SAP - ABAP for SAP HANA Development User Guide | CUSTOMER

%NW-ASABAP-LONG-7.55% FPS00 and SAP Cloud Platform ABAP Environment

Document Version: 3.16 – 2021-02-03

SAP - ABAP for SAP HANA Development User Guide

Client Version 3.16

Content

1 About the ABAP for SAP HANA Development User Guide. 3

2 Concepts. 4

2.1 Database Procedures 4

ABAP Managed Database Procedures (AMDP). 4

2.2 AMDP Profiling. 6

Understanding AMDP Profiling Results. 8

3 Tasks. 13

3.1 Fundamental Tasks and Tools. 13

Previewing Results of ABAP Managed Database Procedures (AMDP). 14

3.2 Using Troubleshooting Tools. 15

Working with the AMDP Debugger. 16

Working with the AMDP Profiler. 33

4 What's New in ABAP for SAP HANA Development. 36

4.1 Version 3.6. 36

4.2 Version 3.4. 37

4.3 Version 3.0. 38

1 About the ABAP for SAP HANA Development User Guide

Scope of Documentation

☁ This documentation describes the functionality and the usage of tools for integration of native SAP HANA objects (in the context of SAP HANA repository content) within the ABAP layer.

Context

This guide provides documentation about features which are client-specific or require a specific back-end version.

Consequently, this documentation covers all client-specific and back-end-specific dependencies.

To highlight and contrast back-end-specifics in the relevant context, the following icons are used:

- ☁ for SAP Cloud Platform ABAP Environment shipments

Target Audience

ABAP developers who develop content for ABAP-based applications that are optimized for SAP HANA databases.

Validity of Documentation

This documentation belongs to ABAP Development Tools **client version 3.14** and refers to the range of functions that have been shipped as part of the standard delivery for:

- ☁ SAP Cloud Platform ABAP Environment

2 Concepts

2.1 Database Procedures

The SAP HANA database comes with a variety of programming options for application logic at the database level.

The database procedures (which are used to implement the application logic) can be written as queries that follow the SAP HANA database SQLScript syntax. Database procedures can have multiple input parameters and output parameters; these are either of scalar (such as `integer, double, varchar`) or table type.

Using ADT tools, you can implement SAP HANA database procedures by means of...

- ABAP Managed Database Procedures (AMDP) [page 4]
  The basic idea of AMDP is to manage SAP HANA procedures and their lifecycle inside the ABAP server. To allow native consumption of SAP HANA features from within the ABAP layer, the SAP HANA database procedure language SQLScript has been integrated into the ABAP stack. AMDP is implemented in ABAP class methods (so-called AMDP methods) that serve as a container for SQLScript code. This approach offers many significant advantages:
  - It enables the shipment of AMDP in the same way as any other ABAP development object (lifecycle management)
  - It allows you to implement and ship corrections for AMDPs, just like it is possible for ABAP classes, including SAP Note support (supportability and extensibility).

2.1.1 ABAP Managed Database Procedures (AMDP)

ABAP Managed Database Procedures is one of the recommended patterns for use in ABAP code optimization within the context of ABAP development on SAP HANA.

Basics

AMDPs allow you as an ABAP developer to write database procedures directly in ABAP. Special ABAP classes (so-called AMDP classes) can contain embedded code (SQLScript) that is used to generate DB procedures in the SAP HANA DB layer.

The basic idea of AMDP is to manage SAP HANA procedures and their lifecycle inside the AS ABAP server. To allow native consumption of SAP HANA features from within the ABAP layer, the SAP HANA database procedure language SQLScript has been integrated into the ABAP stack. AMDP is implemented in ABAP class methods (so-called AMDP methods) that serve as a container for SQLScript code.

This approach offers many significant advantages:

- It enables the shipment of AMDP in the same way as any other ABAP development object (lifecycle management)

Editor

ABAP Managed Database Procedures (AMDP) are the preferred way for developing SAP HANA DB procedures on the ABAP platform.

Since ABAP Managed Database Procedures are implemented as methods of a global ABAP class, the editing environment for AMDP is the ABAP class editor. In concrete terms, an ABAP Managed Database Procedure is written in a database-specific language, such as Native SQL or SQL Script, and is implemented within an AMDP method body of an AMDP class. So, developing a database procedure is similar to the editing of ABAP class methods with the same tool environment.

Syntax of AMDP Classes

An AMDP is implemented in an AMDP class with a regular static method or instance method in any visibility section.

```
CLASS <my_amdp_class> DEFINITION.
  PUBLIC SECTION.
* Marker interface with SAP HANA DB as database type
  INTERFACES IF_AMDP_MARKER_<DB_TYPE>.
   ...
  METHODS <my_amdp_method>.
   ...
ENDCLASS.
CLASS <my_amdp_class> IMPLEMENTATION.
...
* AMDP method
  METHOD <my_amdp_method> BY DATABASE PROCEDURE
        FOR <db_type>
        LANGUAGE <db_language>
         OPTIONS <db_options>
         USING   <db_entity>.

      "Implementation of the procedure in a DB-specific language
     ...
  ENDMETHOD.
    ...
ENDCLASS.
```

Accessing Help on AMDP

- Visit the SAP help portal for detailed help on AMDP . The documentation
  - guides you through the concepts of AMDP,
  - describes the language syntax,
  - informs you about restrictions

- Alternatively, you can access the AMDP help by pressing `F1` in the context of AMDP method implementation in the ABAP source editor.

Tool Support

Along the lines of other ABAP for SAP HANA tools, the editing of ABAP Managed Database Procedures is only supported in the Eclipse-based development environment (ABAP Development Tools)

2.2 AMDP Profiling

AMDP profiling enables you to measure and analyze the performance of ABAP Managed Database Procedures (AMDP) and the executed SQL statements.

Overview

AMDP Profiler has been fully integrated into the ABAP Profiler of ABAP Development Tools (ADT).

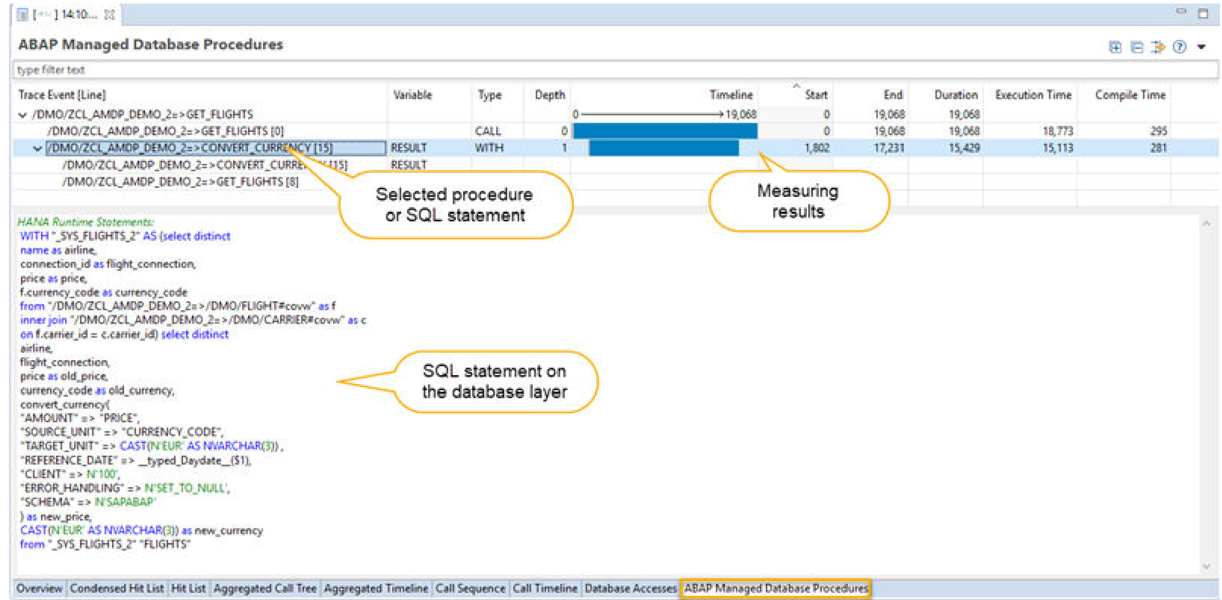
You have the following possibilities to trigger the AMDP Profiler from the source code of an AMDP:

- *Context menu*: Choose `Profile as` + `ABAP Application (Console)`
- *Context menu*: Choose `Profile as` + `ABAP Unit Test`
- *Menu*: Choose `Run` + `Profile ABAP Development Object...`
- from the *ABAP Trace Requests* view.

> **i Note**
>
> To enable the AMDP Profiler, you have to select the *Enable AMDP trace* checkbox in the profile configuration.

The result of the AMDP profiling is contained in the ABAP trace and can be opened from the AMDP trace options*ABAP Trace* view.

The analysis results are provided in the *ABAP Managed Database Procedures* tab. Here, a tabular view with the details is displayed.



Example for displaying and analyzing the ABAP trace of an AMDP

The *Trace Event* column represents the structure of the AMDP execution.

The tree structure does not represent a call hierarchy. Each line can represent, for example, one of the following:

- A top level call from ABAP into an AMDP
- A procedure call from AMDP to an AMDP
- The execution of a SQL statement
- A source code position (see SQL Inlining [page 10])

In addition, the SQL statement is displayed which was executed at runtime on the SAP HANA database.

> **i Note**
>
> To get more information about SQLScript basics, which is essential for working with the AMDP Profiler, see the subsequent documentation.

Features

AMDP Profiler in ADT provides the following features:

- You can measure the time which was spent by SQL statements and procedure calls.
- It includes dynamic SQL statements using `EXEC`.
- It is fully integrated within ADT by, for example, supporting navigation to AMDP source code.
- It enables basic filtering, sorting, file exports, and so on.

Enablement

The settings and configuration options of AMDP Profiler are also fully integrated into the ABAP Profiler.

You have the following options to enable AMDP profiling:

- To define if an ABAP trace also comprises an AMDP trace, you need to select the *Enable AMDP trace* checkbox in the *AMDP trace options* section from the *Trace Parameters* window.
- To predefine AMDP Profiler for each execution by default, open the *Window* > *Preferences* > *ABAP Development* > *Profiling* preference page and select the *Enable AMDP trace* checkbox in the *AMDP trace options* section.

Restrictions

AMDP Profiler does not support the following:

- Fine granular information about single SQLScript elements such as `loops`, `if`, and `calculation`
- Calculation Engine (CE) functions
- ABAP CDS table functions that are called using ABAP SQL
- Source code information about certain SQL statements

Related Information

ABAP Managed Database Procedures (AMDP) [page 4]
Working with the AMDP Profiler [page 33]
Understanding AMDP Profiling Results [page 8]

2.2.1 Understanding AMDP Profiling Results

The following sections provides basic knowledge about SQLScript which is required to understand the results of AMDP profiler.

SQLScript Optimizer

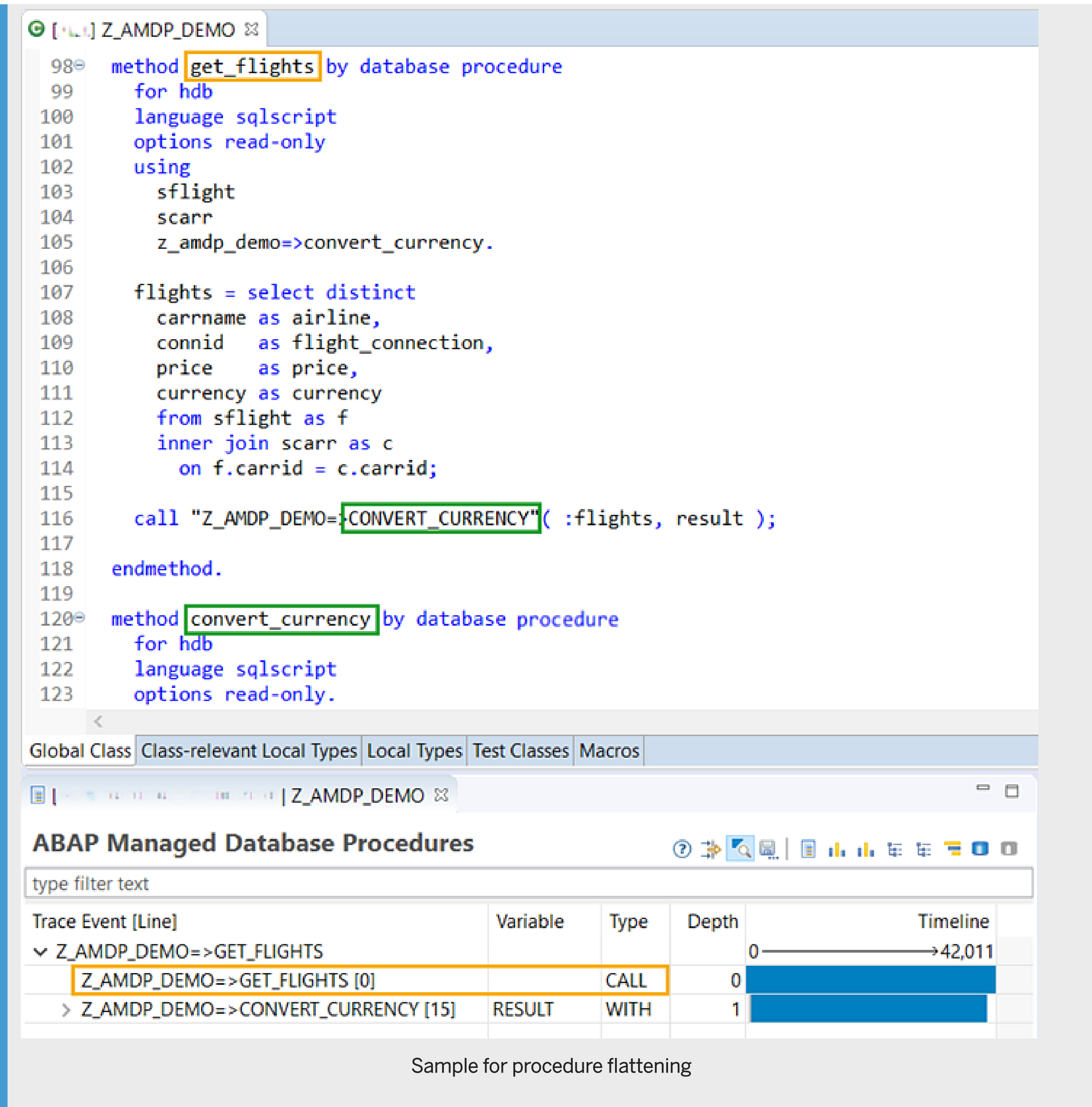
SQLScript is designed to provide superior optimization options. Therefore, the SQLScript code written at design time is adjusted by the SQLScript optimizer before execution at runtime.

These optimizations may contain:

Procedure Flattening

This optimization describes how multiple procedures at design time are combined by SAP HANA into a single procedure at runtime.

> **Example**
>
> | Written code | |
> | --- | --- |
> | procedure p1
`call p2();`
procedure p2
`select...` | procedure p1
`select...` |
>
> In the table 1, procedure **p1** calls procedure **p2**. The SQLScript optimizer puts the code of **procedure p2** inside **procedure p1** and removes the call during procedure flattening. Therefore, it is possible that during runtime fewer procedures are called than written in the SQLScript code at design time.
>
> AMDP Profiler shows what actually is executed at runtime. Consequently, it might show fewer procedure calls than written in the code. However, the content of the flattened procedures (for example, SQL statements) is still shown in the profiling results.
>
> In the following sample, the procedure `convert_currency` was flattened into procedure `get_flights` by the SQLScript optimizer. Therefore, the AMDP profiler shows only the call of the `get_flights` procedure and does not show call of the `convert_currency` procedure.



Sample for procedure flattening

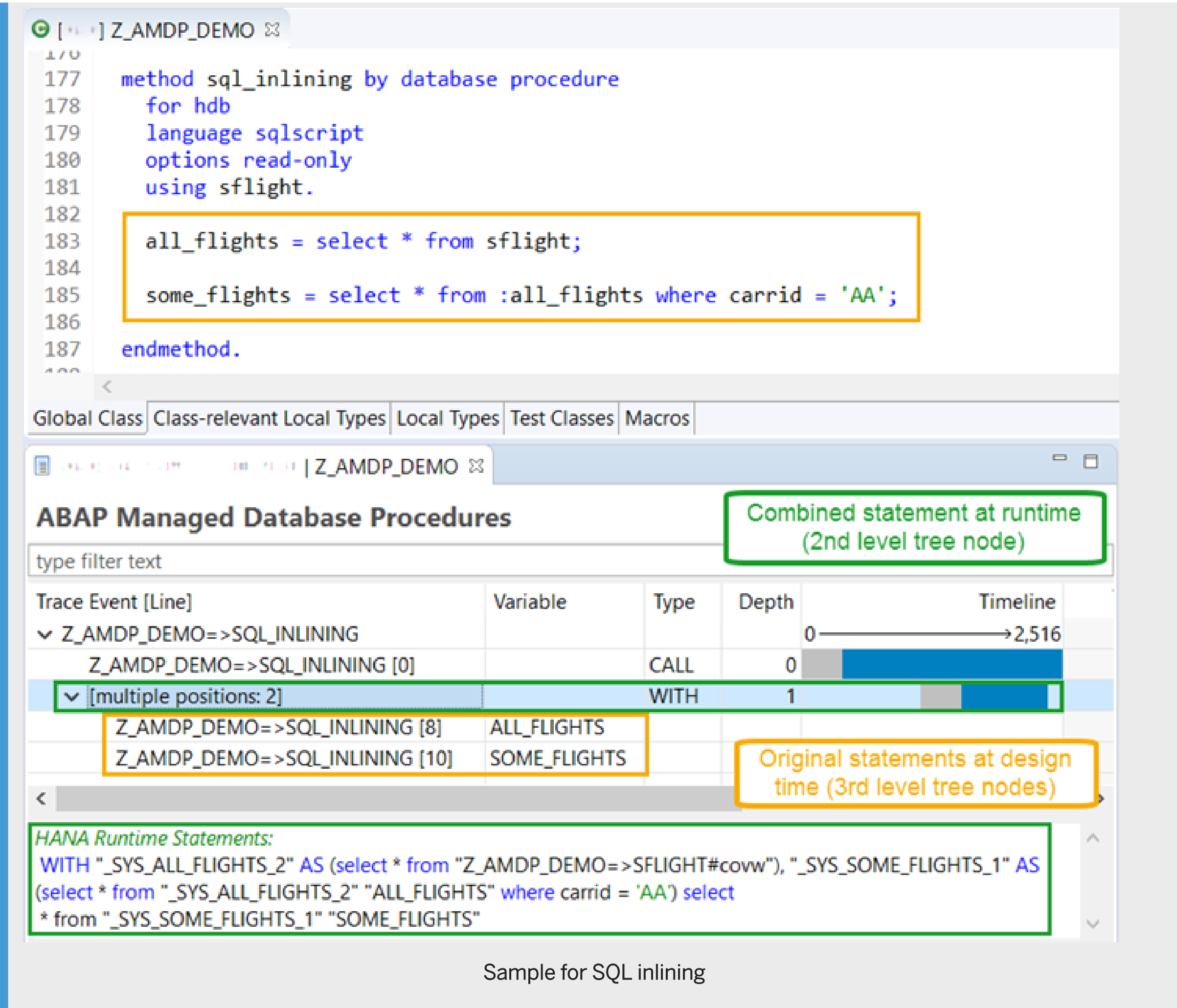
SQL Inlining

It may be that multiple SQL statements at design time are combined into a single SQL statement at runtime for improving performance.

AMDP Profiler shows the combined SQL statement that was executed at runtime as a second level tree node and the full statement below. The code positions of the original SQL statements at design time are shown as third level tree nodes.

Example

In the following sample, the two statements `all_flights` and `some_flights` are combined into one statement.



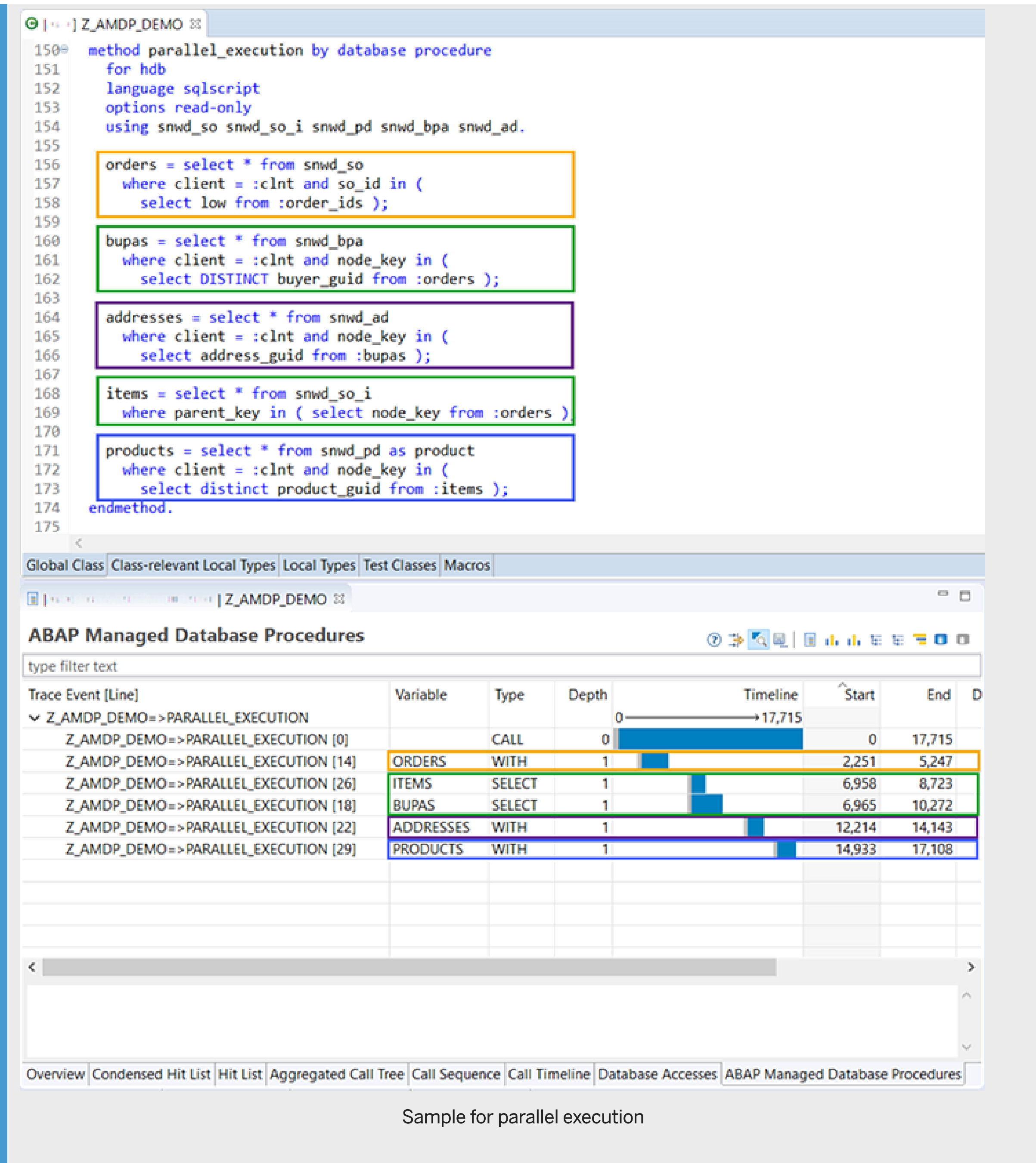
Sample for SQL inlining

Parallel Execution

It may be that SQL statements or procedure calls are executed in parallel threads and in a different order than written in the source code (if the order is semantically not relevant).

Example

In the following sample, the select statements for the values `bupas` and `items` are processed in parallel. Both depend on the value `orders`. The execution order at runtime differs from the order within the source code. Note that there is no semantic difference, because dependencies between statements are always respected.



Sample for parallel execution

3 Tasks

3.1 Fundamental Tasks and Tools

Overview

SAP HANA is a relational database management system (RDBMS). This platform combines an in-memory database and its application layer for data-intensive logic.

Your ABAP environment runs on an Application Server ABAP which is the base for all SAP products. Here are all development objects stored and all development activities and executions performed.

ABAP is SAP's proprietary language to develop business applications within the AS ABAP on the ABAP platform. ABAP SQL defines the ABAP statements of the SQL subset which can be used as interface to access the database of an AS ABAP.

So, ABAP for SAP HANA describes the code push-down of data-intensive logic from the ABAP platform layer to the SAP HANA database layer using ABAP Managed Database Procedures (AMDP).

ADT Tools and Features

The subsequent tools enable you to benefit from the capabilities of SAP HANA by integrating native SAP HANA objects in the ABAP platform layer.

Related Information

Previewing Results of ABAP Managed Database Procedures (AMDP) [page 14]

3.1.1 Previewing Results of ABAP Managed Database Procedures (AMDP)

After you have implemented or consumed an AMDP in an ABAP code, you need to quickly execute and preview the results of the implemented AMDP.

Context

A preview helps you to:

- ensure that the AMDP provides the expected results
- evaluate the performance of applications while using AMDPs

Since AMDPs are required in day-to-day development activities, integration of the Data Preview tool within ABAP Development Tools (ADT) is required to facilitate faster development and easier consumption of AMDPs.

> **i Note**
>
> - Data Preview currently supports AMDP methods only with public visibility
> - This is a simple tool that enables you to quickly see the glue code needed for consuming the AMDP method, execute AMDP methods, and view the results. Hence, the generated glue code in the case of AMDP classes with complex constructor signatures (with object references and deep structures) does not have the complete code for creating objects.

Procedure

1. In the *Project Explorer* view, choose an *ABAP Project*.
2. Choose the required ABAP package.
3. Expand the *ABAP package* ▶ *Source Library* ▶ *Classes*.
4. Expand the required class.
5. Choose a public AMDP method.
6. In the method context menu, choose *Data Preview*.
   The AMDP glue code editor and an empty result set appear.

   > **i Note**
   >
   > The table below describes the various features available in the glue code editor:

Feature	Description
Check	Use this feature to verify the syntax.
Execute	Use this feature to run the database procedure and view the corresponding result set.

Feature	Description
Max. Rows	Enter a value to display maximum records in the result set. Data Preview considers only this value while displaying records even though the result set from the method execution contains all records.

7. Choose *Execute* to view the result set.
   If required, you can use the glue code editor to provide different input values and preview the corresponding result sets. For procedures containing multiple exporting or importing parameters, you can switch between parameters using the dropdown option in the result set section. The dropdown option appears at the top of the result set.

3.2 Using Troubleshooting Tools

ABAP Debugger

The **ABAP Debugger** enables you to stop a program during runtime and examine the flow and results of each statement during execution. Stepping through an ABAP application with the debugger helps you to detect and correct errors in the source code of a development object.

AMDP Debugger

The **AMDP debugger** enables you to analyze running ABAP Managed Database Procedures (AMDPs) and CDS table functions.

ABAP Profiler

The **ABAP Profiler** tools show you where runtime is being consumed, and where effort for refactoring and optimization can best be applied. They also let you analyze and understand program flow, which is useful when you are trying to understand a problem or learn about code you need to analyze or maintain.

AMDP Profiler

The **AMDP Profiler** enables you to measure and analyze the performance of ABAP Managed Database Procedures (AMDP) and the executed SQL statements. It has been fully integrated into the ABAP Profiler of ABAP Development Tools (ADT).

Related Information

Working with the AMDP Debugger [page 16]
AMDP Profiling [page 6]
Working with the AMDP Profiler [page 33]

3.2.1 Working with the AMDP Debugger

The AMDP Debugger enables you to analyze running ABAP Managed Database Procedures (AMDPs), CDS table functions, and GraphScript source code.

Prerequisites

- **Database:**
  - AMDPs: SAP HANA DB SPS9, or higher (SPS8 with restricted functionality only)
  - CDS Table Functions: SAP HANA DB SPS11, or higher
- **ADT Client:**
  - AMDPs: Client version 2.36 or higher.
  - CDS Table Functions: Client version 2.68 or higher.

Overview

The AMDP Debugger is part of the ADT client installation. It allows you to debug the embedded AMDP code, CDS table functions, and GraphScript within ADT. It is not part of the ABAP Debugger.

The differences are as follows:

- ABAP Debugger is used to debug the execution of ABAP programs running on the AS ABAP.
- AMDP Debugger is used to debug the execution of DB procedures running on SAP HANA DB.

Features

The AMDP Debugger provides you with the following basic functions:

- Activation of the AMDP Debugger
- Setting breakpoints
- Stepping
- Viewing variables
- Viewing table contents

- Deactivation of the AMDP Debugger.

Related Information

ABAP Managed Database Procedures (AMDP) [page 4]
GraphScript Language
Parallel Execution of Procedures in the AMDP Debugger [page 23]
Video: How to debug an ABAP Managed Database Procedure (AMDP)
Tutorial: How to Debug an ABAP Managed Database Procedure

3.2.1.1 Activating AMDP Debugger

Prerequisites

- To activate the AMDP Debugger in ABAP Development Tools, you need the standard authorization profile to debug ABAP programs.

  > → Remember
  >
  > There is no need for extra DB user or other authorizations.

- Debugging is enabled (default setting) in the debugger settings. **More on this**: AMDP Debugger Settings [page 19]

Procedure

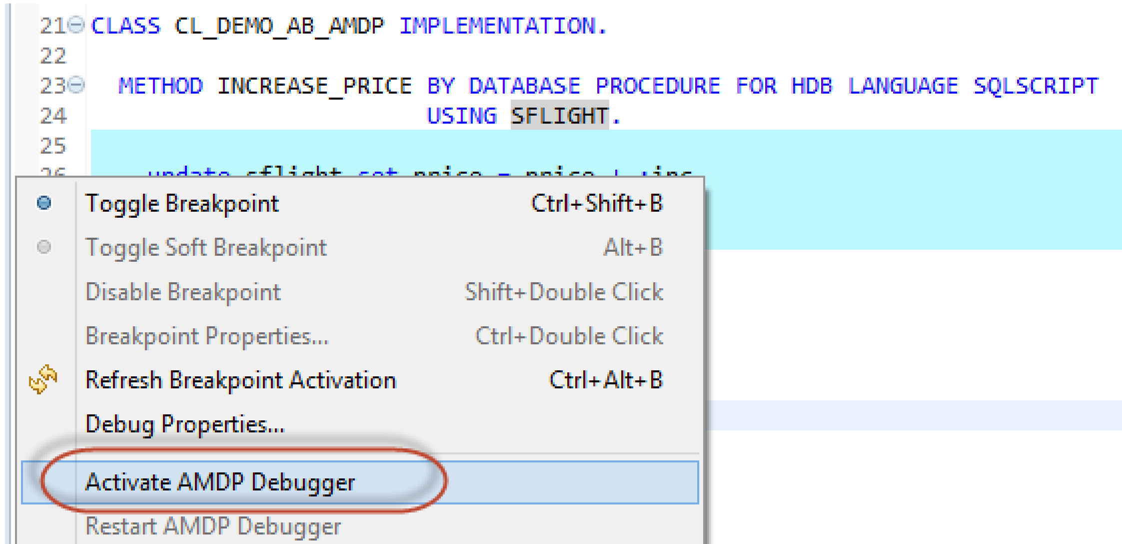
Unlike what you thus far know from the ABAP Debugger, you must first activate the AMDP Debugger in an ABAP project before using it. This means that you must activate the AMDP Debugger before you can stop the execution of a database procedure, perform stepping, or inspect variable values.

The debugger activation can be operated in two ways:

- By **explicit activation** of the AMDP Debugger
- **Implicitly** by setting breakpoints in AMDP methods.

To activate the AMDP Debugger explicitly, proceed as follows:

1. Open an AMDP class in the relevant ABAP project.
2. Position the cursor within the ruler (left bar) of the source editor at the line that contains the AMDP code (for example, SQLScript code) you are interested in.
3. Choose *Activate AMDP Debugger* from the context menu.



Explicit activation

To activate the AMDP Debugger implicitly, follow the link below:

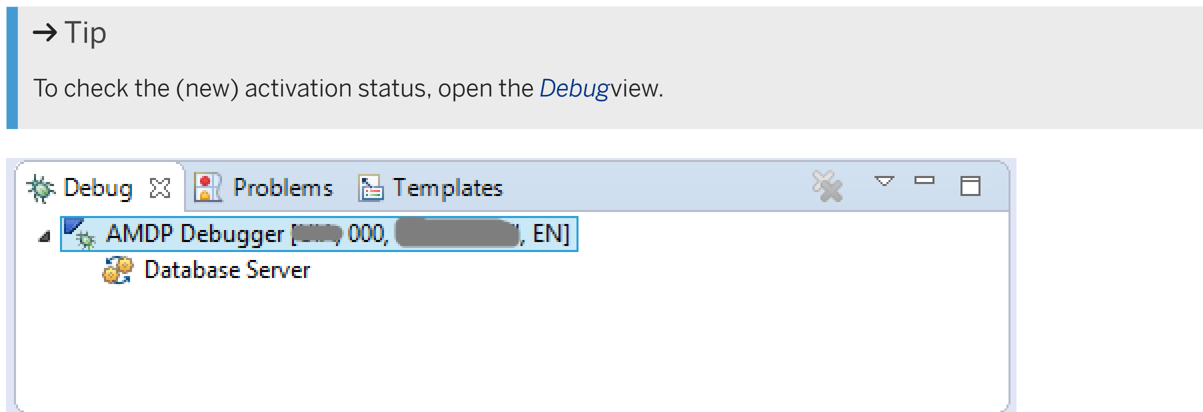
- Setting AMDP Breakpoints [page 20]

Results

A message informs you that the AMDP Debugger has been activated. This activation enables AMDP debugging within the current ABAP project.

→ Tip

To check the (new) activation status, open the *Debug*view.



Debugger status is displayed n the Debug view

Related Topics

- Setting AMDP Breakpoints [page 20]
- Status of the AMDP Debugger [page 22]
- Deactivating AMDP Debugger [page 32]

3.2.1.1.1 AMDP Debugger Settings

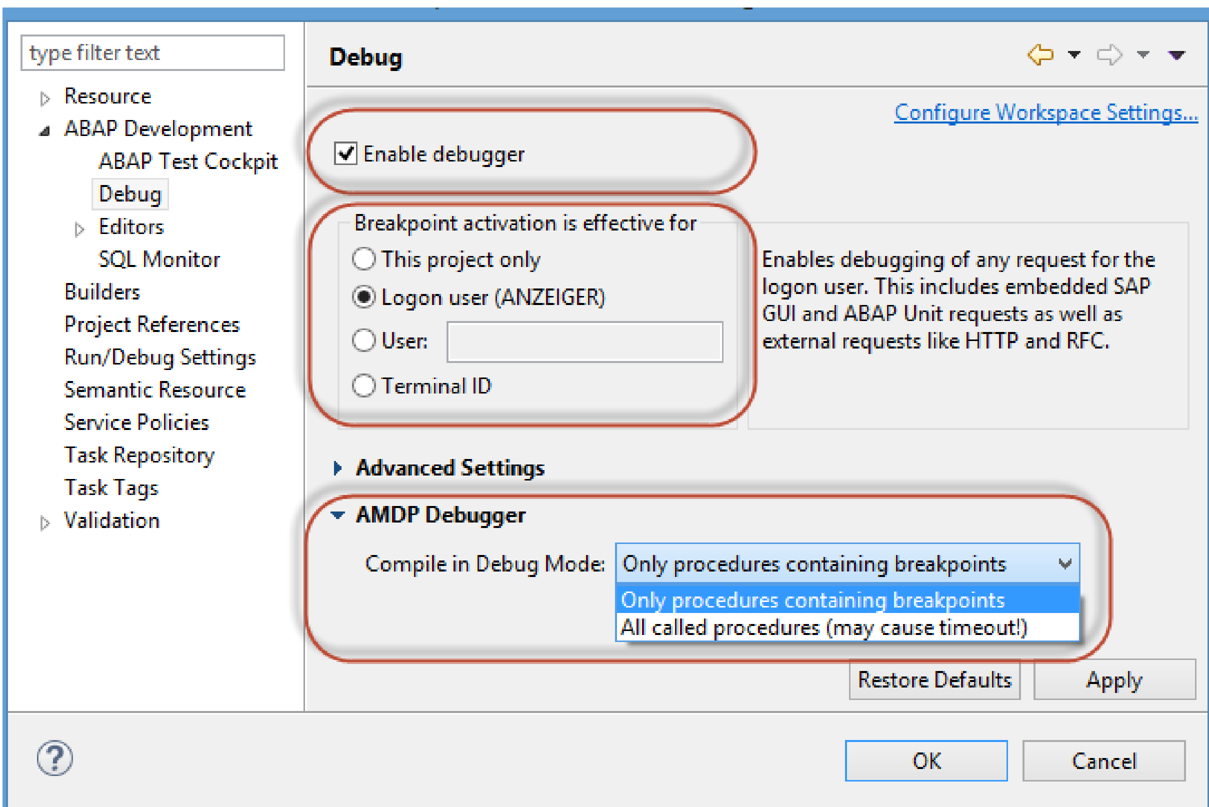
Project-Specific AMDP Settings

To view or change the AMDP Debugger settings, proceed as follows:

1. Open the debugger preference page. (Choose *Window* > *Preferences* > *ABAP Development* > *Debug*).
2. Choose the link *Configure Project Specific Settings...* and select the ABAP Project for which you wish to change (view) the debug settings.

On the Debug dialog, you can choose between two *Debug Mode* options for the AMDP Debugger:

- *Only procedures containing breakpoints*
- *All called procedures*



Settings that are relevant for AMDP debugging

What's behind the *Debug Mode* settings?

During the execution of AMDPs, every single procedure of the call hierarchy runs either in debug mode or in optimized mode.

- *Optimized mode* improves performance by using mechanisms like parallelization, inlining, and others. These optimizations can potentially interfere with debugging. One not quite realistic but simple example would be a read-only procedure without any exporting parameters. Since such a procedure can never have any effect, it is not executed at all in optimized mode and therefore breakpoints within this procedure would not ever be reached.
- *Debug mode* will prevent such optimizations. Additionally, only procedures in debug mode will recognize breakpoints that are newly created during an active debug session.

The decision regarding which procedures run in debug mode and which in optimized mode can be influenced by the *Compile in Debug Mode* setting:

- *Only procedures containing breakpoints*: The AMDP Debugger starts quickly, but can have optimization side-effects. (Default, necessary for very large procedures / call hierarchies).
- *All called procedures*: No side-effects, but can cause long waiting times for the AMDP Debugger to start up initially. (Recommended for small procedures / call hierarchies).

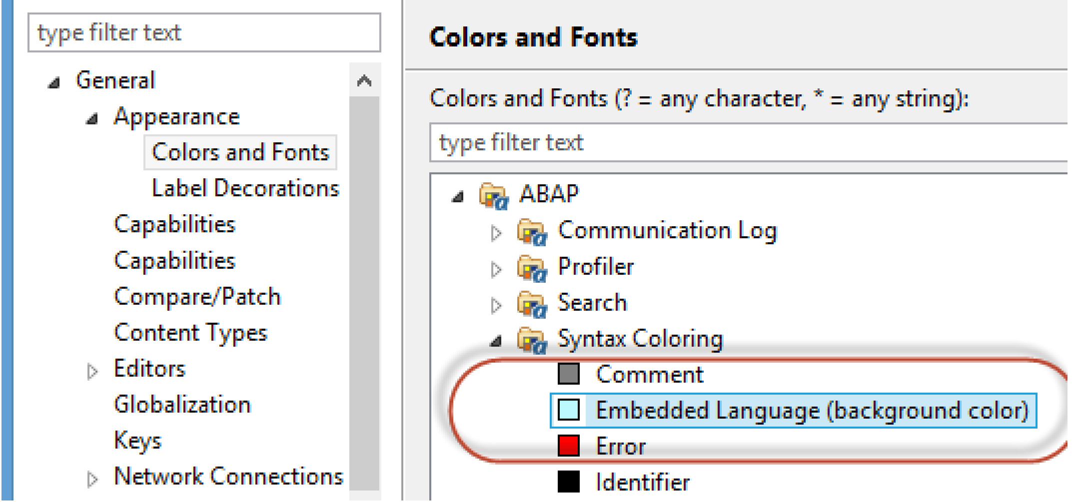
See also: Parallel Execution of Procedures in the AMDP Debugger [page 23]

Useful Preferences for AMDP Debugger

It can be very helpful for you if you work with AMDP classes to highlight those parts of your class that contain embedded AMDP source code. This is not done by default for the source editor so you will have to change the corresponding preference settings.

To change the syntax coloring for AMDP code, proceed as follows:

1. Open the preferences from the menu bar (*Windows* > *Preferences*).
2. Choose *General* > *Appearance* > *Colors and Fonts* .
3. On the *Colors and Fonts* preferences page, choose tree element *ABAP* > *Syntax Coloring* > *Embedded Language* and assign the color of your choice.



Syntax coloring for embedded AMDP code

3.2.1.2 Setting AMDP Breakpoints

Prerequisites

You can set dynamic AMDP breakpoints

- In the active version of an AMDP class
- For any executable statement within the AMDP code.

Procedure

Setting Breakpoints in the Editor ruler Through the Context Menu

1. Position the cursor within the ruler (left bar) of the source editor at the line that contains an executable AMDP code.
2. Choose **Toggle Breakpoint** from the context menu.

Setting Breakpoint in the Editor Ruler Through Double-Click.

1. Within the ruler of the source editor, double-click the line that contains the executable statement you are interested in.

Setting AMDP Breakpoints at Lines of Code

1. Position the cursor on the statement (within the AMDP code) where you want to stop.
2. Then choose the menu option *Run* > *Toggle Breakpoint.*

Results

The AMDP breakpoint is assigned to the line of code where you set it. It stays assigned even if the AMDP debugger has been deactivated in the meantime (for example, after closing the ADT session). However, in that case the breakpoint will become inactive.

> i Note
>
> The status of a breakpoint is indicated by the color.

More on this: Status of the AMDP Debugger [page 22]

> i Note
>
> Bear in mind that the creation of a breakpoint can cause recompilation of DB procedures. Depending on your AMDP Debugger Settings [page 19], only the current and called procedures are affected. Recompilation only happens if a debug version of the respective procedure is not yet available. While the recompilation is taking place, the breakpoint changes its state from inactive (gray) to pending (blue).

Further Activities

Toggle a breakpoint again to delete it. You can also disable breakpoints without deleting them.

You can display and manage AMDP breakpoints in the *Breakpoints* view in the *Debug* perspective.

3.2.1.2.1 Status of the AMDP Debugger

Since the status of the AMDP Debugger is quite important, it is not only visible in the *Debug* view of ABAP Development Tools but also indicated in the ABAP source editor by the color of the breakpoints:

AMDP breakpoint color	Indicates that ...
Green	the AMDP Debugger is active and the AMDP breakpoint is valid and confirmed.
Blue	the AMDP Debugger is active but the AMDP breakpoint is still pending. This means, it is validated but the related DB procedure is not ready yet to be debugged.
Gray	the AMDP Debugger is inactive.

3.2.1.3 Stepping in AMDP Debugger

ABAP Managed Database Procedures (AMDP) are usually nested, which means there are multiple procedures along a call hierarchy. The call stack and the current position within that stack is shown within the *Debug* view in ABAP Development Tools.

If procedures that run in *optimized mode* (see also: AMDP Debugger Settings [page 19]) are part of the call stack, these are gray colored, and it is not possible to show the exact code position of the call. Hence, you will see from which procedure your currently debugged procedure was called, but not the actual line of the call statement. Normal black colored entries indicate procedures in debug mode.

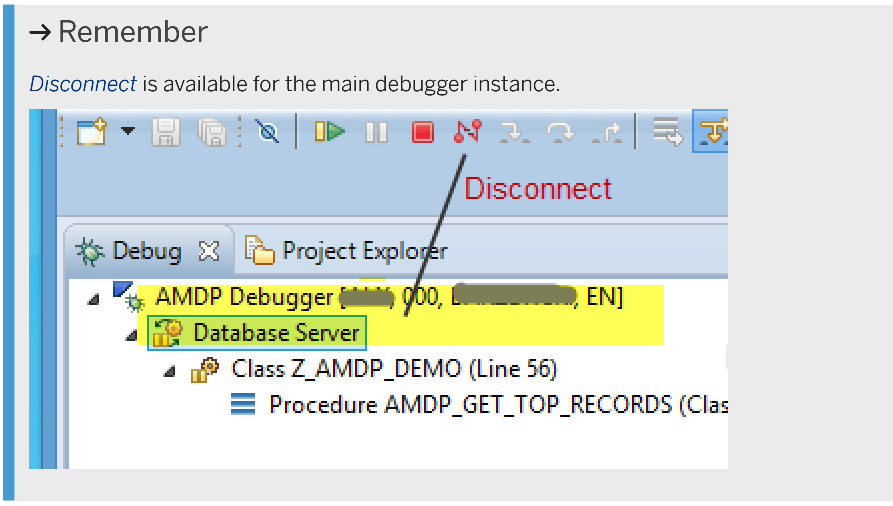
To proceed after reaching an AMDP breakpoint, the following functions for navigating in the AMDP debugger are currently available:

Function	Effect
`F8` – Resume	Run to the next AMDP breakpoint or to the end of the program.
`F6` – Step Over	Execute the next SQLScript statement. If the next step is a procedure call, run the entire procedure.

<table>
<tr><th>Function</th><th>Effect</th></tr>
<tr><td>Disconnect</td><td>Cancel the AMDP Debugger while the debugged application resumes execution, ignoring any intervening breakpoints.

→ Remember

Disconnect is available for the main debugger instance.


</td></tr>
<tr><td>Terminate</td><td>Cancel the AMDP Debugger and the execution of the debugged application (debuggee).</td></tr>
</table>

→ Tip

Since the function *Step Into* (F5) is missing, you can add a breakpoint before entering a new procedure instead.

i Note

Both the call stack and *Step Over* (F6) require SAP HANA DB >=SPS9.

3.2.1.3.1 Parallel Execution of Procedures in the AMDP Debugger

This topic describes an issue concerning the parallel execution of procedures that can occur during debugging of standard AMDPs as well as during debugging of CDS table functions. In particular, you could take this issue into account while debugging **table functions** that are **used in unions and joins**.

Multithreading Issue When Debugging AMDPs

Depending on your chosen debugger settings and breakpoints, it is possible that only some of your procedures are executed in debug mode, while others are executed in optimized mode.

More on this: AMDP Debugger Settings [page 19]

Procedures that run in optimized mode can potentially trigger parallel execution of called procedures. These parallel executions can then have an impact on debugging.

The following example demonstrates this kind of situation for AMDPs:

- Procedure A calls procedure B multiple times.
- Procedure A runs in **optimized mode**, while procedure B runs in **debug mode**. The source code of procedure B contains a breakpoint.
- The multiple calls of the procedure B are executed in parallel threads at the same time.

The AMDP debugger then stops all parallel thread executions of procedure B, but only the first one is accessed by the debugger for the variable inspection and user commands.

After choosing *Continue* (`F8`) or *Step-Over* (`F6`), the first thread continues with the execution, and one of the other remaining threads is stopped for debugger access.

⚠ Caution

The main impacts on debugging are as follows:

- It is not easy to identify which thread is currently being debugged.
- As there are no thread IDs, they are not displayed in any particular sequence in the AMDP debugger.

→ Tip

For the example above, it can be helpful if you create an additional breakpoint in the procedure A. Then, the debug mode would be enabled also in the calling procedure.

Multithreading Issue When Debugging Table Functions

The multithreading issue can also occur if a table function is called in parallel by a SQL statement, for example, due to a `JOIN` or `UNION` clause. In that case, it is not possible to solve the problem by creating an another breakpoint and the described multithreading issue is unavoidable!

3.2.1.4 Inspecting Variables

You can display and check the value of variables at runtime while debugging an AMDP.

Context

In the AMDP debugger, you have the following possibilities to inspect variables:

- Inspecting Scalar Variables and Table Variables [page 25]
- Inspecting Long Variable Values [page 26]
- Inspecting Table Contents [page 27]

- Inspecting Contents of Global Temporary Tables [page 28]

3.2.1.4.1 Inspecting Scalar Variables and Table Variables

In ABAP Managed Database Procedures (AMDPs) there are scalar and table variables available, but no structures.

Context

Scalar variables can be inspected in the *Variables* view of the *Debug* perspective while table variables can be inspected in the *Data Preview* tool.

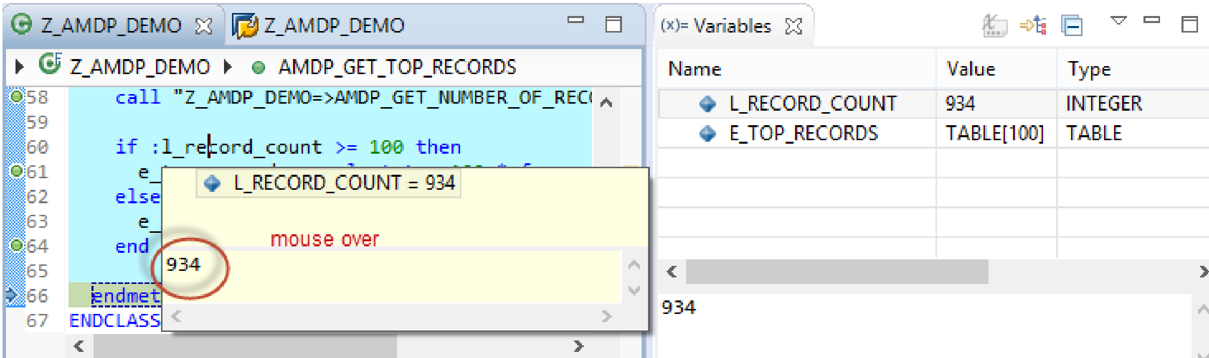
Additionally there is the option to mouse hover over variables in the source editor.

Procedure

To view the variable values within the SQLScript, proceed as follows:

Double-click the variable in the editor to open it in the *Variables* view of the *Debug* perspective.

> → Tip
>
> Another way to inspect the variable values is to hover with the mouse cursor over the variable name in the source editor in the *Debug* perspective.



Inspecting scalar variables (mouse over or Variables view) in the Debug perspective

> i Note
>
> The types of the variables are native DB types like `VARCHAR`. If you want to see input/output parameters after the type mapping to ABAP data types, you can use the ABAP Debugger.

Related Information

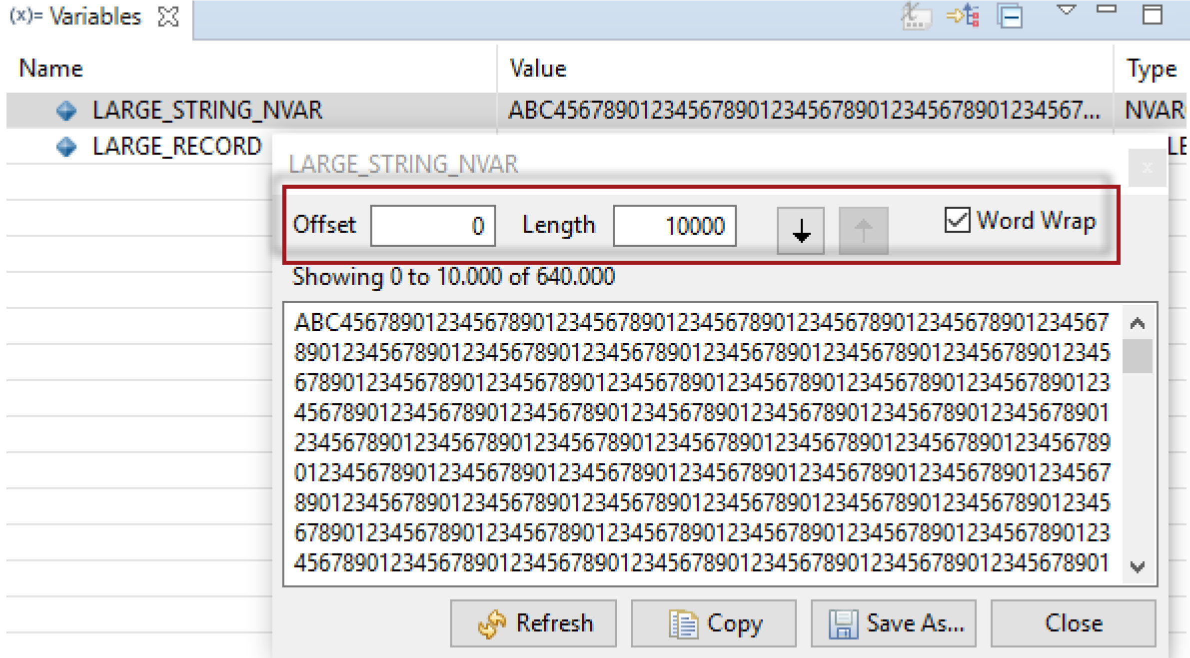
[Inspecting Long Variable Values [page 26]](#)
[Inspecting Table Contents [page 27]](#)
[Inspecting Contents of Global Temporary Tables [page 28]](#)

3.2.1.4.2 Inspecting Long Variable Values

As a rule, the presentation of long variable values is truncated in the AMDP debugger. This is for performance reasons, since variables are refreshed during stepping in the debugger, and there is natural limit on the available space on the UI.

If needed however, you have the option of inspecting the whole variable value:

To open the details screen with a long variable value, double-click a variable value (scalar or table cell) or choose the relevant function from the context menu.



Details screen with a long variable value

By default, the first 10,000 characters (offset = 0, length = 10,000) of the value presentation are shown in the detail screen. You can manually enter any value here for *Offset* and *Length*, or use the arrow buttons to page through the value.

The *Word Wrap* option adds line breaks in accordance with the size of the popup screen. Without word wrap, every line shows 1024 characters.

→ Remember

The word wrap option has a significant impact on the performance of the value rendering:

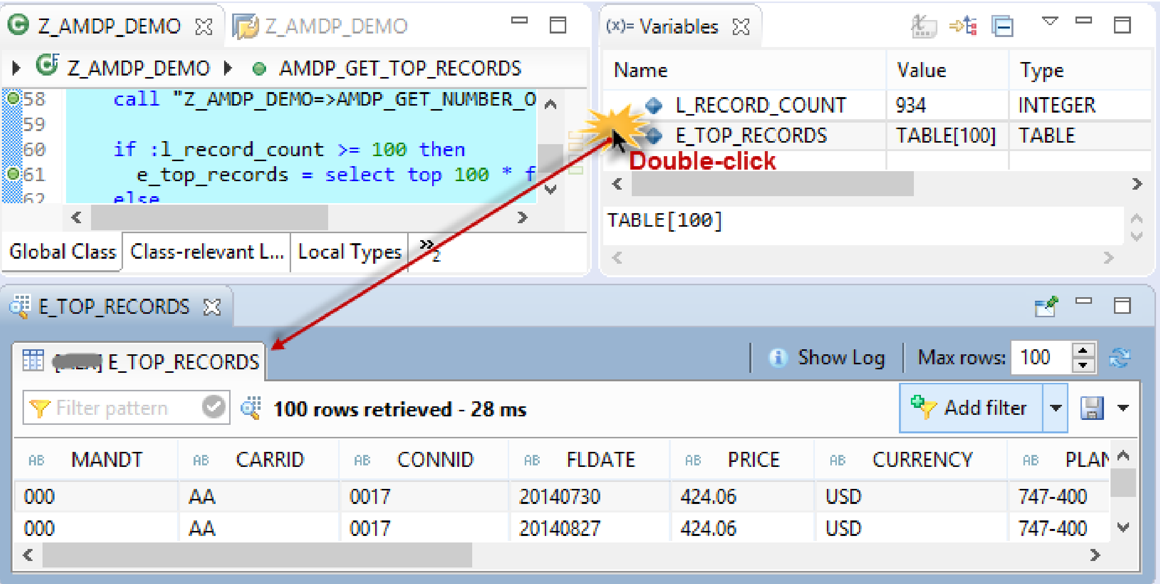
- Without activating the *Word Wrap* option, it should be easily possible to present 500,000 or more characters at once (~0.5-1s response time)
- With *Word Wrap* activated, presenting 10,000 characters at once is fast (up to 1s) but presenting 100,000 characters already takes up to 10 seconds.

3.2.1.4.3 Inspecting Table Contents

Procedure

To view the table contents...

1. Double-click the table variable in the source editor to open it in the *Variables* view In the *Debug* perspective.
2. Double-click the table entry in *Variables* view or choose *Show in Data Preview* from the context menu.



Inspecting table contents in Data Preview tool that is integrated in the Debug perspective

3.2.1.4.4 Inspecting Contents of Global Temporary Tables

You, as an ABAP developer, use global temporary tables (GTT) to read and write temporary data within a database LUW. This data can be accessed both through ABAP source code and an ABAP Managed Database Procedure (AMDP) as well as be displayed in the *Data Preview*.

Prerequisites

- The AMDP Debugger is activated.
- The AMDP Debugger is located, for example at a breakpoint within an ABAP Managed Database Procedure (AMDP).

Context

You want to display and check the content of a global temporary table (GTT), for example while debugging an AMDP.

Procedure

You have the following possibilities to trigger the data preview of GTTs:

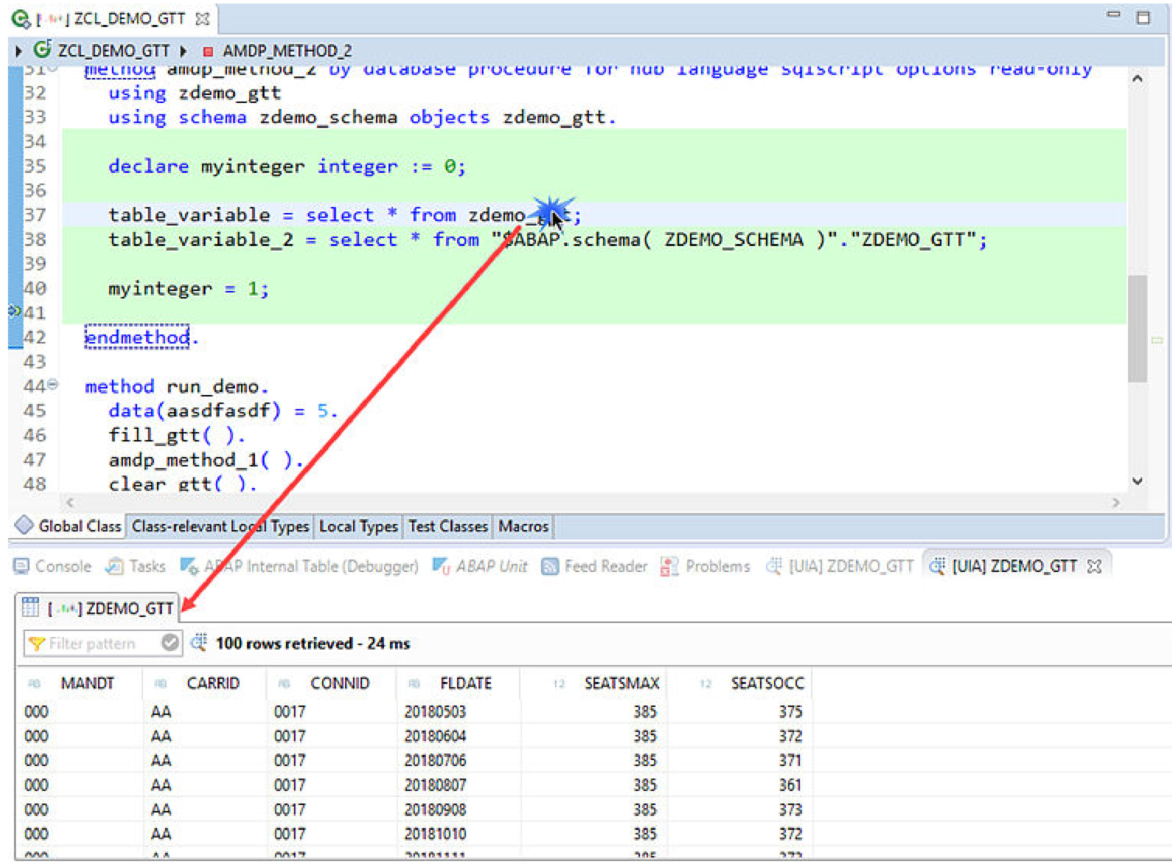
Triggering From an Explicit Occurrence within the Source Code

If the explicit name of the relevant GTT is used in the AMDP that is currently opened in the editor, proceed as follows:

Sample Code

```
... table_variable = select * from ZDEMO_GTT; ...
```

1. From the editor, double-click the explicit name of the GTT in the source code.



Triggering the preview from the source code

Triggering From a Fully Qualified Occurrence within the Source Code

If the fully qualified name of the relevant GTT and/or a database schema is used in the AMDP that is currently opened in the editor, proceed as follows:

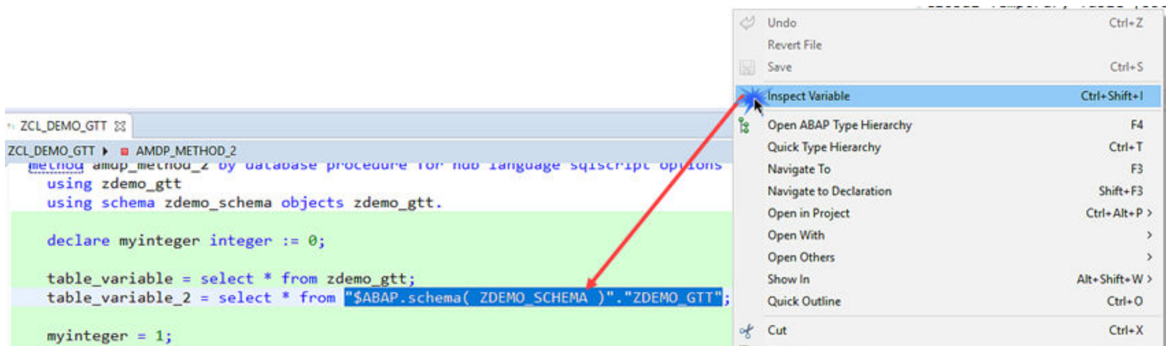
Sample Code

```
... table_variable_2 = select * from
"$ABAP.schema( ZDEMO_SCHEMA )"."ZDEMO_GTT"; ...
```

→ Recommendation

SAP recommends this approach if you use macros in an AMDP.

1. From the editor, select the full qualified name of the database schema and the name of the GTT inclusive the opening and closing hyphens.
2. From the context menu, choose *Inspect Variable*.

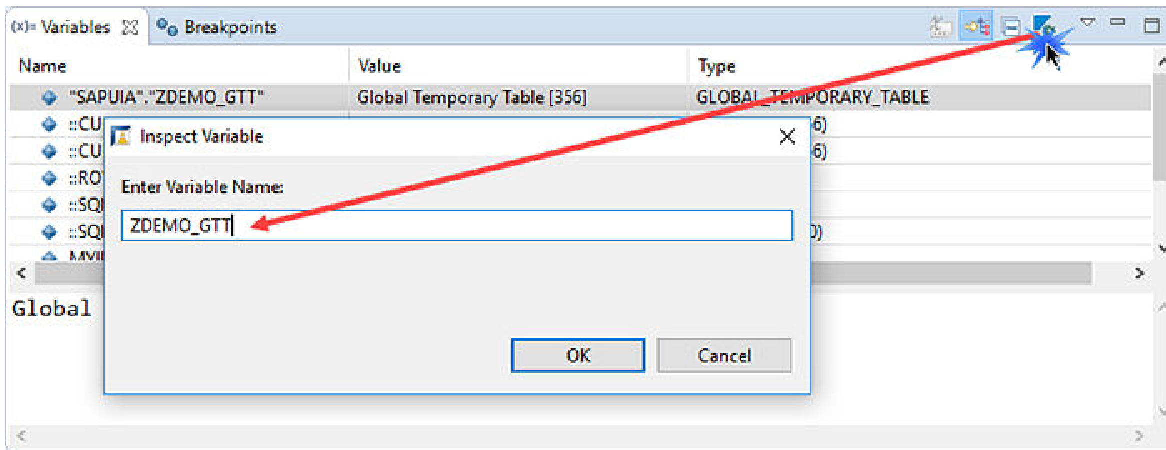


Triggering the context menu for inspecting variables from the full qualified names of the database schema and GTT

General Triggering For GTTs

If you only know the variable name which is not used in the currently opened editor, proceed as follows:

1. From the *Variables* view, choose the *Inspect Variable...* icon from the toolbar.
   The *Inspect Variable* dialog is opened.



Triggering the preview from the Variables view

2. Enter the name of the GTT or the full qualified name which might include a macro.
3. To confirm, choose *OK*.

Results

The *Data Preview* is opened. From here, you can filter, sort, and so on.

Related Information

Global Temporary Tables

3.2.1.5 Changing Scalar Variables

Scalar variables are used to store an unstructured value in an ABAP-Managed Database Procedure (AMDP).

Prerequisites

You have to explicitly activate the AMDP Debugger.

Context

You want to change the value of a variable while debugging an AMDP method, for example, to test the subsequent SQLScript code.

> **Note**
>
> In accordance with the underlying database, you can change variables of the following types:
>
> - `NVARCHAR`
> - `STRING`
> - `INTEGER`
> - `DOUBLE`
>
> `BLOB` and `CLOB` are currently not supported.

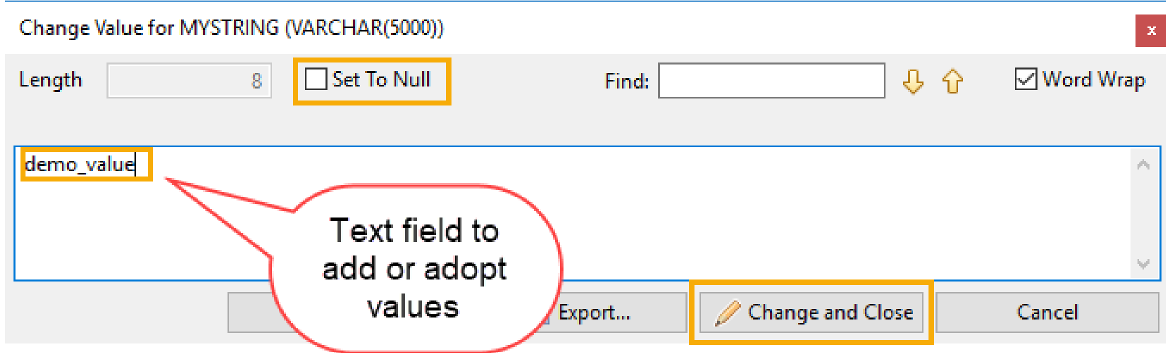
Procedure

1. In the source code of an AMDP method, set a new breakpoint or hover over an existing one and activate the AMDP Debugger.
2. Trigger the AMDP Debugger.

   The *Debugger* perspective is opened.

   All variables and their values are displayed in the *Variables* view.
3. In the *Variables* view, select a variable and choose *Change Variable...* from the context menu.

   The *Change Value* dialog is opened.



Dialog to change the value of scalar variables

4. In the text field, enter the new value or choose the *Set to Null* checkbox.
5. To confirm your changes, choose *Change and Close*.

Results

The dialog is closed and the value of the variable is adopted in the database.

Related Information

Activating AMDP Debugger [page 17]
Setting AMDP Breakpoints [page 20]

3.2.1.6 Deactivating AMDP Debugger

Context

One technical restriction of AMDP Debugger is that it is quite resource-intensive while it is active. Therefore, it should be terminated when not needed.

Procedure

- There are several ways to do this:
- By **explicit deactivation** of the AMDP Debugger.
    a. Choose *Terminate* AMDP Debugger from the context menu in the ruler of the source editor or in the Debug view.
    As a result of this action, both the debugged application (debuggee) and the AMDP Debugger will be canceled.

b. Choose *Disconnect*.
   This action is only available during a running debug session; it cancels the AMDP Debugger, while the debugged application resumes execution.

- Through timeout, after 10 minutes without user activity.
- **Implicitly** by closing the ABAP Development Tools.

3.2.2 Working with the AMDP Profiler

You use AMDP profiling to analyze and measure the runtime behavior of ABAP Managed Database Procedures (AMDP).

AMDP Profiler has been fully integrated into ABAP Profiler of ABAP Development Tools (ADT).

You have the following possibilities when working with AMDP Profiler:

- Running AMDP Profiler [page 33] to trigger the analysis.
- Opening the AMDP Profiling Result [page 35] to open the analysis from the *ABAP Trace* view.

Related Information

AMDP Profiling [page 6]

3.2.2.1 Running AMDP Profiler

Context

You need to run the ABAP Profiler to collect the data from the execution of an AMDP.

Procedure

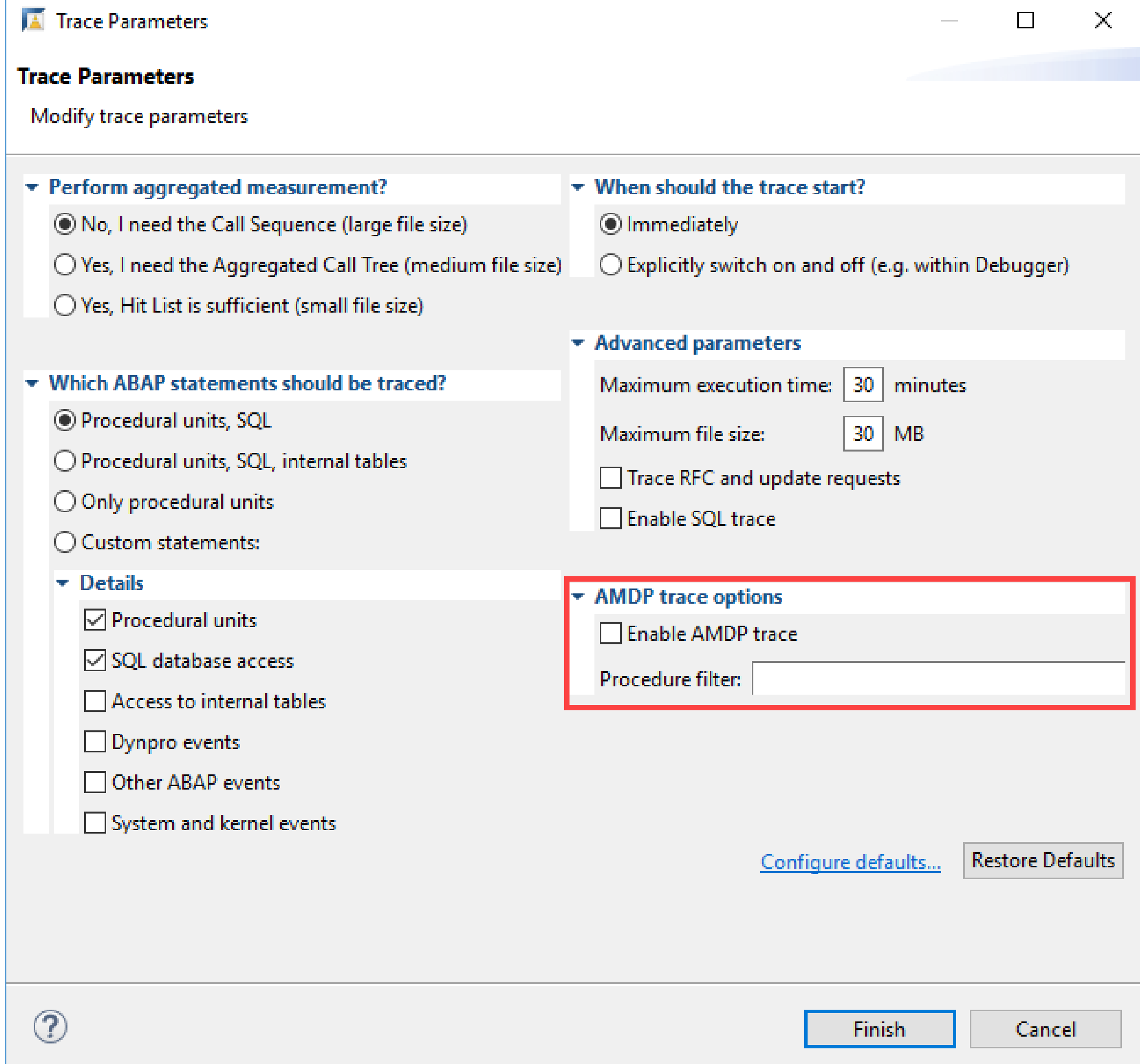
1. To profile the AMDP, choose the relevant execution option from the ABAP source code editor.

   You have the following execution options, to trigger AMDP profiling:
   - Choose `Run` + `Profile ABAP Development Object...` from the menu bar and select the relevant execution option.
   - Choose `Profile as` + `ABAP Application (Console)` from the context menu of the editor.
   - Choose `Profile as` + `ABAP Unit Test` from the context menu of the editor.
   - from the *ABAP Trace RequestsCreate an ABAP trace request* view.

   The *Trace Parameters* window is then opened.

> **i Note**
>
> You can also run an ABAP unit test. In this case, set the *AMDP trace options* in the *Profile Configurations...* window.
>
> If there is no entry for the ABAP unit test in the profile configurations, run the ABAP unit test profiling once to create it.

2. Select the *Enable AMDP trace* checkbox in the *AMDP trace options* section.



Trace Parameter window for defining the AMDP trace options

3. [Optional:] To run AMDP profiling only for specific procedures, set a procedure filter by entering the name of the relevant procedure(s) in the *Procedure filter* input field.

   Use a colon as separator to enter multiple procedures.
4. Choose *Finish*.

Results

The execution and profiling of the AMDP is started. When the execution is finished, the ABAP trace is created and is then ready to be analyzed in the *ABAP Traces* view.

3.2.2.2 Opening the AMDP Profiling Result

Context

You want to get detailed information, for example, about the duration of the execution of a specific procedure.

Procedure

1. To display the analysis result of the AMDP profiling, choose the relevant ABAP project and double-click the relevant trace in the *ABAP Traces* view

   The *Overview* is opened.
2. Choose the *ABAP Managed Database Procedures* tab from below.

   The *ABAP Managed Database Procedures* overview is opened and displays the result of the AMDP profiling analysis in a table.
3. To get more details about the statement which has been executed at ABAP runtime, view the table and expand the relevant procedure nodes.
4. [Optional:] To navigate to the relevant position within your ABAP source code, double-click the corresponding statement.

   The development object is opened and the cursor is positioned at the relevant position.

4 What's New in ABAP for SAP HANA Development

SAP ABAP for SAP HANA is an integral part of the ABAP Development Tools (ADT). The relevant ABAP back ends support certain ADT functionalities.

ABAP Environment

The following list gives you an overview of the released ADT client versions:

- Version 3.6 [page 36]
- Version 3.4 [page 37]

4.1 Version 3.6

Here is an overview of the most significant changes in the context of ABAP for SAP HANA building tools that relate to the following:

- Client: **ABAP Development Tools (ADT) 3.6**
- Back end version: SAP Cloud Platform ABAP Environment **1911**.

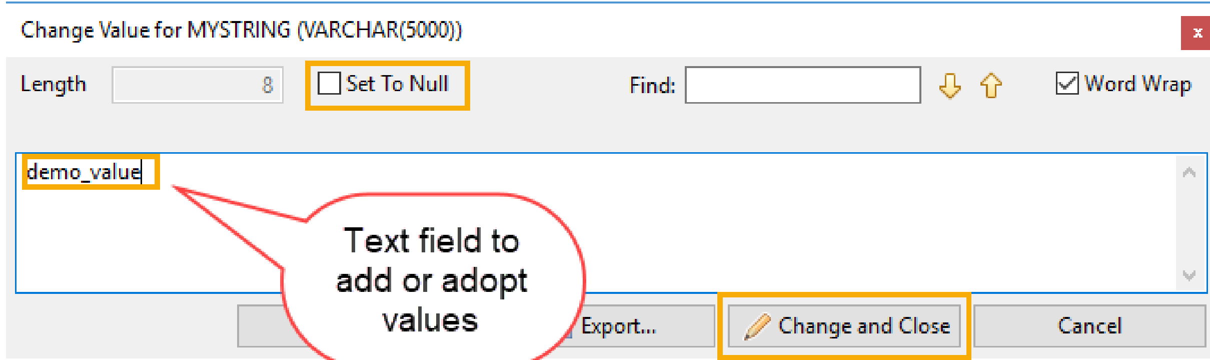
The following features that are highlighted with a '*' are client-specific and are therefore available for all supported ABAP systems.

Using Troubleshooting Tools

Changing Scalar Variables While Debugging an AMDP

You can now change the value of a variable while debugging an AMDP method, for example, to test the subsequent SQLScript code.

You can trigger this function from the *Change Value* dialog in the *Variables* view while debugging an AMDP. To open this dialog, select the relevant variable and choose *Change Variable...* from the context menu.



Dialog to change the value of scalar variables

For more information, see Changing Scalar Variables [page 31]

4.2 Version 3.4

Here is an overview of the most significant changes in the context of ABAP for SAP HANA building tools that relate to the following:

- Client: **ABAP Development Tools (ADT) 3.4**
- Back end version: SAP Cloud Platform ABAP Environment **1908**.

The following features that are highlighted with a '*' are client-specific and are therefore available for all supported ABAP systems.

Using Troubleshooting Tools

Debugging GraphScript

GraphScript is a high-level, domain-specific programming language for development on SAP HANA.

You can now use AMDP Debugger for ABAP source code that contains GraphScript code.

This enables you, for example to set breakpoints for GraphScript statements and step iteratively through your ABAP application.

For more information, see

- Working with the AMDP Debugger [page 16]

4.3 ⌂ Version 3.0

Here is an overview of the most significant changes in the context of ABAP for SAP HANA building tools that relate to the following:

- Client: **ABAP Development Tools (ADT) 3.0**
- Back end version: ☁ SAP Cloud Platform ABAP Environment **1902**.

The following features that are highlighted with a '*' are client-specific and are therefore available for all supported ABAP systems.

☁ Using Troubleshooting Tools

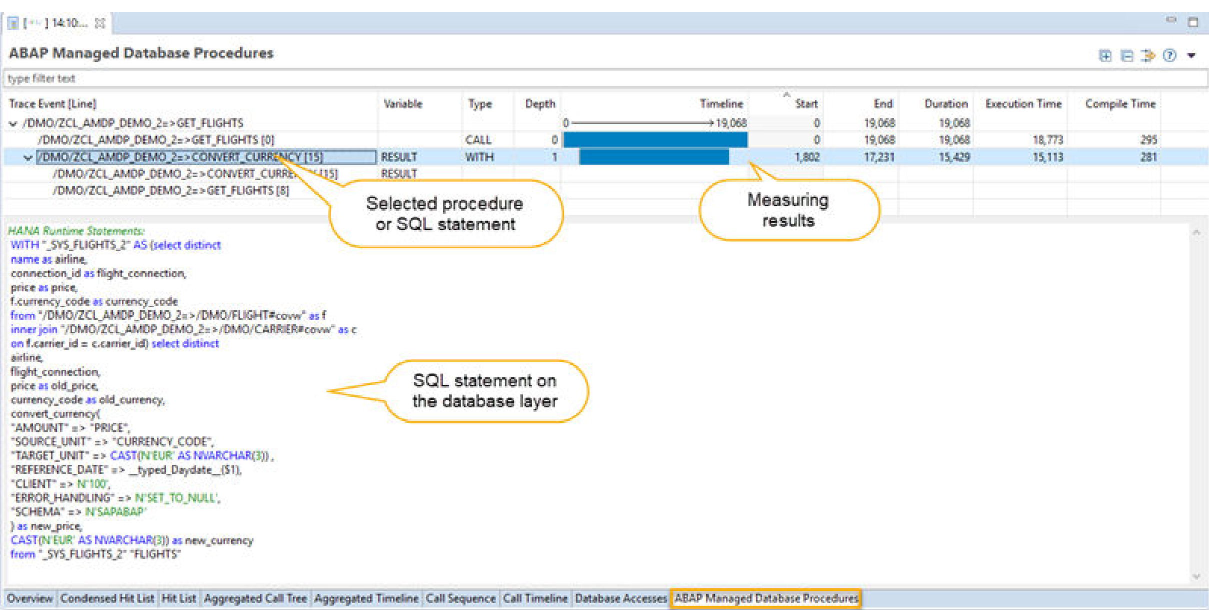
AMDP Profiling

You can now use AMDP profiling to analyze the runtime behavior of an ABAP Managed Database Procedure (AMDP). AMDP Profiler has been fully integrated into the ABAP Profiler.

You have the following options, to trigger AMDP profiling:

- Choose `Profile as` + `ABAP Application (Console)` from the context menu of the editor.
- Choose `Profile as` + `ABAP Unit Test` from the context menu of the editor.
- from the *ABAP Trace Requests* view.

The analysis results are generated for an ABAP trace and displayed in the *ABAP Managed Database Procedures* tab.



Example for displaying and analyzing the ABAP trace of an AMDP

For more information, see

- AMDP Profiling [page 6]
- Working with the AMDP Profiler [page 33]

Debugging AMDPs

You can now debug AMDP code and CDS table functions within the ABAP Development Tools.

For more information, see Working with the AMDP Debugger [page 16]

Displaying the Content of Global Temporary Tables While AMDP Debugging

You can now display the content of global temporary tables (GTTs) at runtime in the *Data Preview* while debugging AMDPs.

For more information, see Inspecting Contents of Global Temporary Tables [page 28]

Profiling ABAP Applications (Console)

You can now profile ABAP applications which are executed in the console. To do this, choose `Profile As` + `ABAP Application (Console)` from the context menu in the ABAP source code editor.

For more information, see

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

www.sap.com/contactsap

© 2020 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.