.

Example

Erroneous code example:

```cds
// index.cds
using from './Model';
```

using following directory tree:

```
├── index.cds
└── model.cds
```

On case-insensitive systems, the file can be loaded, but the compiler will warn about the mismatch. On case-sensitive file systems, compilation will fail, as the imported file can't be found.

While in this case, compilation will fail on case-sensitive systems, it could instead end up with semantic changes, too. Given the same `index.cds`, but a different directory tree:

```
├── index.cds
├── Model
│   └── index.cds
└── model.cds
```

On case-sensitive systems, `./Model/index.cds` will be loaded. On case-insensitive systems, however, `model.cds` will be loaded, as the compiler first tries to load `Model.cds`, before looking for `Model/index.cds`.

How to Fix

Adapt the filename in your `using` statement.

If you have both `model.cds` and `Model/index.cds`, but don't want to use a `.cds` suffix, use `using from './Model/'`, i.e. add a trailing slash to indicate that you want to load from the folder `Model`.

redirected-to-ambiguous

The redirected target originates more than once from the original target through direct or indirect sources of the redirected target.

The message's severity is `Error` and is raised by the compiler. The error happens due to an ill-formed redirection, which requires changes to your model.

Example

Erroneous code example:

```cds
entity Main {
      key id : Integer;
    toTarget : Association to Target;
}

entity Target {
    key id : Integer;
}

view View as select from
    Main,
    Target,
    Target as Duplicate
{
    // ❌ This redirection can't be resolved:
    Main.toTarget : redirected to View
};
```

Entity `Target` exists more than once in `View` under different table aliases. In the previous example, this happens through the *direct* sources in the select clause. Because the original target exists twice in the redirected target, the compiler isn't able to correctly resolve the redirection due to ambiguities.

This can also happen through *indirect* sources. For example if entity `Main` were to include `Target`, then selecting from `Target` just once would be enough to trigger this error.

How to Fix

You must have the original target only once in your direct and indirect sources. The previous example can be fixed by removing `Duplicate` from the select clause.

```cds
view View as select from Main, Target {
    Main.toTarget : redirected to View
};
```

If this isn't feasible then you have to redefine the association using a mixin clause.

```cds
view View as select from Main, Target mixin {
    toMain : Association to View on Main.id = Target.id;
} into {
    Main.id   as mainId,
    Target.id as targetId,
    toMain
};
```

Related Messages

- `redirected-to-unrelated`
- `redirected-to-complex`

---

redirected-to-complex

The redirected target is a complex view, for example, contains a JOIN or UNION.

The message's severity is `Info` and is raised by the compiler. It is emitted to help developers identify possible modeling issues.

Example

Erroneous code example:

```cds
entity Main {
    key id : Integer;
    // self association for example purpose only
    toMain : Association to one Main;
```

```
}
entity Secondary {
    content: String;
};
entity CrossJoin as SELECT from Main, Secondary;
entity RedirectToComplex as projection on Main {
    id,
    toMain: redirected to CrossJoin, // ❌
};
```

`Main:toMain` is a to-one association. Since `Main` contains a single key, which is used in the managed association, we know that following the association returns a single result.

The cross join in the view `CrossJoin` results in multiple rows with the same `id`. Following the redirected view now returns multiple results, effectively making the to-one association a to-many association.

Visualizing the tables with a bit of data, this issue becomes obvious:

markdown

```
Main                        Secondary
| id  | toMain_id |         | content |
|-----|-----------|         |---------|
| 1   | 2         |         | 'Hello' |
| 2   | 1         |         | 'World' |

CrossJoin
| id  | toMain_id | content |
|-----|-----------|---------|
| 1   | 2         | 'Hello' |
| 1   | 2         | 'World' |
| 2   | 1         | 'Hello' |
| 2   | 1         | 'World' |
```

How to Fix

First, ensure that the redirected association points to an entity that is a reasonable redirection target. That means, the redirection target shouldn't accidentally make it a to-many association.

Then add an explicit ON-condition or explicit foreign keys to the redirected association. That will silence the compiler message.

Related Messages

- *redirected-to-ambiguous*
- *redirected-to-unrelated*

---

redirected-to-unrelated

The redirected target doesn't originate from the original target.

The message's severity is *Error* and is raised by the compiler. The error happens due to an ill-formed redirection, which requires changes to your model.

Example

Erroneous code example:

```cds
entity Main {
    key id : Integer;
    // self association for example purpose only
    toMain : Association to Main;
}
entity Secondary {
    key id : Integer;
}
entity InvalidRedirect as projection on Main {
    id,
    // ❌ Invalid redirection
    toMain: redirected to Secondary,
};
```

Projection *InvalidRedirect* tries to redirect *toMain* to *Secondary*. However, that entity doesn't have any connection to the original target *Main*, that means, it doesn't originate from *Main*.

While this example may be clear, your model may have multiple redirections that make the error not as obvious.

Erroneous code example with multiple redirections:

```cds
entity Main {
    key id : Integer;
    toMain : Association to Main;
}
entity FirstRedirect as projection on Main {
    id,
    toMain: redirected to FirstRedirect,
}
entity SecondRedirect as projection on FirstRedirect {
    id,
    // Invalid redirection
    toMain: redirected to Main,
}
```

The intent of the example above is to redirect `toMain` to its original target in `SecondRedirect`. But because `SecondRedirect` uses `toMain` from `FirstRedirect`, the original target is `FirstRedirect`. And `Main` doesn't originate from `FirstRedirect` but only vice versa.

How to Fix

You must redirect the association to an entity that originates from the original target. In the first example above you could redirect `SecondRedirect:toMain` to `SecondRedirect`. However, if that isn't feasible then you have to redefine the association using a mixin clause.

```cds
view SecondRedirect as select from FirstRedirect mixin {
    toMain : Association to Main on id = $self.id;
} into {
    FirstRedirect.id as id,
    toMain
};
```

Related Messages

- `redirected-to-ambiguous`
- `redirected-to-complex`

rewrite-not-supported

The compiler isn't able to rewrite ON conditions for some associations. They have to be explicitly defined by the user.

The message's severity is `Error`.

Example

Erroneous code example:

```cds
entity Base {
  key id     : Integer;
  primary    : Association to Primary on primary.id = primary_id;
  primary_id : Integer;
}

entity Primary {
  key id       : Integer;
  secondary    : Association to Secondary on secondary.id = secondary_id;
  secondary_id : Integer;
}

entity Secondary {
  key id : Integer;
  text   : LargeString;
}

entity View as select from Base {
  id,
  primary.secondary // ❌ The ON condition isn't rewritten here
};
```

In the previous example, the ON condition in `View` of `secondary` can't be automatically rewritten because the associations are unmanaged and the compiler can't determine how to properly rewrite them for `View`.

How to Fix

You have to provide an explicit ON condition. This can be achieved by using the `redirected to` statement:

```cds
entity View as select from Base {
  id,
  primary.secondary_id,
  primary.secondary: redirected to Secondary on
    secondary.id = secondary_id
};
```

In the corrected view above, the association `secondary` gets an explicit ON condition. For this to work it is necessary to add `secondary_id` to the selection list, that means, we have to explicitly use the foreign key.

Related Messages

- `rewrite-undefined-key`

---

rewrite-undefined-key

The compiler isn't able to rewrite an association's foreign keys, because the redirected target is missing elements to match them.

The message's severity is `Error` .

Example

Erroneous code example:

```cds
entity model.Base {
  key ID : UUID;
  toTarget : Association to model.Target; // (1)
}
entity model.Target {
  key ID : UUID; // (2)
  field : String;
}

service S {
  entity Base as projection on model.Base; // ❌ (3) Can't redirect 'toTarget
```

```
  entity Target as projection on model.Target {
    field, // (4) No 'ID'
  };
}
```

In the example, the projected association `toTarget` at (3) in entity `S.Base` can't be redirected to `S.Target`, because `S.Target` does not project element `ID` (4).

`toTarget` (1) is a managed association and hence foreign keys are inferred for it. The compiler generates a foreign key `ID`, which corresponds to element `ID` of `model.Target` (2).

As both entities are exposed in service `S`, the compiler tries to redirect `S.Base:toTarget` to an entity inside the same service, to create a "self-contained" service. It notices, however, that `S.Target` does not have element `ID`, and therefore can't match the foreign key to a target element and emits this error message.

How to Fix

If you don't need to expose association `toTarget` in `S.Target`, you can exclude it in the projection via an `excluding` clause.

cds
```
service S {
  entity Base as projection on model.Base
    excluding { toTarget };
  // ...
}
```

If the association is required in the service, you need to either project element `ID` in `S.Target`, or redirect the association explicitly.

The easiest fix is to select `ID` explicitly:

cds
```
service S {
  // ...
  entity Target as projection on model.Target {
    field, ID, // Explicitly select element ID
  };
}
```

However, if you don't want to expose `ID`, redirect association `toTarget` explicitly, matching the foreign key to another element:

```cds
service S {
  entity Base as projection on model.Base {
    ID,
    toTarget : redirected to Target { fakeID as ID }, // (1)
  };
  entity Target as projection on model.Target {
    calculateKey() as fakeID : UUID,                  // (2)
    field,
  };
}
```

Note that at (1), we use element `fakeID` of `S.Target` as foreign key `ID`. That changes its semantic meaning and may not be feasible in all cases! In the example, we assume at (2) that a key can be calculated.

Related Messages

- `rewrite-not-supported`

---

syntax-expecting-unsigned-int

The compiler expects a safe non-negative integer here. The last safe integer is `2^53 - 1` or `9007199254740991`.

A safe integer is an integer that fulfills all of the following:

- Can be exactly represented as an IEEE-754 double precision number.
- The IEEE-754 representation cannot be the result of rounding any other integer to fit the IEEE-754 representation.

The message's severity is `Error`.

Example

Erroneous code example:

```cds
type LengthIsUnsafe : String(9007199254740992); // ❌
type NotAnInteger : String(42.1);               // ❌
```

In the erroneous example, the string length for the type `LengthIsUnsafe` is not a safe integer. It is too large. Likewise, the string length for the type `NotAnInteger` is a decimal.

How to Fix

You have to provide a safe integer:

```cds
type LengthIsSafe : String(9007199254740991);
type AnInteger : String(42);
```

At other places, using unsafe integers (or non-integer numbers) is allowed:

- Annotation values: The value is then simply a string.
- Expressions: The `val` property in the CSN contains a string having a sibling `literal: 'number'`.

---

type-missing-enum-value

An enum definition is missing explicit values for one or more of its entries.

Enum definitions that aren't based on string-types do not get implicit values. They have therefore to be defined explicitly in the model.

The message's severity is `Warning` and is raised by the compiler. You need to adapt your model to fix the warning.

Example

Erroneous code example:

```cds
entity Books {
  // …
  category: Integer enum {
    Fiction; // ❌
```

```
    Action;  // ❌
    // …
  } default #Action;
};
```

Both entries `#Fiction` and `#Action` of the enum `category` are missing an explicit value. Because the base type `Integer` is not a string, no implicit values are defined for them.

How to Fix

Explicitly assign a value or change the type to a string if the values are not important in your model. The erroneous example above can be changed to:

```cds
entity Books {
  // …
  category: Integer enum {
    Fiction = 1;
    Action = 2;
    // …
  } default #Action;
};
```

Background

Many languages support implicit values for integer-like enums. However, CAP CDS does not have this feature, because otherwise, if values are persisted, adding a new entry in-between existing ones would lead to issues during deserialization later on.

Assume that CAP would assign implicit values for integer enums. If a new value were to be added between `Fiction` and `Action` in the erroneous example above, then the generated SQL statement for entity `Books` would change:
Instead of default value `2`, value `3` would be persisted. Without data migration, existing action books would have changed their category.

To avoid this scenario, always add explicit values to enums.

---

type-unexpected-foreign-keys

Foreign keys were specified in a composition-of-aspect.

Compositions of aspects are managed by the compiler. Specifying a foreign key list is not supported. If you need to specify foreign keys, use a composition of an entity instead.

The message's severity is `Error`.

Example

Erroneous code example:

```cds
aspect Item {
  key ID : UUID;
  field : String;
};
entity Model {
  key ID : UUID;
  Item : Composition of Item { ID }; // ❌
};
```

`Item` is an aspect. Because an explicit list of foreign keys is specified, the compiler rejects this CDS snippet. With an explicit foreign key list, only entities can be used, but not aspects.

How to Fix

Either remove the explicit list of foreign keys and let the compiler handle the composition, or use a composition of entity instead.

```cds
aspect Item {
  key ID : UUID;
  field : String;
};
entity Model {
  key ID : UUID;
  Item : Composition of Model.Item { ID }; // ok
};
entity Model.Item : Item { };
```

The snippet uses a user-defined entity, that includes the aspects.

Related Messages

- *type-unexpected-on-condition*

---

type-unexpected-on-condition

An ON-condition was specified in a composition-of-aspect.

Compositions of aspects are managed by the compiler. Specifying an ON-condition is not supported. If you need to specify an ON-condition, use a composition of an entity instead.

The message's severity is *Error* .

Example

Erroneous code example:

```cds
aspect Item {
  key ID : UUID;
  field : String;
};
entity Model {
  key ID : UUID;
  Item : Composition of Item on Item.ID = ID; // ❌
};
```

Item is an aspect. Because an ON-condition is specified, the compiler rejects this CDS snippet. With an ON-condition, only entities can be used, but not aspects.

How to Fix

Either remove the ON-condition and let the compiler handle the composition, or use a composition of entity instead.

```cds
aspect Item {
  key ID : UUID;
  field : String;
};
entity Model {
```

```
  key ID : UUID;
  Item : Composition of Model.Item on Item.ID = ID; // ok
};
entity Model.Item : Item { };
```

The snippet uses a user-defined entity, that includes the aspects.

Related Messages

- *type-unexpected-foreign-keys*

---

wildcard-excluding-one

You're replacing an element in your projection, that is already included by using the wildcard `*`.

The message's severity is *Info*.

Example

Erroneous code example:

cds

```
entity Book {
  key  id : String;
     isbn : String;
  content : String;
};
entity IsbnBook as projection on Book {
  *,
  isbn as id, // ❌
};
```

IsbnBook:id replaces *Book:id*, which was included in *IsbnBook* through the wildcard `*`.

How to Fix

Add the replaced element to the list of wildcard excludes:

```cds
entity IsbnBook as projection on Book {
  *,
  isbn as id
} excluding { id };
```

Last updated: 11/12/2025, 15:30

Previous page
Common Annotations

Next page
Aspect-oriented Modelling

Was this page helpful?