An NEC Company

Avaloq Script

Customization Upgrade Guide
Release 5.7
Area: Technical Base

Latest version of this document

The latest version of this document can be found at https://docs.avaloq.com

Feedback

Please send any feedback to documentation@avaloq.com

Version history

Version / Date	Section	Description of the change
5.7v0 / 7 June 2023		New document for ACP 5.7

Contents

1 Introduction

This document describes changes that you may need to make to your customization when you upgrade to Avaloq Core Release 5.7, as a result of enhancements to Avaloq Script in that release.

Previously in Avaloq Script, the "id table type" data type was always translated to the PL/SQL data type as follows (see section 5.6 of *Avaloq Script Language Reference – Reference Guide (doc ID: 1074)*):

- varchar2(2000) – for variables and constants
- varchar2 – for parameters

Starting in Avaloq Core Release 5.7, the Avaloq Script "id table type" will be translated to something more specific for tables that have a primary key that consists of one column whose type is numeric.

In Avaloq Core Release 5.7, an "id table type" in Avaloq Script of any code table, or of the tables MSG, DOC, or OBJ are translated as follows.

This Avaloq Script data type . . .	Is translated to this PL/SQL data type . . .
id table code table	pls_integer
id table msg	number(12)
id table doc	number(12)
id table obj	pls_integer

1.1 Results of this change

As a result of this change, some code modifications may be necessary during the release upgrade.

Passing text to an id table with a number type

Any context where these types are expected (assignments, function calls, comparisons, function returns) cannot contain a text value. This would lead to a runtime exception, unless the text value can be converted to a number.

Where possible without changing behaviour, the compiler will prevent runtime exceptions.

Where it is not possible, a warning is provided by ASMD and the Compiler Framework to indicate where runtime exceptions might occur. In these cases, action may need to be taken by adapting customization code (note that often these warnings will highlight existing business logic or programming mistakes).

Because these types are now numeric types, they are treated like a number, including in comparisons.

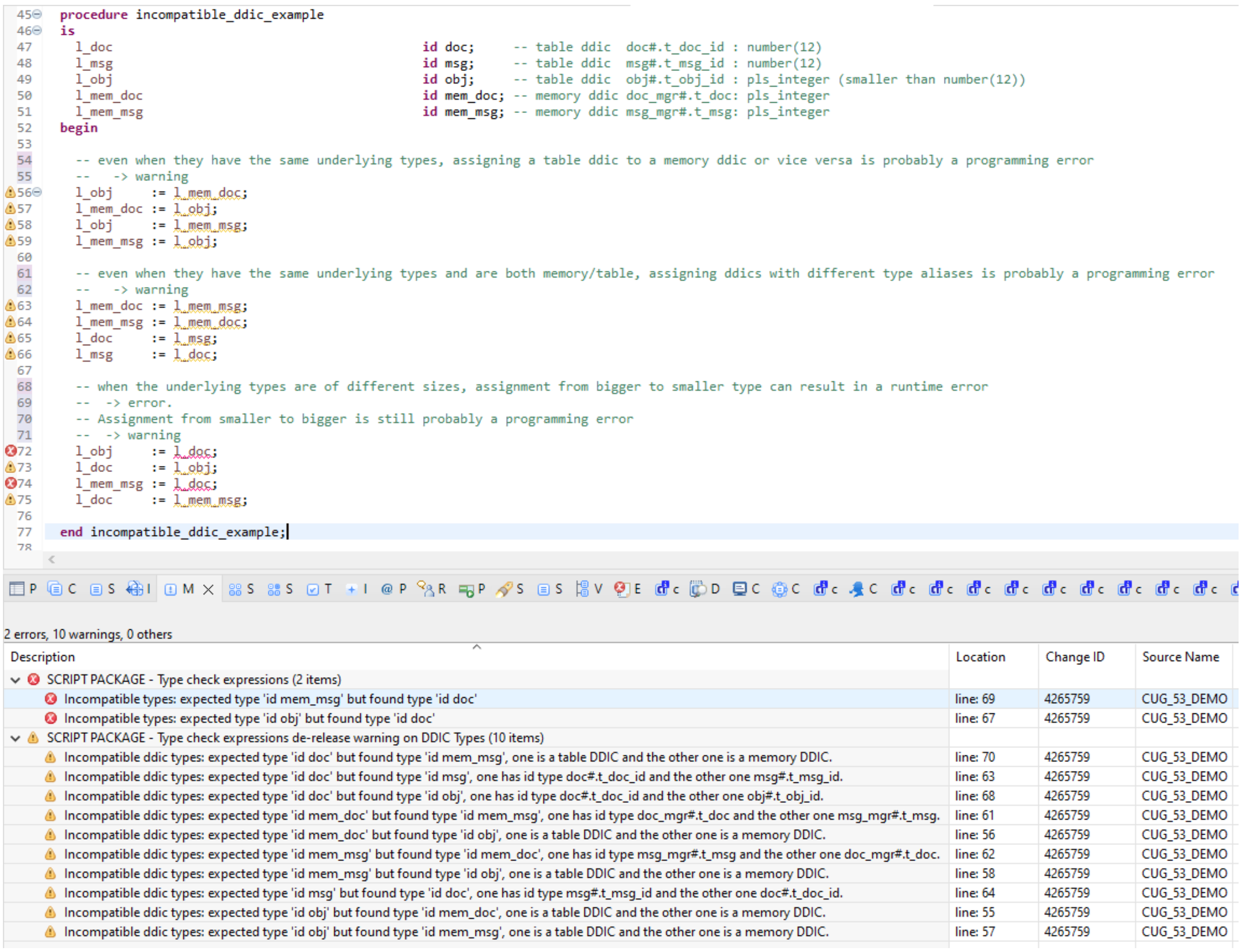
Action to be taken:

- Analyse occurrences of the "Potential runtime type conversion error" warning during release upgrade
- If the text value is known to contain a number (for example, a value from a textual API or from a list stored in lib_co), no action is needed and the warning can be ignored
- If the text value could actually be text, the custom code should be adapted (for example, change the type of the containing variable, or change the value to an appropriate numeric value)

Passing id table types of different types to each other

An id type of MEM_DOC, MEM_MSG or OBJ might not be able to hold an id type of DOC or MSG, because the PL/SQL types for MEM_DOC, MEM_MSG or OBJ have a smaller range than the ones for DOC and MSG. So a compile-time error might also arise if the types are misused, that is, if a type with a smaller range is assigned from a type with a larger range (for example, assigning a DOC to a MEM_DOC).

The compiler will issue a warning for any other assignment between an id type of a memory DDIC and a table DDIC, or any assignment of id types with different PL/SQL types.



```
45  procedure incompatible_ddic_example
46  is
47    l_doc                                      id doc;     -- table ddic  doc#.t_doc_id : number(12)
48    l_msg                                      id msg;     -- table ddic  msg#.t_msg_id : number(12)
49    l_obj                                      id obj;     -- table ddic  obj#.t_obj_id : pls_integer (smaller than number(12))
50    l_mem_doc                                  id mem_doc; -- memory ddic doc_mgr#.t_doc: pls_integer
51    l_mem_msg                                  id mem_msg; -- memory ddic msg_mgr#.t_msg: pls_integer
52  begin
53
54    -- even when they have the same underlying types, assigning a table ddic to a memory ddic or vice versa is probably a programming error
55    --    -> warning
56    l_obj     := l_mem_doc;
57    l_mem_doc := l_obj;
58    l_obj     := l_mem_msg;
59    l_mem_msg := l_obj;
60
61    -- even when they have the same underlying types and are both memory/table, assigning ddics with different type aliases is probably a programming error
62    --    -> warning
63    l_mem_doc := l_mem_msg;
64    l_mem_msg := l_mem_doc;
65    l_doc     := l_msg;
66    l_msg     := l_doc;
67
68    -- when the underlying types are of different sizes, assignment from bigger to smaller type can result in a runtime error
69    --   -> error.
70    -- Assignment from smaller to bigger is still probably a programming error
71    --   -> warning
72    l_obj     := l_doc;
73    l_doc     := l_obj;
74    l_mem_msg := l_doc;
75    l_doc     := l_mem_msg;
76
77  end incompatible_ddic_example;
78
```

2 errors, 10 warnings, 0 others

Description	Location	Change ID	Source Name
SCRIPT PACKAGE - Type check expressions (2 items)			
Incompatible types: expected type 'id mem_msg' but found type 'id doc'	line: 69	4265759	CUG_53_DEMO
Incompatible types: expected type 'id obj' but found type 'id doc'	line: 67	4265759	CUG_53_DEMO
SCRIPT PACKAGE - Type check expressions de-release warning on DDIC Types (10 items)			
Incompatible ddic types: expected type 'id doc' but found type 'id mem_msg', one is a table DDIC and the other one is a memory DDIC.	line: 70	4265759	CUG_53_DEMO
Incompatible ddic types: expected type 'id doc' but found type 'id msg', one has id type doc#.t_doc_id and the other one msg#.t_msg_id.	line: 63	4265759	CUG_53_DEMO
Incompatible ddic types: expected type 'id doc' but found type 'id obj', one has id type doc#.t_doc_id and the other one obj#.t_obj_id.	line: 68	4265759	CUG_53_DEMO
Incompatible ddic types: expected type 'id mem_doc' but found type 'id mem_msg', one has id type doc_mgr#.t_doc and the other one msg_mgr#.t_msg.	line: 61	4265759	CUG_53_DEMO
Incompatible ddic types: expected type 'id mem_doc' but found type 'id obj', one is a table DDIC and the other one is a memory DDIC.	line: 56	4265759	CUG_53_DEMO
Incompatible ddic types: expected type 'id mem_msg' but found type 'id mem_doc', one has id type msg_mgr#.t_msg and the other one doc_mgr#.t_doc.	line: 62	4265759	CUG_53_DEMO
Incompatible ddic types: expected type 'id mem_msg' but found type 'id obj', one is a table DDIC and the other one is a memory DDIC.	line: 58	4265759	CUG_53_DEMO
Incompatible ddic types: expected type 'id msg' but found type 'id doc', one has id type msg#.t_msg_id and the other one doc#.t_doc_id.	line: 64	4265759	CUG_53_DEMO
Incompatible ddic types: expected type 'id obj' but found type 'id mem_doc', one is a table DDIC and the other one is a memory DDIC.	line: 55	4265759	CUG_53_DEMO
Incompatible ddic types: expected type 'id obj' but found type 'id mem_msg', one is a table DDIC and the other one is a memory DDIC.	line: 57	4265759	CUG_53_DEMO

Action to be taken:

- Often, assigning different DDIC types to each other is a programming or logic error that you should fix. If it's not, add a type cast.