

CBA SAP MCP Servers

1. MCP server for ABAP documentation including keyword documentation, official SAP Help, cheat sheets and community posts AND CBA guardrails.
2. MCP server for connecting to CBA development system DB3.

⚠ Currently these can only be configured in VSC. MCP servers are blocked by Security in GitHub Copilot and Copilot is the only AI tool working in Eclipse. An official ADT plugin for VSC is coming Q2 2026

Remember if you want to use VSC for ABAP dev you will need the ABAP FileSystem plugin.

1. ABAP Documentation

MCP server up and running on DHP. Connection URL - <https://d-s-jarvis-sapdocs-alb.dotdtcbp01dev01.aws.test.au.internal.cba/mcp>

Example settings in Roo Code:

```
1 "sap-docs": {  
2     "type": "streamable-http",  
3     "url": "https://d-s-jarvis-sapdocs-  
alb.dotdtcbp01dev01.aws.test.au.internal.cba/mcp",  
4     "alwaysAllow": [  
5         "search",  
6         "sap_help_search",  
7         "sap_community_search",  
8         "sap_help_get",  
9         "fetch"  
10    ],  
11    "disabled": false  
12},
```

- ① If you have trouble connecting you may need to download the DHP certificate. Use a similar process for installing the CBA Root Certificate: Windows - <https://engineering-handbook.pages.commbank.io/knowledge-base/windows-soe-guides/CBA-root-certificate/> Mac - <https://engineering-handbook.pages.commbank.io/knowledge-base/mac-guides/CBA-root-certificate>

Example query.

Prompt:

You said

What are the best ways in modern ABAP to do inline data declarations?

Response:

✓ Roo code using mcp-abap-adt and claude sonnet 4.5

Best Ways to Use Inline Data Declarations in Modern ABAP

1. DATA(. . .) - Basic Inline Declaration

The most common and versatile inline declaration method:

```
1 " Instead of declaring separately
2 DATA lv_result TYPE i.
3 lv_result = 5 + 3.
4
5 " Use inline declaration
6 DATA(lv_result) = 5 + 3. " Type automatically inferred as i
7
8 " With method calls
9 DATA(lv_customer) = get_customer( iv_id = '12345' ).
10
11 " With table operations
12 DATA(lv_lines) = lines( lt_table ).
```

2. FIELD-SYMBOL(<fs>) - Inline Field Symbol Declaration

Declare field symbols directly in loop statements:

```
1 " Traditional approach
2 FIELD-SYMBOLS <fs_line> TYPE ty_structure.
3 LOOP AT lt_table ASSIGNING <fs_line>.
4   <fs_line>-field = 'X'.
5 ENDLOOP.
6
7 " Modern inline approach
8 LOOP AT lt_table ASSIGNING FIELD-SYMBOL(<fs_line>).
9   <fs_line>-field = 'X'.
10 ENDLOOP.
11
12 " With READ TABLE
13 READ TABLE lt_table ASSIGNING FIELD-SYMBOL(<fs_entry>)
```

```
14  WITH KEY id = '123'.
15  IF sy-subrc = 0.
16    <fs_entry>-status = 'Active'.
17  ENDIF.
```

3. FINAL(. . .) - Immutable Inline Constants

Create constants inline (available from ABAP 7.40 SP08):

```
1 " Declare unchangeable value
2 FINAL(lc_max_items) = 100.
3
4 " In loops
5 LOOP AT lt_data INTO FINAL(ls_data).
6   " ls_data is read-only within loop
7 ENDLOOP.
8
9 " Method parameters
10 calculate_total(
11   iv_rate = FINAL(lc_rate) = '1.19'
12   iv_amount = lv_amount
13 ).
14
15 " Cannot modify FINAL variables
16 " lc_max_items = 200. " Syntax error!
```

4. REF and NEW - Inline Reference Declarations

Create data references inline:

```
1 " Traditional approach
2 DATA lr_data TYPE REF TO ty_structure.
3 CREATE DATA lr_data.
4
5 " Modern inline approach
6 DATA(lr_data) = NEW ty_structure( ).
7
8 " With initial values
9 DATA(lr_customer) = NEW ty_customer(
10   id = '12345'
11   name = 'John Doe'
12 ).
13
14 " Anonymous data objects
15 DATA(lr_dynamic) = NEW data( ).
16
17 " Table references
18 DATA(lr_table) = NEW ty_table_type( ).
```

5. Loop Variants with Inline Declarations

Multiple ways to declare within loops:

```
1 " INTO with inline declaration
2 LOOP AT lt_sales INTO DATA(ls_sale).
3   ls_sale-amount = ls_sale-amount * 2. " Copy, not reference
```

```

4   ENDOLOOP.
5
6   " ASSIGNING with inline field symbol (better performance)
7   LOOP AT lt_sales ASSIGNING FIELD-SYMBOL(<fs_sale>).
8     <fs_sale>-amount = <fs_sale>-amount * 2. " Direct modification
9   ENDOLOOP.
10
11  " REFERENCE INTO with inline reference
12  LOOP AT lt_sales REFERENCE INTO DATA(lr_sale).
13    lr_sale->amount = lr_sale->amount * 2. " Reference modification
14  ENDOLOOP.
15
16  " FINAL for read-only iteration
17  LOOP AT lt_sales INTO FINAL(ls_sale).
18    " ls_sale cannot be modified
19    WRITE: / ls_sale-amount.
20  ENDOLOOP.

```

6. VALUE Constructor with Inline Declaration

Build structures and tables inline:

```

1 " Structure construction
2 DATA(ls_order) = VALUE ty_order(
3   order_id = '1000'
4   customer = 'CUST001'
5   amount = '999.99'
6   ).
7
8 " Table construction
9 DATA(lt_items) = VALUE ty_item_table(
10   ( item_id = '10' quantity = 5 )
11   ( item_id = '20' quantity = 3 )
12   ( item_id = '30' quantity = 7 )
13   .
14
15 " Based on existing table
16 DATA(lt_filtered) = VALUE ty_table(
17   FOR <line> IN lt_source WHERE ( status = 'A' )
18   ( <line> )
19   .

```

7. Table Operations with Inline Declarations

```

1 " READ TABLE
2 READ TABLE lt_customers INTO DATA(ls_customer)
3   WITH KEY id = '12345'.
4
5 " READ TABLE with field symbol (better performance)
6 READ TABLE lt_customers ASSIGNING FIELD-SYMBOL(<fs_customer>)
7   WITH KEY id = '12345'.
8
9 " READ TABLE with reference
10 READ TABLE lt_customers REFERENCE INTO DATA(lr_customer)
11   WITH KEY id = '12345'.
12
13 " With table expression

```



```
4                                     ->get_result( ).  
5  
6 " Functional style  
7 DATA(lt_filtered) = NEW lcl_filter( lt_data )->by_status( 'A' )  
8                                         ->by_date( sy-datum )  
9                                         ->get_results( ).
```

11. Best Practices

✓ DO's

1. Use inline declarations for local variables

```
1 " Good: Type is clear from context.  
2 DATA(lv_count) = lines( lt_table ).  
3 DATA(ls_config) = get_configuration( ).
```

2. Prefer FIELD-SYMBOL for table loops

```
1 " Better performance - no copying  
2 LOOP AT lt_large_table ASSIGNING FIELD-SYMBOL(<fs_item>).  
3   <fs_item>-processed = abap_true.  
4 ENDLOOP.
```

3. Use FINAL for immutability

```
1 " Prevent accidental modifications  
2 LOOP AT lt_data INTO FINAL(ls_data).  
3   display_data( ls_data ).  
4 ENDLOOP.
```

4. Combine with modern operators

```
1 DATA(lv_result) = COND #(  
2   WHEN lv_value > 100 THEN 'High'  
3   WHEN lv_value > 50  THEN 'Medium'  
4   ELSE 'Low'  
5 ).
```

✗ DON'Ts

1. Don't use for globally visible variables

```
1 " Bad: Type not clear in class definition  
2 CLASS-DATA gv_counter TYPE i. " Better than DATA(gv_counter)
```

2. Avoid when type is not obvious

```
1 " Bad: What type is this?  
2 DATA(lv_result) = calculate_something( ).  
3  
4 " Better: Make type explicit if unclear  
5 DATA lv_result TYPE ty_calculation_result.  
6 lv_result = calculate_something( ).
```

3. Don't overuse in complex expressions

```
1 " Hard to read
2 DATA(iv_x) = REDUCE i( INIT s = 0 FOR wa IN lt_tab NEXT s = s +
3   COND #( WHEN wa-type = 'A' THEN wa-val * 2 ELSE wa-val ) ).
4
5 " Better: Break down
6 DATA(iv_sum) = REDUCE i(
7   INIT sum = 0
8   FOR line IN lt_table
9   NEXT sum = sum + get_value( line )
10 ).
```

12. Performance Considerations

Method	Performance	Use Case
FIELD-SYMBOL	★★★★★	Modifying large tables
REFERENCE INTO	★★★★★	Read-only access to large structures
INTO DATA(...)	★★★	Small datasets, readability
FINAL(...)	★★★★	Read-only iteration, safety

Summary

Modern ABAP inline declarations make code:

- More concise - Less boilerplate
- More readable - Declaration near usage
- Type-safe - Compiler infers types
- Maintainable - Reduced scope of variables

The key is choosing the **right method** for your specific use case while maintaining **code clarity and performance**.

 Our CBA guardrails have also been added allowing the server to make suggestions for improving your code from a CBA specific perspective.

Issue - SAP Virtual File System

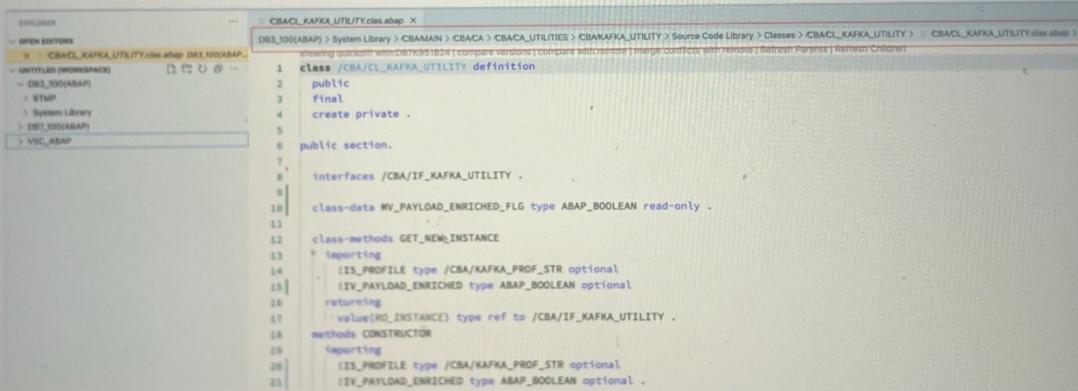
When developing in SAP all objects are located on the server. This creates issues with AI tools not having access to the required code. Another MCP server that can help here is the MCP

ABAP ADT server that can connect to our DB3 backend. More on this MCP below.

If I ask Roo to evaluate some code based on our CBA guardrails it will have the guardrails info thanks to the above MCP but it often won't "see" the code even if it is in the current editor window. Most development environments will create a local workspace on the engineer's laptop where the current code is stored. AI tools have no problem with the code there but fail when the code is actually installed in the SAP backend. In the past I've gotten around this by opening code in VSC from DB3 but then saving a local copy to my current workspace on my laptop.

Having the extra MCP server that can connect to DB3 means I don't have to save locally as can be seen in this example.

Code is open in VSC using ABAP FileSystem plugin:



The screenshot shows the SAP IDE Workbench interface. The left sidebar displays the 'OPEN EDITORS' list, which includes 'CBACL_KAFKA_UTL/ITY.cls.abap' (highlighted in red) and 'DB3_100/ABAP'. The main editor window shows the source code for the 'CBACL_KAFKA_UTL/ITY.cls.abap' file. The code defines a class 'CBACL_KAFKA_UTL/ITY' with various sections and methods. The code is as follows:

```
1 class /CBA/CL_KAFKA.Utility definition
2 public
3 final
4 create private .
5
6 public section.
7
8 interfaces /CBA/IF_KAFKA.Utility .
9
10 class-data MV_PAYLOAD_ENRICHED_FLG type ABAP_BOOLEAN read-only .
11
12 class-methods GET_NEW_INSTANCE
13   importing
14     !IS_PROFILE type /CBA/KAFKA_PROF_STR optional
15     !IV_PAYLOAD_ENRICHED type ABAP_BOOLEAN optional
16   returning
17     value(!RD_INSTANCE) type ref to /CBA/IF_KAFKA.Utility .
18
19 methods CONSTRUCTOR
20   importing
21     !IS_PROFILE type /CBA/KAFKA_PROF_STR optional
22     !IV_PAYLOAD_ENRICHED type ABAP_BOOLEAN optional .
```

Roo prompt:

ⓘ Can you comment on if the code in class /CBA/CL_KAFKA.Utility conforms according to CBA guardrails?
⌚ 16.84 / 128.0h

Roo initially fails to read the code that is in the editor window (@ context prompt does not work here).

 Roo said

I'll help you review the code in class `/CBA/CL_KAFKA.Utility` against CBA guardrails. Let me first read the class to understand its implementation.

 Roo wants to read this file outside of the workspace

System Library//CBA/MAIN//CBA/CA//CBA/CA_UTILITIES//CBA/KAFKA.Utility/Source Code Library/Classes//CBA/CL_KAFKA_U...

 Error

Error reading file System Library/CBA/MAIN/CBA/CA/CBA/CA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes//CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:
File not found: /System Library/CBA/MAIN/CBA/CA/CBA/CA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes//CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap

Roo then uses the MCP ABAP ADT server to get the code.

 Roo wants to use a tool on the mcp-abap-adt MCP server

 mcp-abap-adt

 Completed

 GetClass

 Always all

Retrieve ABAP class source code

```
{  
    "class_name": "/CBA/CL_KAFKA.Utility"  
}
```

 API Request

 Task Completed

I've reviewed `/CBA/CL_KAFKA.Utility` (System Library/CBA/MAIN/CBA/CA/CBA/CA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:1) against CBA guardrails. Here's my assessment:

Task Completed

I've reviewed [/CBA/CL_KAFKA.Utility] (System Library/CBA/MAIN/CBA/CA/CBA/CA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:1) against CBA guardrails. Here's my assessment:

Code is evaluated

Code evaluated according to CBA guardrails

I've reviewed [/CBA/CL_KAFKA.Utility] (System Library/CBA/MAIN/CBA/CA/CBA/CA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:1) against CBA guardrails.

Here's my assessment:

Conforming Aspects

1. **Proper Naming Conventions:** Uses CBA namespace prefix (/CBA/) correctly
2. **Exception Handling:** Uses custom exception class
[/CBA/CX_KAFKA.Utility_FAILURE] (System Library/CBA/MAIN/CBA/CA/CBA/CA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CX_KAFKA.Utility_FAILURE.clas.abap:1)

/CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:93) with T100 message structure

3. **Interface Implementation:** Properly implements [/CBA/IF_KAFKA.Utility] (System Library/CBA/MAIN/CBACA/CBA/CA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:6)
4. **Factory Pattern:** Uses [/CBA/CL_KAFKA.Utility_FACTORY](System Library/ /CBAMAIN/CBACA/CBACA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:154) for dependency injection
5. **Singleton Pattern:** Implements controlled instantiation with
[GET_NEW_INSTANCE()](System Library/CBA/MAIN/CBA/CA/CBA/CA_UTILITIES/ /CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CL_KAFKA.Utility/ /CBA/CL_KAFKA.Utility.clas.abap:12) and CREATE PRIVATE
6. **Input Validation:** Proper validation in [CHECK_IF_INPUTS_ARE_FILLED()] (System Library/CBA/MAIN/CBA/CA/CBACA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:110)

Potential Issues

1. **Inline Declarations:** Uses inline DATA declarations (e.g., line 101:
DATA(lv_data_serialized)), which may not align with strict CBA coding standards if they prefer explicit declarations
2. **Magic Strings:** Kill switch parameters use string literals (lines 145-146) - should consider constants
3. **Commented Code:** Lines 156 and 163 contain commented-out code that should be removed during cleanup as noted in comments
4. **Limited Error Context:** [CHECK_PROFILE_CORRECTNESS()](System Library/ /CBAMAIN/CBA/CA/CBA/CA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:130) only validates profile_id is not initial - could validate other profile fields
5. **CLEAR Statement Placement:** Line 108 clears mv_payload_enriched_flg at end of method - consider if this is the intended behavior for a class variable

Recommendations

1. Remove commented code (lines 156, 163) before production deployment

2. Consider extracting kill switch parameters to constants
3. Add comprehensive profile validation in [**CHECK_PROFILE_CORRECTNESS()**]
(System Library/CBA/MAIN/CBA/CA/CBA/CA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:130)
4. Document the purpose of clearing `MV_PAYLOAD_ENRICHED_FLG` in
[**/CBA/IF_KAFKA.Utility~SEND_DATA_TO_KAFKA()**](System Library/CBA/MAIN/CBA/CA/CBA/CA_UTILITIES/CBA/KAFKA.Utility/Source Code Library/Classes/CBA/CL_KAFKA.Utility/CBA/CL_KAFKA.Utility.clas.abap:71)

Overall Assessment: The code largely conforms to CBA guardrails with proper use of namespacing, exception handling, and design patterns. Main concerns are housekeeping items (commented code) and minor enhancements for maintainability.

More details on this MCP server can be found here - [marijanfoo/mcp-sap-docs](#)

CBA Dev System DB3

*** Coming Soon ***

Harish is working on moving this MCP server to DHP. Currently I have it running on my laptop.

Example query, asking about the structure of a CBA table:

Prompt:

You said

what is the structure of database table /cba/bp_ar_info

API Request

Roo said

I'll retrieve the structure of the database table `/CBA/BP_AR_INFO` using the MCP ABAP ADT server.

Roo wants to use a tool on the mcp-abap-adt MCP server

mcp-abap-adt

Completed

GetTable

Retrieve ABAP table structure

Always allow

```
{
  "table_name": "/CBA/BP_AR_INFO"
}
```

Response:

✓ Roo code using mcp-abap-adt and claude sonnet 4.5

Structure of Database Table /CBA/BP_AR_INFO

Label: Privacy EU: BP Info from Omnia

Table Properties:

- Category:** Transparent table (#TRANSPARENT)
- Delivery Class:** Application table (#A)
- Data Maintenance:** Restricted (#RESTRICTED)
- Enhancement:** Not extensible (#NOT_EXTENSIBLE)

Key Fields

Field Name	Data Element	Description
client	mandt	Client (NOT NULL)
run_no	/cba/ca_bulk_dte_x un_number	Bulk DTE Run Number (NOT NULL)
ext_appl	/cba/ca_bulk_dte_o bj_appl	External Application (NOT NULL)
id	bapc_dte_log_objec t_id	DTE Log Object ID (NOT NULL)

Non-Key Fields

Field Name	Data Element	Description
partner	bu_partner	Business Partner Number
bpext	bu_bpext	Business Partner External ID
type	bu_type	Business Partner Type
no_accounts_flag	xfeld	No Accounts Flag (X = Yes)

bank_verified	/cba/bp_dte_bnk_verified	Bank Verification Status
crm_verified	/cba/bp_dte_crm_verified	CRM Verification Status
erp_verified	/cba/bp_dte_erp_verified	ERP Verification Status
bank_deleted	/cba/bp_dte_bnk_deleted	Bank Deletion Status
crm_deleted	/cba/bp_dte_crm_deleted	CRM Deletion Status
erp_deleted	/cba/bp_dte_erp_deleted	ERP Deletion Status
overall_status	/cba/bp_dte_ar_status	Overall Archive Status
error_log_system	/cba/bp_dte_ar_sys tem	Error Log System
error_log	bapi_msg	Error Log Message
received_on	date	Date Received (NOT NULL)
received_tstmp	bca_dte_cn_link_valid_to_real	Received Timestamp

Purpose

This table stores **Business Partner Archive Information from Omnia for Privacy EU** compliance, tracking verification and deletion status across Bank, CRM, and ERP systems as part of bulk data archiving operations.