PUBLIC

Extend SAP S/4HANA in the cloud and on premise with ABAP based extensions

Guidelines for extension project managers, key users, and ABAP developers

Version: 2.3
Date: August 2025
Owner: SAP BTP ABAP Product Management

ACTIVITY LOG

Version 2.3, August 2025	
Transformation of 3-tier extensibility to clean core extensibility based on clean core level concept	
CHAPTER	NOTE
5	Changed technical extensibility recommendations in line with clean core extensibility guidance from 3-tier model to ABAP Cloud and classic ABAP development
5.1	Changed set up from 3-tier model to governance for the classic ABAP development
5.1.1	Added chapter about SAP classic APIs including matching to clean core levels
5.1.2	Changed ATC setup with clean core checks
5.1.3	Added chapter about clean core governance using ATC exemptions
5.1.4	Added chapter about changelog for SAP objects for assessing upgrade stability if using SAP internal objects
5.1.5	Updated set up summary considering ABAP Cloud and classic ABAP development models
5.1.7	Changed custom code use cases relation from 3-tier model to clean core levels
6.1	Changed transformation options for legacy code from 3-tier model to ABAP Cloud and classic ABAP developments
Version 2.2, April 2024	
Repaired broken links in chapter 4.1	
Version 2.1, April 2024	
Minor changes in text for clarification and better understanding.	
Version 2.0, April 2023	
CHAPTER	NOTE
Document	Activity log added.
4.2	Added chapter about analytics in ABAP Cloud.
4.6	Updated recommendations for BC apps for SAP S/4HANA Cloud Private Edition, and SAP S/4HANA on-premise.

5.1	Updated recommendations for the setup of the 3-tier model.
5.2	Update on recommendations for wrappers.

TABLE OF CONTENTS

1 INTRODUCTION

Extensibility is a key capability of every Enterprise Resource Planning (ERP) solution. It enables customers to create a competitive advantage by customizing their business processes and allows partners to enrich ERP with tailor-made solutions. The importance of extensibility has been confirmed for SAP's on-premise ERP and will remain valid for the more standardized cloud ERP.

SAP S/4HANA is SAP's flagship product providing the intelligent ERP in the cloud and on-premise. This document helps SAP S/4HANA customers and partners to choose, implement and use the extensibility options correctly, and considers the various customer environments (public cloud, private cloud or on-premise).

The goal is to move from classic custom ABAP extensions to an SAP S/4HANA extensibility model that allows to consume SAP innovations smoothly, leading to future-proof extensions that adhere to the clean core guidelines provided by the Clean core extensibility whitepaper.

This enables customers and partners on the one hand side to reach the upgrade stability of their existing landscape while also paving the way for their next cloud transformation steps.

1.1 Motivation for a new extensibility model

During the last decades SAP's on-premise customers and partners have mainly used classic ABAP extensibility to extend their ERP solution. Classic extensibility allows ABAP developers to use and even to modify all SAP objects. This is very powerful and flexible. But the missing clear interface between SAP code and the extensions adds a lot of customer specific test and adaptation effort during SAP upgrades.

In the public cloud there are no customer-specific SAP upgrade projects. Instead, automated software updates run for all customers in parallel. Therefore, classic extensibility, based on classical custom ABAP code, can no longer be used.

SAP S/4HANA Cloud Public Edition provides a new upgrade-stable cloud extensibility model that clearly separates SAP code and extensions via mandatory publicly released SAP APIs and SAP extension points. It supports the following standard extensibility scenarios:

- **On-stack** extensions that depend on proximity to or tight coupling with SAP S/4HANA Cloud data, transactions or apps and therefore run on the SAP S/4HANA Cloud stack.
- **Side-by-side** extensions running on the separated SAP Business Technology Platform (SAP BTP) for all other loosely-coupled extension scenarios integrating with SAP S/4HANA.

The different personas working on extensibility use cases are supported via:

- **Low-code/no-code tools** for key users/business experts.
- A **development environment** (IDE) for professional developers.

SAP S/4HANA Cloud covers these dimensions with the following extensibility options:

SAP S/4HANA CLOUD EXTENSIBILITY OPTIONS		
PERSONAS	Tightly coupled extension on SAP S/4HANA (on-stack extensions)	Loosely-coupled extension on the side-by-side Platform SAP BTP (side-by-side extensions)
BUSINESS EXPERT, KEY USER, CITIZEN DEVELOPER	SAP S/4HANA Cloud key user extensibility	Low-code applications, workflows and automations (SAP Build Apps, SAP Build Process Automation and others)

SAP S/4HANA CLOUD EXTENSIBILITY OPTIONS		
PERSONAS	Tightly coupled extension on SAP S/4HANA (on-stack extensions)	Loosely-coupled extension on the side-by-side Platform SAP BTP (side-by-side extensions)
DEVELOPER	Developer extensibility using the SAP S/4HANA Cloud ABAP environment (a.k.a. Embedded Steampunk[1])	Full-stack applications (SAP Build Code, SAP BTP ABAP environment (a.k.a. Steampunk[1]))

Table 1.1 - SAP S/4HANA Cloud extensibility options

All these extensibility options use only SAP APIs and SAP extension points that have been released and are stable.

This new SAP S/4HANA Cloud extensibility model, first introduced in SAP S/4HANA Cloud Public Edition, is now available in all SAP S/4HANA editions, to achieve smoother upgrades everywhere and to pave the way to the cloud.

Table 1.2 below lists the different SAP S/4HANA editions and why a customer should adopt the new cloud extensibility model in the respective environment. The right column in the table lists which topics are covered in this guide.

	WHY CLOUD EXTENSIBILITY?	WHAT WILL BE EXPLAINED
SAP S/4HANA Cloud Public Edition	**Mandatory** Classical ABAP extensibility is not supported	• The SAP S/4HANA cloud extensibility model • When to use which cloud extensibility option • The ABAP Cloud development model
SAP S/4HANA Cloud Private Edition and on-premise **Greenfield installation**	**Start with the superior cloud extensibility model** Smoother SAP software updates Future-safe extensions for the next cloud transformation steps	• Technical setup to use the new cloud extensibility model • When to use the cloud extensibility model and when classic extensibility still must be used • Guidance on how to handle classic extensibility in parallel with the new cloud extensions
SAP S/4HANA Cloud Private Edition and on-premise **Installation with converted classic extensions**	**Get the benefits described above for new and existing extensions** Developers learn the new extensibility model and can transform classic extensions step-by-step to cloud-ready extensions	• Best practices on how to manage, eliminate or transform existing classic extensions based on the clean core extensibility model

Table 1.2 - Overview about the positioning of the new cloud extensibility model in the different SAP S/4HANA editions

[1] Steampunk and Embedded Steampunk are the SAP internal project names to deliver the ABAP environment on SAP BTP and in SAP S/4HANA Cloud. These project names are often used in development-related blogs and articles.

The list below gives some recommendations on how to read this guide:

- Chapters 1-3 are recommended for all readers to get an overview of the ABAP-related aspects of the new cloud extensibility model and to understand when to use which extensibility option.
- Chapter 4 introduces the new ABAP Cloud development model.
- Chapters 5 and 6 provide initial guidance on how to apply the cloud extensibility model in SAP S/4HANA Cloud Private Edition or on-premise. Here we differentiate between greenfield SAP S/4HANA systems and converted SAP S/4HANA systems containing existing custom ABAP code.

This guide concentrates on the ABAP-based extensibility options (key user and developer extensibility on SAP S/4HANA and the SAP BTP ABAP environment).

For low-code/no-code tools on SAP BTP please refer to Low-Code/No-Code Development Tools in SAP community.

More information on Node.js and Java development on SAP Business Technology Platform as well as the SAP Cloud Application Programing Model (CAP) is referenced on the More Information page at the end of this document.

2 THE SAP S/4HANA CLOUD EXTENSIBILITY MODEL

SAP S/4HANA Cloud Public Edition allows customers and partners to take full advantage of all the innovations delivered by SAP without the need to take responsibility for the cloud infrastructure and operations. SAP manages all operation and lifecycle management tasks such as continuous feature delivery or providing hotfixes and regular upgrades to new software versions.

Consequently, the new cloud extensibility model as explained in chapter 1 is mandatory to ensure that customer and partner extensions continue to run without any change or adaptation effort independent of changes made by SAP.

Therefore, all extensions must adhere to the following rules to ensure cloud-readiness:

- **Rule 1** - Extensions can only use released remote or local SAP APIs. SAP keeps these APIs stable.
- **Rule 2** - SAP objects can only be extended via predefined extension points. SAP keeps these extension points stable. Modifications of SAP objects as in on-premise systems are no longer supported.
- **Rule 3** - Extensions can only use cloud-enabled and released technologies.

Many benefits of the classic extensibility model known from the on-premise world have been preserved:

- Extensions are built using the exact same programming model that SAP uses to develop the standard applications.
- The ABAP code of the SAP applications can be analyzed and inspected by customers and partners. This allows seamless end-to-end debugging of the extensions and provides a quick learning path for extension projects.

2.1 Overview of the extensibility options for SAP S/4HANA Cloud

In this chapter we will introduce the three ABAP-based extensibility options for SAP S/4HANA Cloud editions that follow the rules of the cloud extensibility model (→ Figure 2.1).

Type 1 and 2 extensions run directly on the SAP S/4HANA Cloud stack. They are implemented based on the technology stack of the core solution (ABAP Platform for SAP S/4HANA):

- Key user extensibility (1) for low-code/no-code extensions created by key users, such as adapting the user interface or adding custom fields.
- On-stack developer extensibility (2) using the SAP S/4HANA Cloud ABAP environment for developer extensions that are implemented in ABAP directly on the SAP S/4HANA Cloud technology stack.

Type 3 extensions run side-by-side to the core on SAP BTP:

- Side-by-side extensibility (3) for developer extensions using the development and runtime environments offered by SAP BTP. For ABAP this is the SAP BTP ABAP environment.

All three extensibility options are based on the usage of public SAP interfaces providing access to publicly released SAP APIs and SAP extension points.

Key user extensions and on-stack developer extensions have access to **local** public interfaces released by the underlying ABAP Platform or the SAP S/4HANA Cloud applications. These APIs can be released for key user extensibility, for on-stack developer extensibility or for both options.

Side-by-side extensions can access SAP S/4HANA Cloud business objects through **remote** SAP APIs (e.g., OData services). In addition, these side-by-side extensions on SAP BTP ABAP environment have access to the exposed local released SAP APIs and extension points of the underlying ABAP Platform stack.

Both SAP BTP ABAP environment and SAP S/4HANA Cloud ABAP environment are based on the same ABAP Platform stack and use the same local ABAP Platform interfaces.

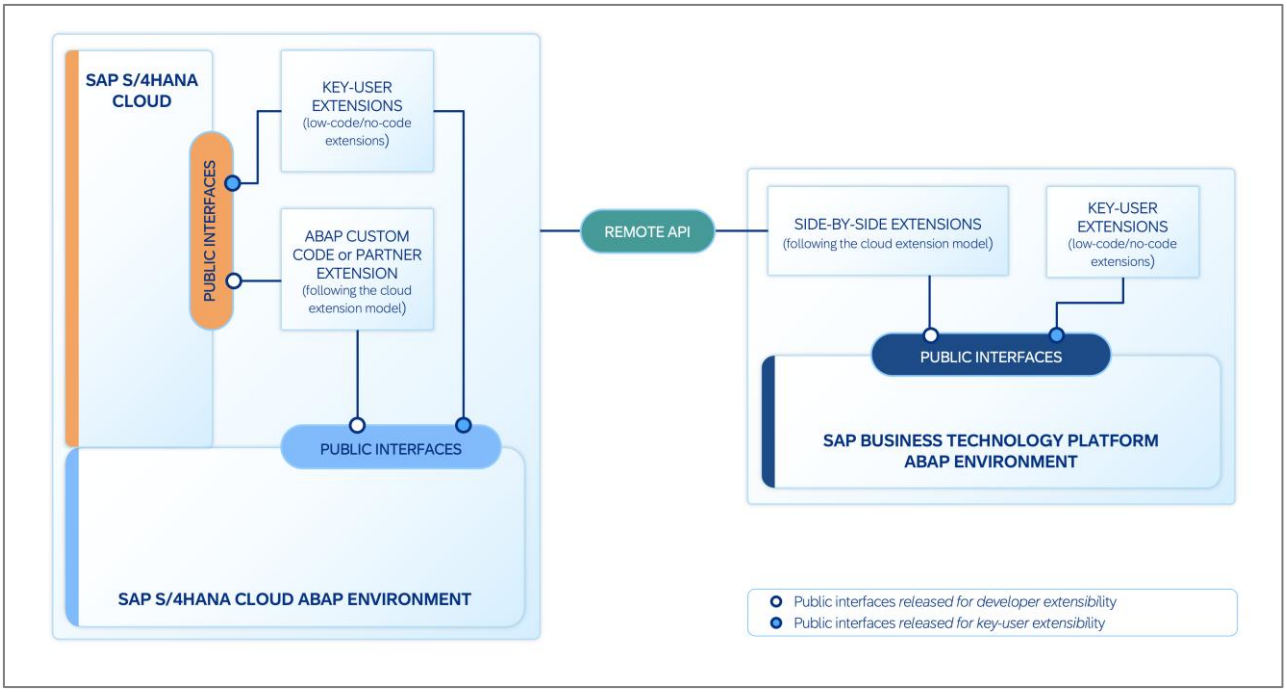


Figure 2.1 - Overview of extensibility options

The three extensibility options are not isolated from each other. In many scenarios they are combined, for example developing a side-by-side application in conjunction with a thin on-stack extensibility layer that offers more suitable remote APIs to access the SAP S/4HANA Cloud functionality.

2.2 Key user extensibility

SCENARIO	Low-code/no-code adaptation and extensions of SAP S/4HANA applications
USE CASES	Adapting UIs, adding custom fields, adding custom business objects and more.
PERSONA	Business expert, implementation consultant, citizen developer, key user.

Key user extensibility (formerly also known as in-app extensibility) empowers business experts to add extensions to SAP S/4HANA Cloud solutions without the need to dive deeply into the implementation details of the underlying SAP applications. Key users typically have a deep knowledge of the SAP S/4HANA processes and business configuration. The focus is on so-called *last mile extensions* extending SAP S/4HANA user interfaces, processes, or forms using low-code/no-code tools.

Key users typically have no or only limited coding skills and therefore do not need a fully integrated development environment, with capabilities such as versioning, dependency handling, refactoring, or debugging.

The main argument for using key user extensibility is that simple extensions can be realized quicker than with developer extensibility, because the communication overhead between the business expert (responsible for specification of the extension, and later for testing and approval) and the developer (responsible for development and developer test) is avoided.

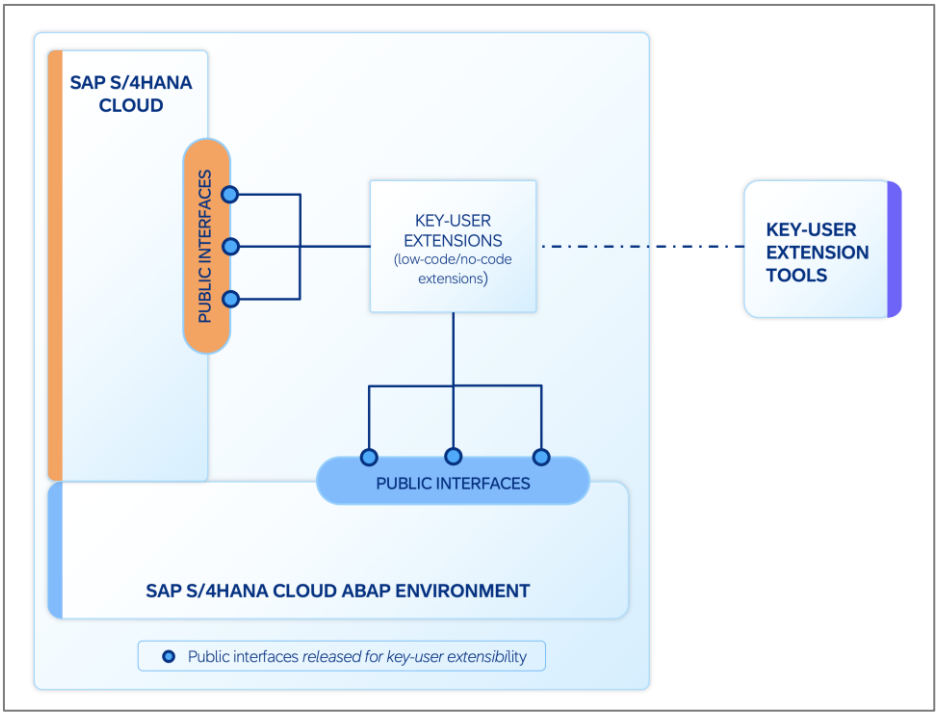


Figure 2.2 - Key user extensibility

With the provided key user tools the key user can achieve the following:

- Adapt the screen layout such as moving fields and field groups, hiding fields, changing labels etc.
- Add custom fields to business objects. The custom field is then available in the entire application stack (from the UI to the database tables or for developer extensibility).
- Add custom business objects to handle custom data with very little coding efforts.
- Add custom logic to validate the custom fields using cloud BAdIs.
- Add custom fields to a process group (e.g., from sales quotation and sales order to delivery and invoice) to provide a consistent end-to-end extensibility.
- Add custom Core Data Service (CDS) views and create new analytical applications (reports, KPIs, ...).
- Copy and adapt print and email form templates.

The adaptations made by a key user are registered in transport requests which can be propagated from a development environment to quality assurance and production.

The following screenshots illustrate typical key user tasks such as adding custom fields (→ Figure 2.3) or user interface adaptation for SAP Fiori apps (→ Figure 2.4).

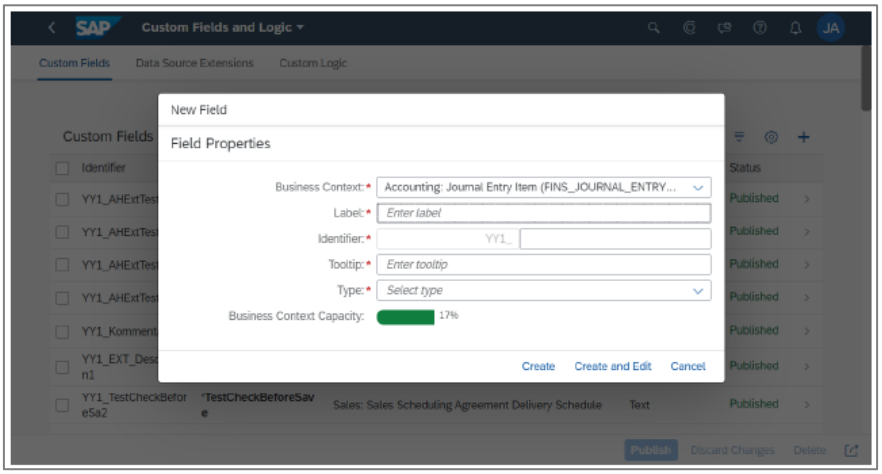


Figure 2.3 – Adding a custom field with key user extensibility

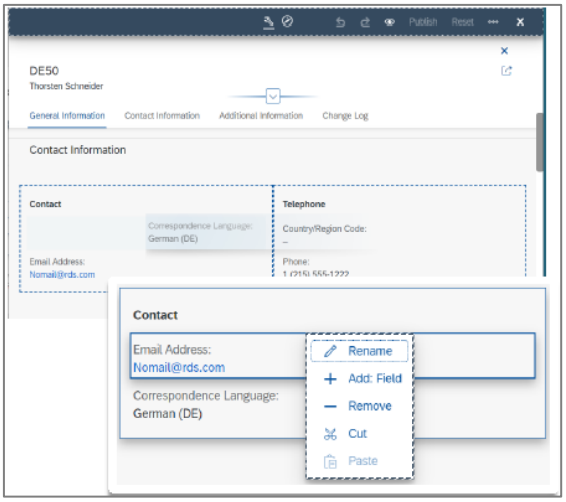


Figure 2.4 – Key user adaptation

The following blog posts and documentation provide more details on key user extensibility:

- The Key User Extensibility Tools of SAP S/4HANA.
- SAP S/4HANA Extensibility: A Learning Journey.
- SAP Help Portal - Extending an SAP Fiori Application.
- Extending SAP-delivered SAP Fiori elements apps using adaptation projects to create SAP S/4HANA app variants.

2.2.1 *On-stack developer extensibility with SAP S/4HANA Cloud ABAP environment*

SCENARIO	Custom ABAP development projects that need proximity or coupling to SAP S/4HANA data, transactions, or apps
USE CASES	Custom applications with frequent or complex SQL access to SAP S/4HANA data. Custom extensions running in the same logical unit of work[2] (LUW) as SAP applications. Tailored custom remote APIs or services which serve side-by-side SAP BTP apps. ABAP Cloud add-on products by partners.
PERSONA	ABAP developer.

On-stack developer extensibility[3] is intended for development projects requiring proximity to or coupling with SAP S/4HANA data, transactions, or apps. The requirements of the extension project go beyond the scope of key user extensions and therefore require full access to development capabilities like debugging, refactoring support, version control etc.

In contrast to side-by-side extensions, on-stack extensions are developed and run on the same software stack as the underlying SAP S/4HANA Cloud system. This allows extensions to access SAP S/4HANA logic and data via SAP extension points, local SAP APIs or via SQL queries.

Typical on-stack scenarios are extensions with frequent or complex SQL access to SAP data (e.g., SQL queries that join customer and SAP data), which cannot be realized easily by remote data access or data replication. Another typical on-stack pattern are extensions that store custom data in the same logical unit of work[2] as the extended SAP S/4HANA app.

[2] A sequence of programming steps and data updates distributed across multiple work processes, whose database changes are updated within a single database commit.
[3] On-stack developer extensibility uses the SAP S/4HANA Cloud ABAP environment.

On-stack developer extensibility allows ABAP developers to connect to an SAP S/4HANA Cloud system using the ABAP development tools (→ Figure 2.5). This feels almost like developing custom ABAP code on an SAP S/4HANA on-premise system. On-stack developer extensibility offers the standard ABAP development tools support like ABAP Unit, ABAP test cockpit, ABAP profiler, ABAP debugger and the well-known SAP lifecycle management (change and transport system).

However, with SAP S/4HANA Cloud ABAP environment, extensions are developed using the new ABAP Cloud development model (→ Chapter 4). The ABAP Cloud development model ensures that no SAP object is modified and that only local publicly released SAP APIs and extension points are used in the extensions.

SAP S/4HANA Cloud and the ABAP Platform offer a large set of local APIs and extension points which can be used in on-stack extensions. Developers can explore these in the *released objects tree* of ABAP development tools for Eclipse (→ Figure 2.6). Additionally, the SAP API Business Hub[4] provides a section for developer extensibility and SAP CDS views showing the core local SAP APIs.

Depending on the use case, one of these options can be used to expose the functionality built using on-stack developer extensibility on an SAP Fiori UI:

- Create a custom SAP Fiori elements or freestyle SAPUI5 app using SAP Fiori tools.
- Extend an SAP-delivered app, e.g., with additional fields using Developer Adaptation[5].
- Extend an SAP-delivered SAP Fiori elements app using ABAP CDS annotations or XML annotations.

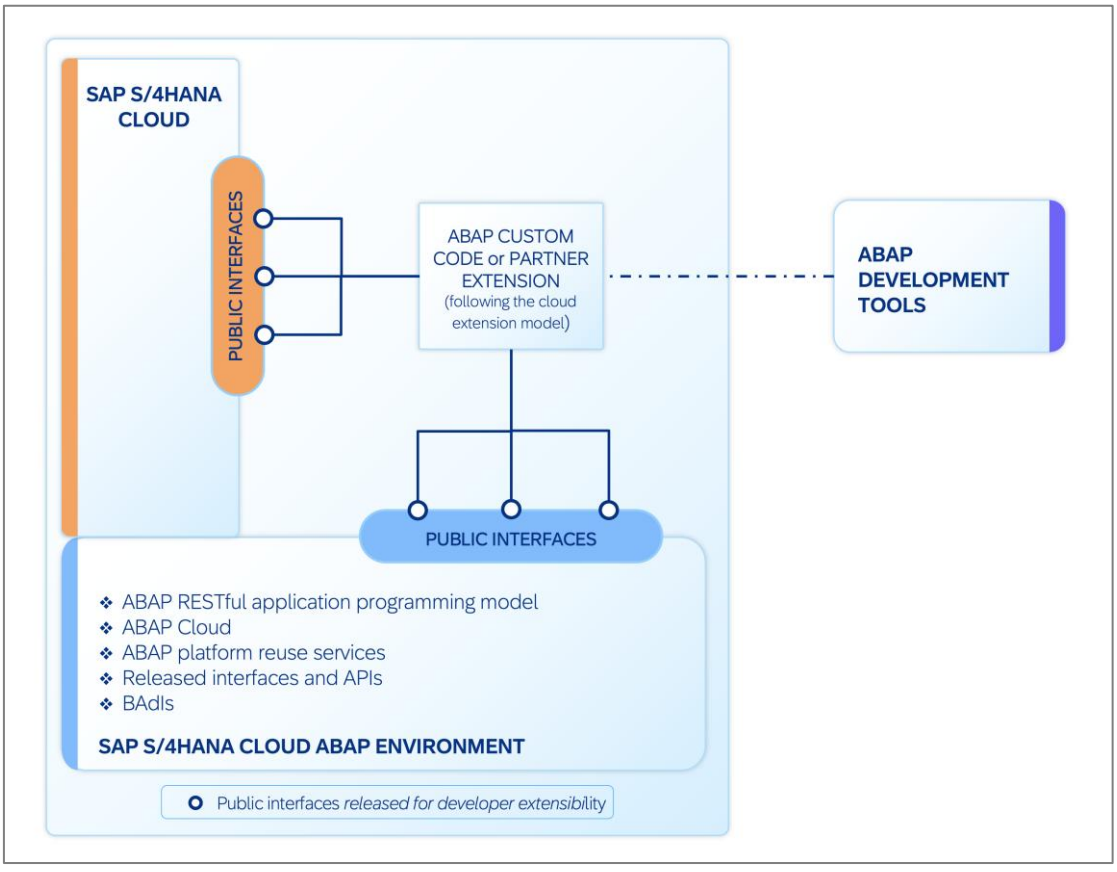


Figure 2.5 - SAP S/4HANA Cloud ABAP environment

[4] Business Object Interface | SAP S/4HANA Cloud | SAP API Business Hub.
[5] Note that Developer Adaptation is not yet available for SAP S/4HANA Cloud.

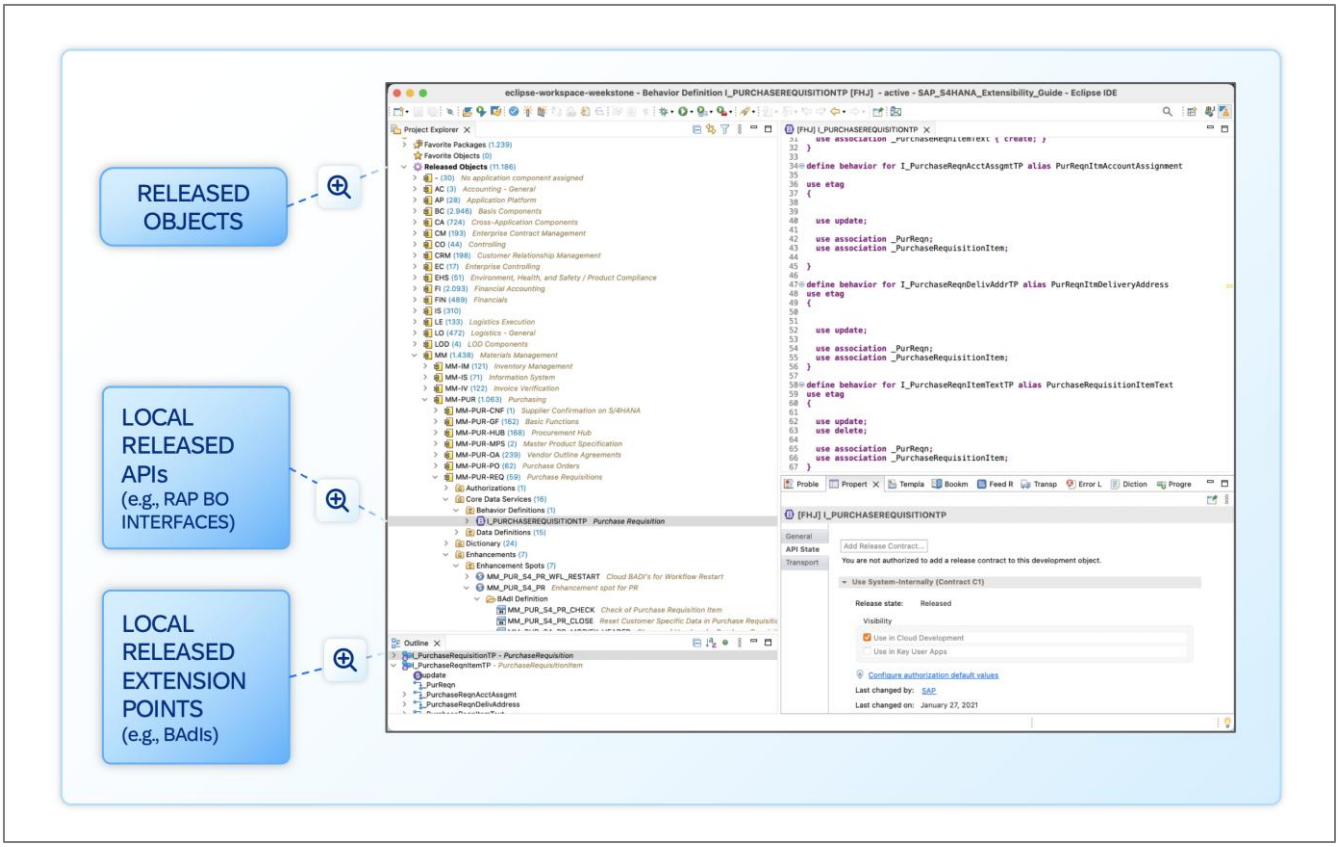


Figure 2.6 - Released object tree in ABAP development tools

2.3 Side-by-side extensibility using the SAP BTP ABAP environment

SCENARIO	Loosely-coupled applications and partner SaaS solutions
USE CASES	Custom applications for a separate target group (no ERP users). Custom application workload that shall run separated from ERP. Custom applications needing proximity to intelligent SAP BTP services Solutions integrating with several ERP systems and cloud services. SaaS applications provided by partners.
PERSONA	ABAP developer.

The SAP BTP ABAP environment provides the ABAP Platform as a service on SAP BTP. ABAP-minded customers and partners can reuse their ABAP skillset to build new cloud solutions, or to transform already existing on-premise ABAP assets to the cloud.

The cloud applications and extensions run side-by-side to the extended SAP S/4HANA system. This model is the preferred option for scenarios which are loosely coupled to SAP S/4HANA data, transactions, or apps.

A typical side-by-side use case is the hub scenario. A hub solution integrates with several ERP systems and cloud services, e.g., to consolidate SAP S/4HANA data and trigger derived actions via separate cloud services. By their very nature, hub scenarios are loosely coupled and can run side-by-side with the SAP S/4HANA systems.

Another use case is that partners want to provide a SaaS solution and therefore need to operate their service independently of SAP S/4HANA Cloud.

SAP BTP ABAP environment is specially optimized for large SAP or partner SaaS solutions which benefit from the multitenancy offerings and services for partners to run and operate the solution.

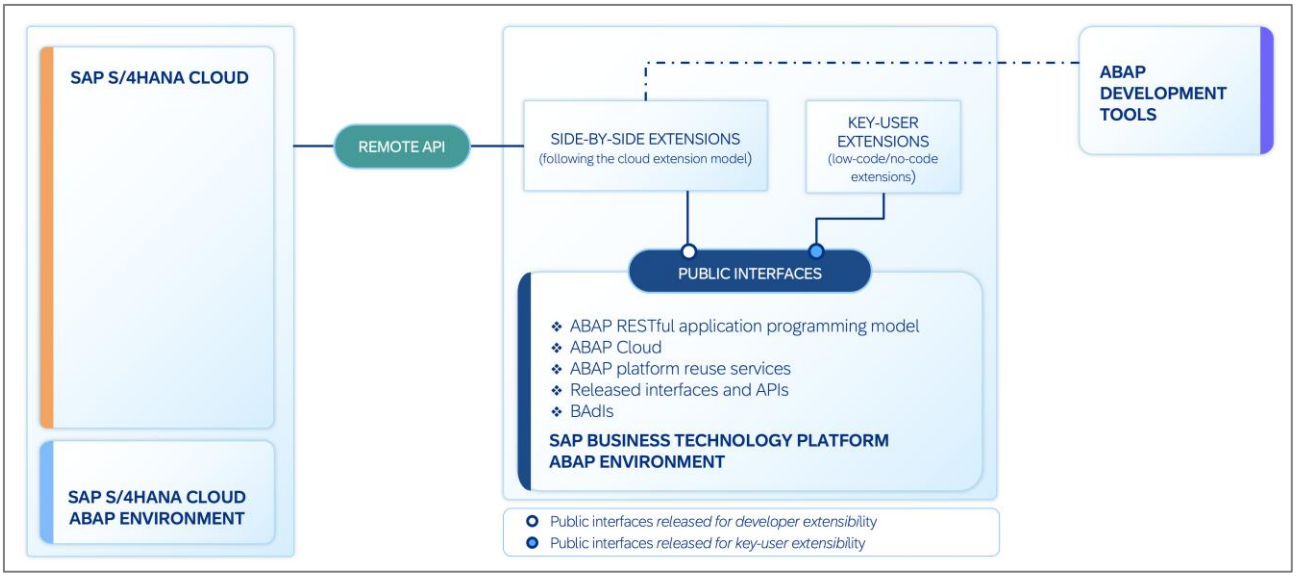


Figure 2.7 – Side-by-side extensions with SAP BTP, ABAP Environment

An ABAP developer accesses SAP BTP with the ABAP development tools and uses the ABAP Cloud development model (→ Chapter 4) to build SAP Fiori apps[6] or services based on the ABAP RESTful application programming model (RAP) (→ Figure 2.7). The ABAP Cloud development model ensures that only released local APIs of the underlying ABAP Platform can be used.

One main difference compared to the on-stack extensibility model is that accessing business objects of SAP S/4HANA Cloud is only possible using **remote APIs** which are published in the SAP Business Accelerator Hub (https://api.sap.com/).

2.4 Summary of the extensibility options for SAP S/4HANA Cloud

The following picture shows a recap of the three ABAP-related extensibility options which are available in SAP S/4HANA Cloud, and which were introduced in the previous sections.

	KEY USER EXTENSIBILITY Business expert, implementation consultant, citizen developer, key user	ON-STACK DEVELOPER EXTENSIBILITY ABAP developer	SIDE-BY-SIDE EXTENSIBILITY ABAP developer
SCENARIO	Low-code/no-code adaptations and extensions of SAP S/4HANA applications	Custom ABAP development projects that need proximity or coupling to SAP S/4HANA data, transactions, or apps	Loosely-coupled applications and partner SaaS solutions
USE CASES	Adapting UIs, adding custom fields, adding custom business objects etc.	Custom applications with frequent or complex SQL access to SAP S/4HANA data Custom extensions running in the same logical unit of work (LUW) as SAP applications Tailored custom remote APIs or services which serve side-by-side SAP BTP apps	Custom applications for a separate target group (no ERP users) Custom application workload that shall run separated from ERP Custom applications needing proximity to intelligent SAP BTP services like machine learning, AI etc. Solutions integrating with several ERP systems and cloud services SaaS applications provided by partners
BENEFIT	Fully managed and integrated in SAP S/4HANA Cloud No or only very basic development skills required	Development of extensions inside the SAP S/4HANA Cloud system No remote access or data replication Use and extend released SAP S/4HANA Cloud objects	Decoupled extensions independent of SAP S/4HANA Cloud operation and lifecycle management
	On-stack extension domain		Side-by-side extension domain

Figure 2.8 - Summary of ABAP-related extensibility options in SAP S/4HANA Cloud

[6] The corresponding SAP Fiori UI can be created using SAP Fiori elements or SAPUI5 freestyle.
→ Developing Apps with SAP Fiori Elements and → SAP UI5 documentation)

3 WHEN TO USE WHICH CLOUD EXTENSIBILITY OPTION

This chapter describes when to use which extensibility option. In many cases a combination of the extensibility options is needed to solve the extensibility task.

The major aspects to consider are:

- **Extension use case** - is the extension a new application, or an extension to an SAP app?
- **Extension architecture** - is the extension loosely or tightly coupled to SAP S/4HANA?
- **Extension scope & size** - who provides the extension for which purpose? For example, is the extension a small extension that is provided by key users in a line of business organization, a custom development project that is provided by a development organization, or a partner SaaS application that is provided to many customers (even independent from the core product)?

The following Figure 3.1 provides a decision tree to select the right extensibility option.

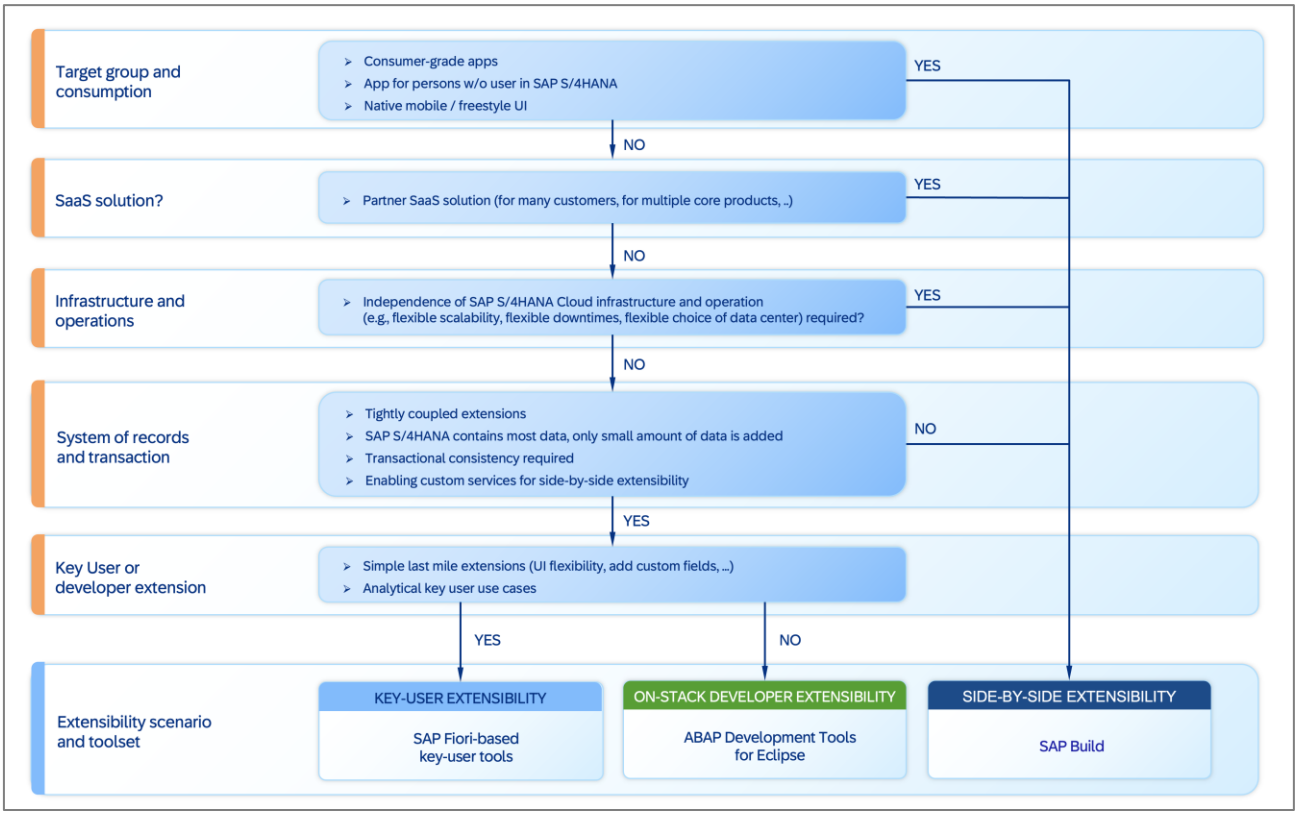


Figure 3.1 - Sequence diagram on how to find the right extensibility options.

The first three options describe typical side-by-side scenarios on SAP BTP:

- **External target group or consumption via mobile and consumer-grade UIs:** this includes native mobile apps, apps that need freestyle UI development for consumer-grade UIs, or apps for persons without a named user in SAP S/4HANA.
- **Partner SaaS solution:** a partner solution shall be provided as a cloud service operated by a partner. This use case can only be realized separated from the SAP S/4HANA Cloud operation model and is therefore always a side-by-side scenario. SAP BTP provides multitenancy concepts to minimize the cost of service and provides services to build and operate the partner SaaS solution on SAP BTP.
- **Independent infrastructure and operation:** side-by-side on SAP BTP, customers and partners are independent of the cloud infrastructure and operation model of SAP S/4HANA Cloud. The main benefits are:
  - Customers can flexibly access resources for the respective runtime environment. Resources can be scaled without impact on the core system. This means customers can physically outsource the

workload and resource consumption of the extensions to safeguard the performance of their ERP system.
- Customers can schedule the downtime of their extensions independently of SAP S/4HANA Cloud.
- Customers can choose the data center for the SAP BTP extension. This can be of interest if e.g., the co-location of the extension with certain SAP BTP services like AI or machine learning is more important than the proximity to SAP S/4HANA Cloud.
- Customers can freely choose which services, libraries or products shall be used or integrated in their SAP BTP extension.

The next important selection criteria are the extension requirements concerning proximity to and coupling with SAP S/4HANA Cloud data, transactions, and apps. Here we distinguish between loosely and tightly coupled extensions:

Loosely coupled extensions and custom apps:

- Stand-alone applications or new process steps with occasional usage of SAP S/4HANA Cloud data.
- Data is replicated to SAP BTP or read via remote API calls from the core product.
- Custom data is not changed together with core data (no transactional consistency required).
- Data hubs or integration hubs that integrate, collect, or distribute data from many systems across company units are typical loosely coupled scenarios.

For loosely coupled extensions and custom apps, side-by-side extensibility on SAP BTP is the default choice.

Tightly coupled extensions need coupling and proximity to SAP S/4HANA Cloud data, transactions, or apps. Typical patterns for tightly coupled extensions are:

- Frequent read-access or changes to SAP data (many roundtrips).
- Reading of customer data and SAP data in complex SQL queries (e.g., joins) and with high data volume.
- Required transactional consistency when customer data and SAP data is changed jointly.
- Extends the UI of an SAP app, extends the SAP data model, implements the extension point of an SAP app.
- Uses local SAP APIs to build an own tailored remote service for a side-by-side SAP BTP solution.

Tightly coupled scenarios shall be realized with key user or on-stack developer extensibility.

The final decision step for tightly coupled extension is to choose between on-stack developer and key user extensibility:

- Key user extensibility shall be chosen for extensions by single business experts as explained in chapter 2.2.
- Developer extensibility shall be chosen for extensions requiring professional ABAP development as explained in chapter 2.2.1.

Customers may additionally define a company-specific policy that considers available business expert and developer resources as well as the extension scenario to choose between these options.

3.1 Additional aspects to consider

Finally, the following aspects should be considered for extension projects:

- **Consider the knowledge and experience of your development teams.** With the ABAP-based on-stack and side-by-side extensibility options, you can leverage the ABAP skills of your development teams. Loosely coupled extensions on SAP BTP can similarly be realized in other development environments, but that would require a different set of skills.
- **Balance between homogenous versus hybrid extensions:** customers and partners must find the right balance between homogenous (only on-stack or only side-by-side) versus hybrid (mixture of on-stack and side-by-side) extensions. Homogenous extensions should be preferred for simpler lifecycles, but in some cases, hybrid extensions are the best solution.
- **Layering of key user and developer extensibility:** on-stack extensions are often a mixture of key user extensions and developer extensibility. For these scenarios customers must consider that these extension types are layered (key user extensions on top of developer extensions). Objects built with developer

extensibility can be released for key user extensibility, but not vice versa. Consequently, objects that are to be re-used in both layers must be built with developer extensibility [7].

3.2 Examples

3.2.1 Key user extensibility

A customer adds custom fields for new package properties (color of the ordered package) and customer member status (level of membership) to the standard SAP S/4HANA Cloud sales process.

These new custom fields are then part of all standard SAP S/4HANA Cloud sales screens and all process steps. This includes prefilling, validating, and using the custom fields along the entire sales process (propagate along sales quotation, sales order, outbound delivery, and billing document).

Additionally, the customer creates custom analytical reports and a custom form template for the order confirmation printout. The new custom fields are part of the analytical reports and the printout as well.

This comprehensive example can be completely realized with key user extensibility tools, see:

Key User Extensibility in SAP S/4HANA Cloud Sales | SAP Blogs

3.2.2 On-stack developer extensibility

A customer simplifies the process for recurring employee purchases below a certain amount. He uses the ABAP RESTful application programming model (→ chapter 4.1) and SAP Fiori elements to implement an employee self-service app on SAP S/4HANA Cloud ABAP environment.

The app uses local publicly released procurement APIs and extension points to validate, create and release the purchase orders automatically.

3.2.3 Side-by-side extensibility

A customer uses multiple SAP S/4HANA systems in several of their subsidiaries to manage his purchasing processes. To improve the purchasing process, he develops a central purchasing approver determination application on SAP BTP.

Based on a set of business rules, the application identifies the allowed approvers for specific purchasing documents and provides the information as a central remote OData API to the distributed purchasing processes.

[7] Exception: custom fields created with key user tools are usable in both layers. For details on the interaction between developer and key user extensibility, see the documentation, section Key User Extensibility and Developer Extensibility.

4 THE ABAP CLOUD DEVELOPMENT MODEL

With the launch of SAP BTP ABAP environment, SAP introduced a new **ABAP Cloud development model** which leads to modern, cloud-ready, and upgrade-stable ABAP applications and extensions. ABAP Cloud development means first and foremost:

- Only publicly released SAP APIs and extension points can be used.
- No modifications to SAP objects are allowed.
- Use ABAP development tools for Eclipse (ADT) as your ABAP development environment.
- Build RAP (ABAP RESTful application programming model) based SAP Fiori apps or services.
- Technologies like Dynpro or Web Dynpro are not supported.

The ABAP Cloud development rules are enforced via:

- ABAP compiler and ABAP runtime checks: the ABAP Cloud development model uses ABAP Cloud as the language version[8]. This cloud-optimized ABAP language defines the set of supported ABAP statements and launches syntax- or runtime errors if e.g., a non-publicly released SAP API is used (→ Chapter 4.2).
- ABAP authorization checks: ABAP Cloud redefined the authorizations in the ABAP Cloud developer role (e.g., no authorization to change SAP objects)[9].

The ABAP Cloud development model is mandatory for ABAP apps developed on SAP BTP and for extensions built on SAP S/4HANA Cloud Public Edition and **can** be enabled in SAP S/4HANA Cloud Private Edition and SAP S/4HANA on-premise when desired.

This chapter will provide details about the main building blocks of this model:

- The ABAP RESTful application programming model (RAP).
- Analytics in ABAP Cloud.
- ABAP Cloud – the cloud-optimized ABAP language version.
- Reuse services.
- Identity and access management.
- Connectivity.
- Business APIs and extension points.
- Business configuration.

The primary purpose of RAP is to build and expose back-end services based on semantic data models. The services are then consumed by SAP Fiori apps or exposed as Web APIs. The RAP programming model is therefore at the core of every extension project (→ Chapter 4.1).

For the implementation layer ABAP Cloud is used (→ Chapter 4.2). In addition, SAP offers a large library of reuse services (→ Chapter 4.4).

SAP Fiori applications follow a role-based approach which can be properly modelled with the identity and access management capabilities of the underlying ABAP Platform (→ Chapter 4.4.3).

Since extensions typically interface, interact and integrate with SAP's business solutions a large set of connectivity options and protocols is needed (→ Chapter 4.4.4). Stable business APIs and extension points provide the semantic link to the standard application (→ Chapter 4.5).

Last, but not least, business configuration frameworks and tools provide the flexibility to deliver extensions for a wider range of business scenarios and use cases.

In the following we will examine the various building blocks in more detail.

[8] ABAP Cloud is also referred to as ABAP for Cloud Development, ABAP classic is also referred to as Standard ABAP.
[9] The ABAP Cloud development model for SAP HANA Cloud private edition is available with release ≥ 2022 and for SAP HANA on-premise with ABAP Platform 2022.

4.1 The ABAP RESTful application programming model

In recent years the requirements for business applications and related technologies have significantly changed. End users expect enhanced user experience qualities. For example, continuous work - start working at home, continue during commuting and finalize the task at the company or the ability to accomplish tasks on different device types. In addition, end users expect personalized, web-like business apps. So, user experience is taking center stage in a constantly evolving business landscape.

On the other hand, modern business apps must be able to run in hybrid system landscapes supporting cloud, on-premise, and virtualization. In addition, these modern business apps must continue to meet the requirements of large enterprises including all the steps of the product lifecycle.

To satisfy all these business needs and to make the development of modern business applications efficient, a standardized ABAP programming model that guides ABAP developers and optimally supports cloud operation, SAP Fiori, and SAP HANA, has become a pressing need.

To meet this need, SAP offers the ABAP RESTful application programming model (RAP). RAP is the new standard ABAP programming model used at SAP to build SAP S/4HANA Fiori apps and services. RAP is the recommended programming model for customers and partners for all ABAP based SAP S/4HANA extensions[10].

4.1.1 The big picture

RAP is a set of concepts, tools, languages, powerful frameworks, and best practices provided on the ABAP Platform for efficient and rapid development of innovative and cloud-ready enterprise applications, as well as the extension of SAP standard applications in the cloud and on-premise. RAP is deeply integrated with the ABAP language; moreover, the ABAP development tools have been optimized to support the RAP development flow. The main RAP building blocks are:

- **ABAP Core Data Services** (CDS) used for SAP HANA optimized queries, to define semantically rich data models for all application domains, and to define the transactional behavior of the modeled entities.
- The modernized and extended **ABAP language** used to implement business logic.
- The **OData protocol** used for stateless communication.
- The concept of **business object (BO)** used for building transactional applications.
- The concept of **business service** used to define services.

RAP offers a standardized development flow in the modern, Eclipse-based ABAP development tools (ADT) and a rich feature set for building applications from different domains, either from scratch or by reusing existing custom code. Built-in capabilities such as ABAP unit tests (→ Unit testing with ABAP unit), the ABAP cross trace tool (→ Working with ABAP Cross trace tools), and knowledge transfer documents (→ Documenting ABAP development objects | SAP Blogs) are offered along the RAP development stack to cover core software quality concepts such as testability, supportability and documentability.

Different types of services can be built with RAP:

- OData-based services for building role-based and responsive SAP Fiori apps.
- OData-based services for exposure as Web APIs.
- Information access based (InA) services for analytical clients.
- Business events.

Figure 4.1 gives an overview of the RAP big picture. A detailed description of the architecture can be found in the RAP developer guide: RAP Developer Guide on SAP Help Portal

[10] The ABAP RESTful application programming model is available in SAP S/4HANA Cloud Public Edition, SAP Business Technology Platform and for SAP HANA Cloud private edition starting with release 2022 and for SAP HANA on-premise with ABAP Platform 2022.

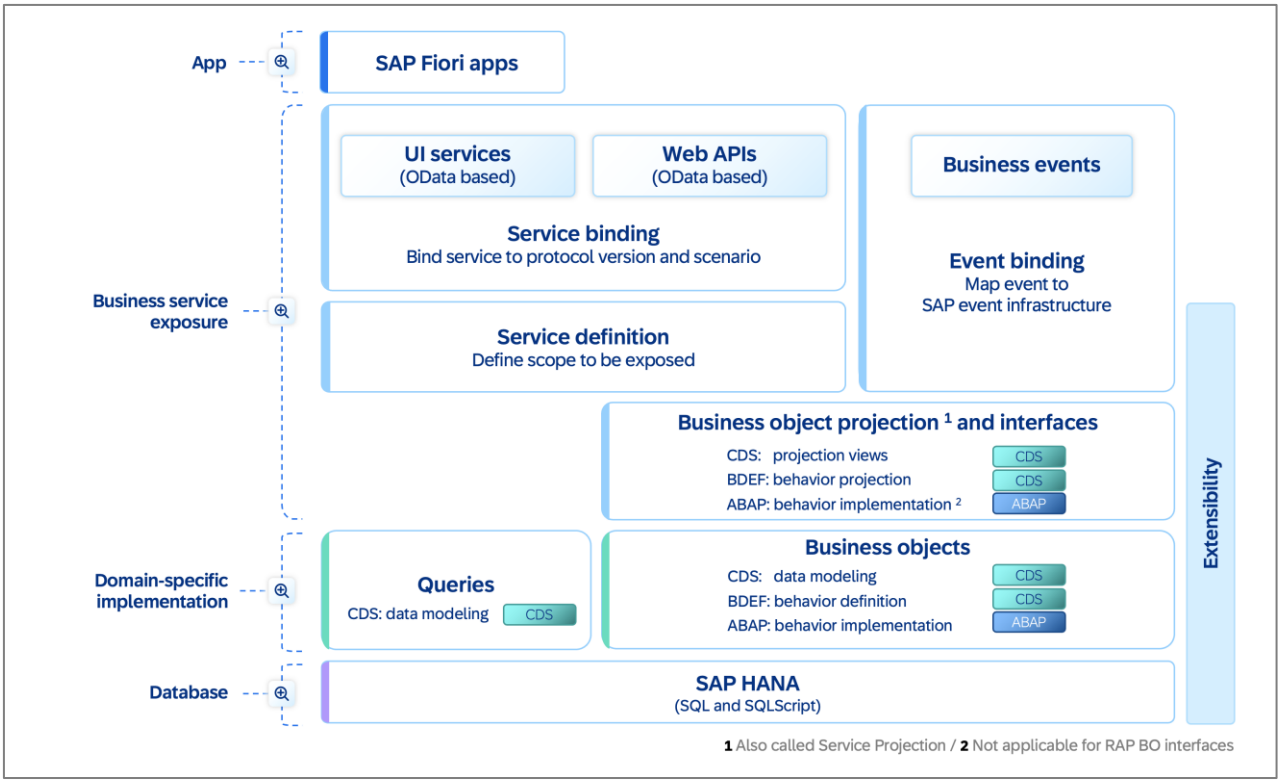


Figure 4.1 - RAP big picture

Cloud-ready and upgrade-stable apps and services can be built when using RAP together with ABAP Cloud, publicly released SAP APIs and objects.

SAP uses RAP to provide local APIs on SAP S/4HANA Cloud for central business objects like *Sales Order*. These APIs can be used for on-stack developer extensibility to e.g., create or change a sales order.

Examples of these local RAP-based SAP APIs can be found in the developer extensibility section of SAP business accelerator hub[11].

Find more information on RAP here: Modern ABAP Development with the ABAP RESTful Application Programming Model

A more detailed description on supported RAP implementation scenarios and when to use which option can be found in this blog: Modernization with the ABAP RESTful Application Programming Model (RAP) | SAP Blogs

4.1.2 Extensibility of RAP business objects (RAP BO)

Business objects developed with RAP can have built-in extensibility options[13]. They can be extended as part of the developer extensibility model. This is particularly useful when SAP S/4HANA business objects or RAP business objects from partner applications must be extended. The extensibility options for RAP business objects are of course designed for lifecycle stability and a separation of concerns between the original RAP BO and its extensions.

By default, extensibility of a RAP BO is disabled. Therefore, RAP BO extensibility must be explicitly enabled on the original BO[12], to allow extensions at dedicated extension points. The extension options are developed to be controlled on a fine-granular level, to give full control over which parts of a BO can be extended with which kind of extension. Trying to define extensions for non-enabled extension points of the original BO leads to a syntax error and it cannot be activated[13].

---

[11] https://api.sap.com/products/SAPS4HANACloud/developerextensibility/bointerface
[12] The original RAP BO is the base BO on which the RAP BO extension bears
[13] Extensibility enablement of RAP BOs created by SAP can only be enabled or changed by SAP and not by customers or partners

It is crucial to understand that:

- Each RAP BO extension can only define behavior specifically for **its own** extension elements and operations and is independent of all other RAP BO extensions.
- RAP BO extensions cannot redefine or change any behavior for any elements of the original RAP BO.

From the consumer's perspective, the original RAP BO and all associated RAP BO extensions appear as one large RAP BO.

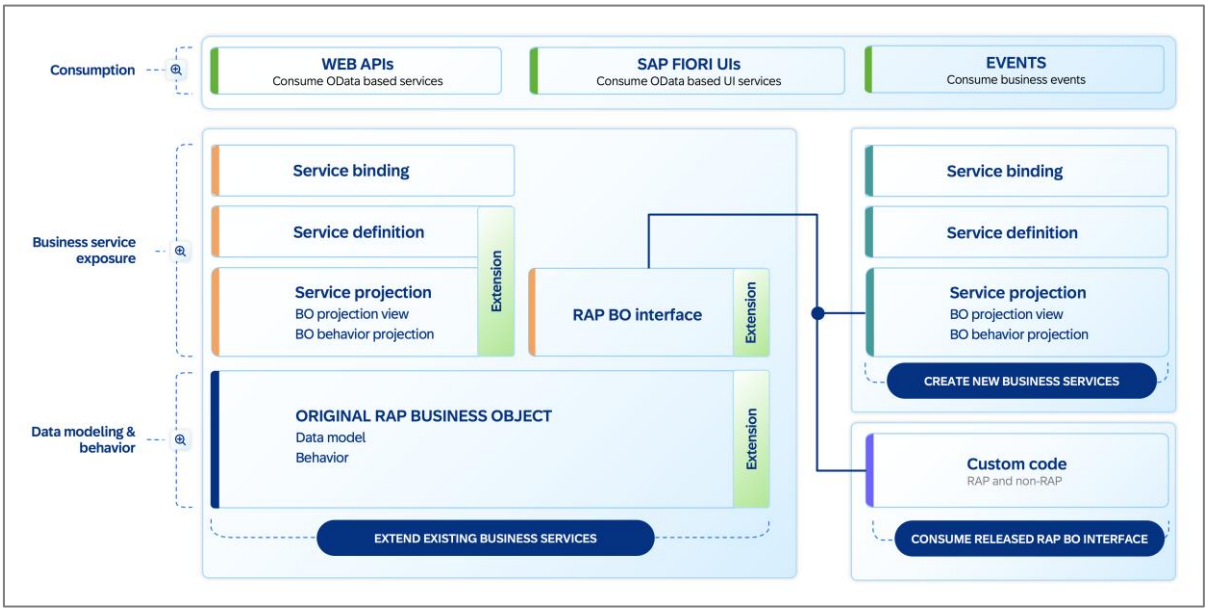


Figure 4.2 - RAP Extensibility options overview

The major extensibility scenarios for RAP BOs, as shown in Figure 4.2 are:

- Adding a new field including related business logic.
- Enriching the business logic of the BO.
- Adding a new action resulting in a new button in an SAP Fiori application.
- Adding a completely new entity to an SAP BO.

The extensibility options for RAP BOs are described in Table 4.1 below and links to further information are provided.

EXTENSIBILITY OPTION	EXTENSIBILITY TASK
DATA MODEL EXTENSION Build full-stack data model extensions by adding new fields and associations including corresponding behavior characteristics and authorization control.	**Extensibility enablement**: Add annotations and extension include structures to the original RAP BO to enable data model extensions. Extension include structures are DDIC structures that are included in all relevant structures and database tables and can be used by the extender for structure extends. For more information about enabling full-stack data extensibility, see Extensibility-Enablement for CDS Data Model Extensions. For an implementation example, see Enabling Field Extensions: **Extension development**: Extends the original RAP BO with new fields or associations including field characteristics depending on the options defined by the extensibility-enabler.

EXTENSIBILITY OPTION	EXTENSIBILITY TASK
	For more information about how to develop data model extensions, see CDS Data Model Extensions. For an implementation example, see Develop Field Extensions.
BEHAVIOR EXTENSION Build additional behavior like new validations, determinations or actions including dynamic feature control and other field-related behavior.	**Extensibility enablement**: Enables data model extensibility and behavior extensibility on the original RAP BO. For more information about how to enable your BO for behavior extensions, see Extensibility Enablement for Behavior Extensions. For an implementation example, see Enabling Non-Standard Behavior and Field-Related Behavior and Enabling Standard Behavior Extensions. **Extension development**: Extends the original RAP BO with new validations, determinations or actions depending on the options defined by the extensibility-enabler. For more information about how to develop different behavior extensions, see Behavior Extensions. For an implementation example, see Develop Behavior Extensions.
NODE EXTENSION Build additional BO nodes with own behavior and data model with node extensibility.	**Extensibility development**: Enables node extensibility on the original RAP BO. For more information about node extensibility enabling, see Extensibility-Enablement for the Node Extensibility. For an implementation example, see Enabling Node Extensions.
	Extension provisioning: Extend the original BO with new nodes that have their own data model and behavior. For more information about how to develop node extension, see Node Extensions.

Table 4.1 - Extensibility options for RAP BOs

4.2 Analytics in ABAP Cloud

Analytics enables multidimensional reporting and data analysis in ABAP in two main scenarios:

Embedded analytics

Embedded analytics allows you to build sophisticated and complex analytical data models to evaluate and analyze business data in your ABAP system. In embedded analytics, the ABAP analytical engine is part of the software stack and operates on the same data persistence layer as the transactional applications. The analytical queries operate directly on the business data without data replication to an external data warehouse system. Instead, the real-time business data is queried to always evaluate the most recent changes and trends in your business data.

Side-by-side analytics scenarios

In a side-by-side analytics scenario, the business data is replicated from an analytical source system to other analytical clients (like SAP Analytics Cloud) or to a data warehouse (like SAP Data Warehouse Cloud). The business data is replicated from an ABAP system based on a predefined integration scenario to allow seamless data exchange. The analytical data model itself is implemented outside of the ABAP system and only the analytical data of the source system is used and is part of the analytical consumption scenario.

Another option to realize side-by-side analytics scenarios is data federation. Data federation is used in scenarios where live access to the source system is required, e.g., to access the most recent data without a time lag, or to rely on data access control mechanisms inside the source system. Data calls are not executed locally in the target system but delegated from there to one or more source systems (where the data is located) and executed on source side.

The following chapters focus on the embedded analytics use case and highlight the advantages of this approach.

4.2.1 The big picture of embedded analytics in ABAP Cloud

Embedded analytics unites the advantages of CDS data modeling and the analytical capabilities of the analytical engine to create real-time evaluations based on business data within the same ABAP system: The CDS framework is at the core of the ABAP Cloud development model, enabling the development of cloud-ready and lifecycle-stable analytical data models for all SAP S/4HANA deployment options and for SAP BTP ABAP environment.

The main building blocks of an analytical application are[14]:

- **Dimensions**: master data-like data used as attributes in the data analysis.
- **Cubes**: an analytical interface view that is used in analytical queries. Cube views are at the center of the multi-dimensional data model.
- **Hierarchies**: define a data hierarchy and can be used for drill-down or roll-up operations to change the data granularity of the result set.
- **Analytical queries** (CDS Analytical Projection Views)[15]: the initial layout of the initial data set and granularity of the query, based on a cube.
- **Service binding**: the Information Access Protocol (InA) is the protocol used to expose an analytical data model for consumption by analytical clients like SAP Analytics Cloud.

The following graphic provides a more detailed overview of the main building blocks used to build a data model for embedded analytics:

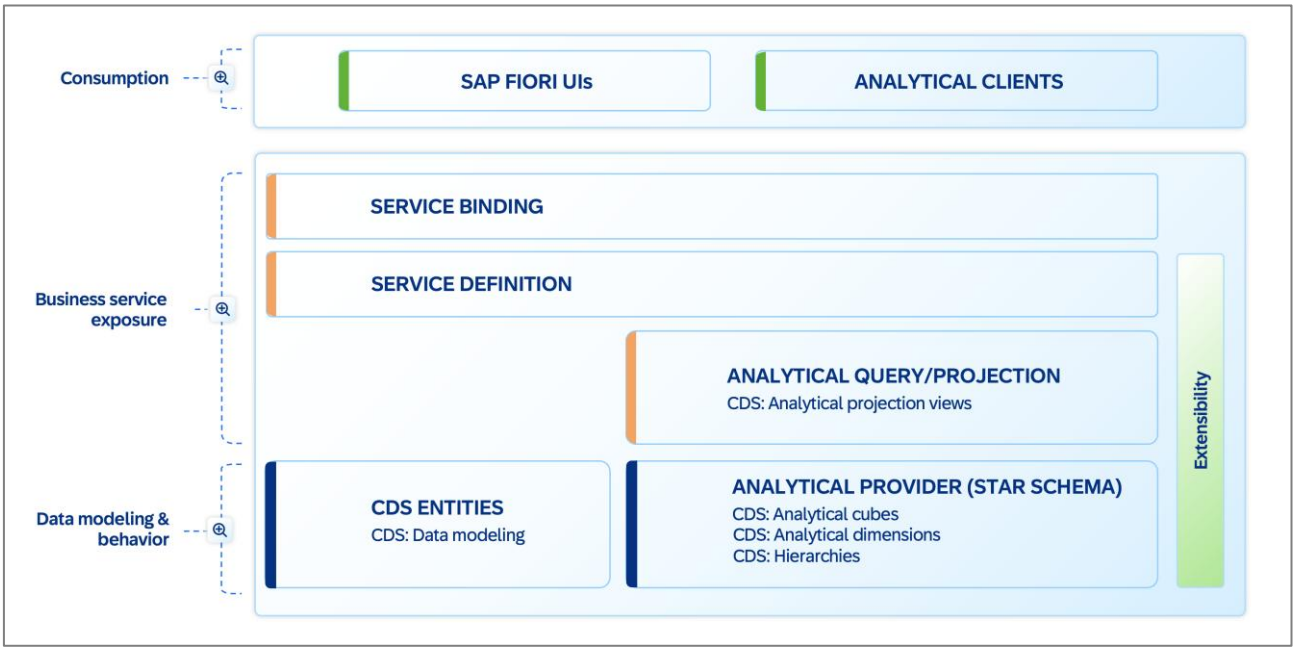


Figure 4.3 - Detailed overview of the main building blocks used to build a data model for embedded analytics.

An analytical data model is multidimensional and is typically modeled in a star or snowflake schema to enable easy data analysis at different levels of granularity[16]. CDS allows easy data modeling and data preparation for display in dashboards such as those in SAP Analytics Cloud.

---

[14] For more information, see https://help.sap.com/docs/BTP/65de2977205c403bbc107264b8eccf4b/483cc0637280445b98e98775dd0383b1.html

[15] For more information, see https://blogs.sap.com/2022/02/18/cds-analytical-projection-views-the-new-analytical-query-model/comment-page-1/

[16] For more information, see https://blogs.sap.com/2022/11/30/embedded-analytics-with-abap-cloud-a-brief-overview-part-1/

For developer extensibility the analytical data models and services are developed in the modern, Eclipse-based ABAP development tools (ADT). ADT offers an efficient development flow, and a rich feature set for creating analytical services.

Key user extensibility for analytical scenarios offers a rich SAP Fiori-based toolset for key users for creating and managing custom analytical queries, analytical reports, KPIs and analytical stories.

See also: Key User Extensibility Overview | sap.com and SAP S/4HANA embedded analytics – User Roles | Blogs).

4.2.2 Extensibility for analytical data models

Extensibility is a built-in quality for all ABAP data models based on CDS. Based on an opt-in approach, the entire data model stack can be enabled for different extensibility use cases depending on the specific data model requirements. Each layer in the stack offers possibilities for dedicated extension points, enabling a well-defined separation of concerns between the original data model and the extensions.

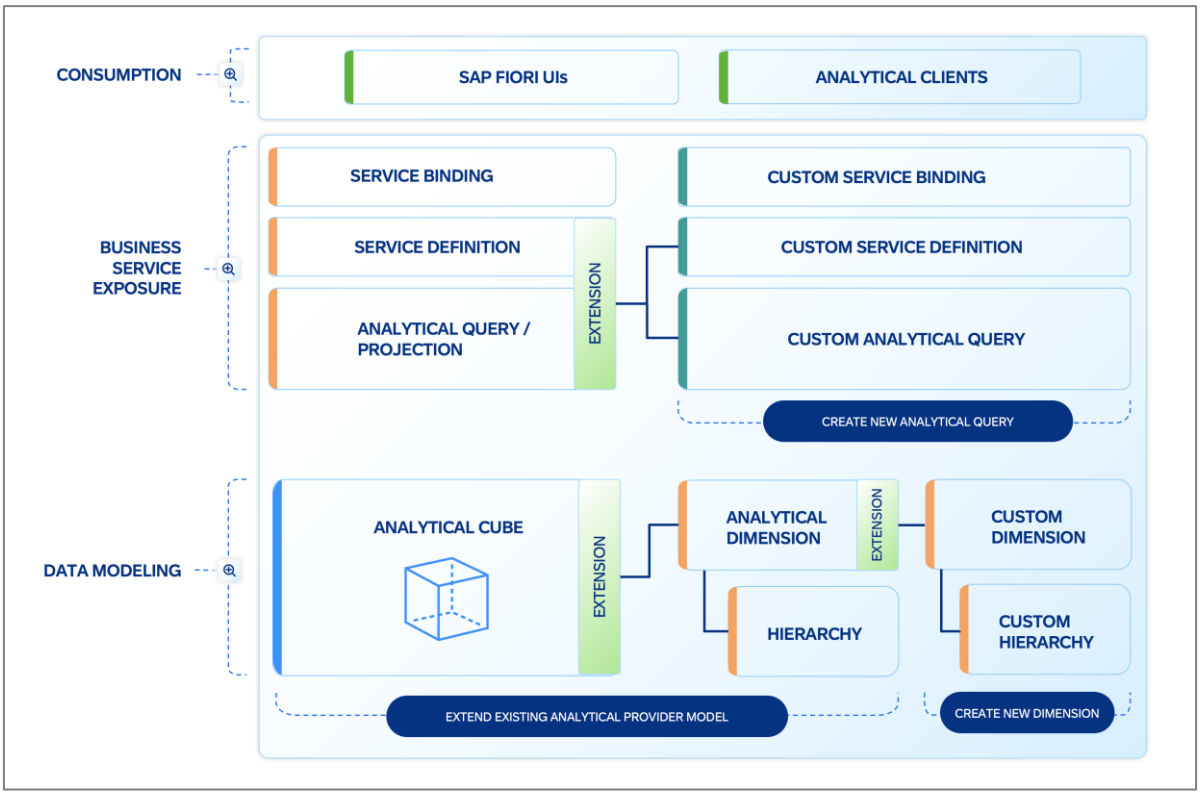


Figure 4.4 - Overview on extensibility options for analytical data models

In an analytical data model, each part of the data model can be extended individually using predefined extension points in the original analytical data model. You can use extension points to make your own analytical data models extensible or to extend pre-delivered SAP content:

EXTENSIBILITY USE CASE	EXTENSIBILITY TASK
DIMENSION EXTENSION Add new hierarchies to a dimension or add new fields or associations to the dimension to diversify the data model.	**Extensibility Enablement:** Add annotations and extension include structures to the original dimension to enable data model extensions like fields or associations.
	Extension Development:

EXTENSIBILITY USE CASE	EXTENSIBILITY TASK
	• Add new hierarchy: To extend a dimension with a new hierarchy, create a new hierarchy and add it as an association to the corresponding dimension. • Add new field extension: Extend the original dimension with additional fields.
CUBE EXTENSION Add new dimensions to a cube to extend the scope of the data analysis or add new measures to the cube to calculate additional values.	**Extensibility Enablement:** Add annotations and extension include structures to the original data model to enable data model extensions like fields or additional dimensions.
	Extension Development: • Add new dimension: To extend a cube with a new extension, create a new dimension or use a pre-delivered SAP dimension and add it as an association to the corresponding dimension. • Add new field(measure): Extend the original cube with additional measures for new calculations.
QUERY EXTENSION Add new numeric fields to the query to extend the scope of data analysis.	**Extensibility Enablement:** Add annotations and extension include structures to the original data model to enable data model extensions like additional numeric fields for calculations in queries.
	Extension Development: Add new fields to extend the scope of the data analysis.
SERVICE DEFINITION EXTENSION Create a new UI based on a released query or extend the service definition to add new queries to a service definition.	**Extensibility Enablement:** Enable the service definition for extensibility to add additional queries to a service.
	Extension Development: • Create a custom UI based on a released UI to adapt the UI to your business requirements. • Add a new query to a service definition to extend the original service definition with additional queries.

Table 4.2 - Extensibility options for analytical data models

4.3 Cloud-optimized ABAP language

ABAP is a programming language optimized for business applications, both at the small and at the large scale. It is designed to minimize the total costs of development for business applications. As such, the ABAP language has evolved over a long time from procedural and dynamic programming to ABAP objects and is the foundation of the ABAP RESTful application programming model. Thereby, both its capabilities and the number of ABAP keywords increased steadily, allowing for enormous flexibility but leading to complexity at the same time.

Now, for the ABAP Cloud development model, the scope of ABAP language commands has been refined to simplify and standardize ABAP development and enable cloud-ready programming. In the new language version named **ABAP for Cloud development**, only modern ABAP statements and concepts, with a focus on cloud-enabled, object-oriented design and modern programming models are supported. Using non-supported statements results in syntax errors. To ensure the integrity of the cloud extensibility model, direct access to the file system, profile parameters, or ABAP server operations is not permitted.

ABAP Cloud is mandatory in SAP S/4HANA Cloud Public Edition and in SAP BTP ABAP environment and **can** be enabled in SAP S/4HANA Cloud Private Edition and SAP S/4HANA on-premise when desired.

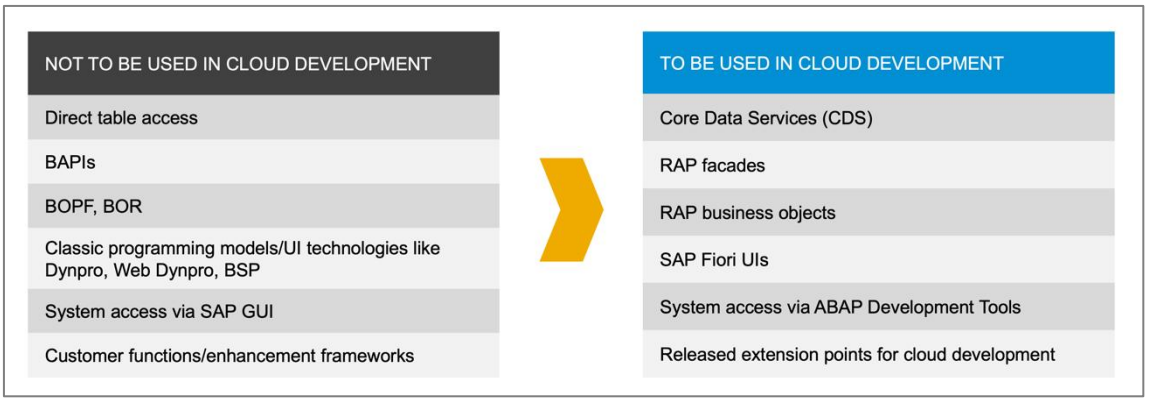


Figure 4.5 - Concepts in Cloud development

More details are provided in chapter 5. An overview of programming concepts is shown below and serves as a (non-comprehensive) list of criteria to decide which concepts can be used or not.

SAP data can only be accessed by using ABAP SQL or ABAP managed database procedures on released SAP CDS views, or by released SAP APIs. For more information on ABAP Cloud scope, see the ABAP Keyword Documentation.

Many technical ABAP Platform features and reuse services are available via publicly released APIs and libraries (→ Chapter 4.4).

4.4 ABAP Platform reuse services

Besides RAP as programming model for cloud-ready development and ABAP Cloud as language version, the ABAP Cloud development model comprises further aspects like usage of modern reuse services, released APIs, and modern business configuration and connectivity frameworks.

4.4.1.1 Released local APIs

The released local APIs constitute the local public interface of the SAP code. This stable interface is the key to the lifecycle independence of SAP code and customer and partner extensions. Hence, only released interfaces, released APIs, frameworks and services can be used in extensions.

The scope of released local ABAP Platform APIs is continuously enhanced and already covers major building blocks:

- Data access APIs (CDS views).
- Technical ABAP Services (Parallel Processing, Compression/Decompression, Runtime Info, Dynamic Programming, XSLT, ...).
- Technical Reuse Services (→ Chapter 4.4.2).

- APIs required to develop integration scenarios (OData, HTTP, RFC, SOAP, Business Events → Chapter 4.4.4).

Important SAP APIs and objects that are not suited for cloud development have been replaced by new and modern APIs that fit into the cloud extensibility model. For these objects, having a direct successor released for cloud development, the syntax error messages directly point towards the successor to be used. An example is given below in Figure 4.4. A direct select on the database table `T005` is forbidden, the successor CDS view `I_COUNTRY`, is directly indicated in the syntax error.

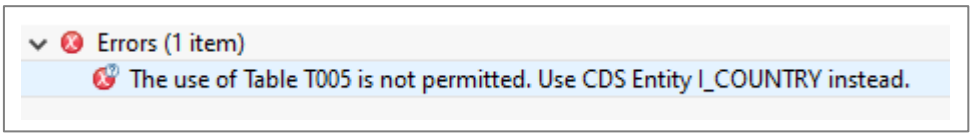


Figure 4.4 - Syntax errors for direct table access

All released APIs are shown in the *released objects tree* in ABAP development tools. If released APIs are missing, please make use of the Customer Influence Channels for the ABAP Platform (https://influence.sap.com/sap/ino/#campaign/2911).

This way you can shape and sharpen the publicly released APIs available for your extensions in the cloud extensibility model.

4.4.2 Technical reuse services

Technical reuse services are essential to lower the TCD and TCO of business applications. Hence, dedicated released APIs and configuration possibilities for well-known ABAP Platform reuse services like Change Documents, Number Ranges, Email Processing, Factory Calendar, and others are provided. In general, to adhere to the principles for cloud development explained above:

- Reuse services are consumed by calling the corresponding released local APIs.
- Necessary development objects (like a number range object) are developed in ABAP development tools.
- Required configuration and monitoring is done via SAP Fiori Apps.

Just as with the scope of the ABAP language, cloud-optimized services used by SAP S/4HANA in the area of reuse services are enabled for consumption in the context of developer extensibility. These services replace partly older services and frameworks or are available in cloud-optimized version. Popular examples of technical reuse services that shall be used in developer extensibility are provided in the table below. Note that this table is only a snapshot providing hints on which services to focus, but is neither a complete list of all services, nor does it state that all services listed there are already available in the full functional scope. Instead, the services listed to be used will be enabled and integrated into the programming model step-by-step.

TECHNICAL REUSE SERVICES USED IN CLOUD DEVELOPMENT		
SERVICE	**STATUS**	**RETIRED PREDECESSOR**
Email Service	●	SAP Office
Email Templates	○	
Factory Calendar	●	
Notes for Application Objects	○	SAPscript Longtexts, STXL
Knowledge Transfer Documents	●	SAPscript, SE61, DOKTL
Forms Processing integrating SAP BTP Forms by Adobe	●	SAPscript, Smartforms
Adobe Form Templates	●	SAPscript, Smartforms

TECHNICAL REUSE SERVICES USED IN CLOUD DEVELOPMENT		
SERVICE	**STATUS**	**RETIRED PREDECESSOR**
Number Ranges	●	
Change Documents	●	
Archive Development Kit	●	
Information Lifecycle Management (Data Destruction)	●	
Printing Queue	●	Spool
Application Jobs	●	Classical Batch Job SM36
Application Logs	●	
XCO ABAP Repository	●	Legacy Workbench APIs, e.g., for DDIC
XCO Standard Library	●	
Translation	●	
Units of Measurement	●	
SAP BTP Document Management Service	●	KPro, Content Server, SAP Office
SAP BTP Workflow	●	
SAP BTP Rules	●	
Time zones	●	
Exchange Rates and Currency Conversion	●	
Attachment Service	○	Generic Object Services, Business Document Service, Archive Link, SAP Office
SAP S/4HANA Output Control	○	Post Processing Framework
Business Even Loging	●	
Legend: ● available ○ planned		

Table 4.3 - Technical reuse services for cloud development

4.4.3 Identity and Access Management

In cloud products, a strict content separation between users and roles is needed to ensure stable cloud operations. Thus, customers do not have direct access to the classical transaction for role maintenance (PFCG). Instead, SAP delivers a new cloud Identity and Access Management (IAM), which consists of:

- IAM SAP Fiori apps for user and role maintenance on top of IAM catalogs and role templates (maintain business roles, ...) and APIs for user upload e.g., from SAP Identity Provisioning Service or other SCIM providers.
- The IAM app, IAM business catalog, and IAM business role templates, for creating IAM content for their own applications and for SAP applications (with released authorization objects).

- IAM catalogs and role templates for cloud-enabled applications delivered by SAP application development.
- Key user extensibility: with a custom catalog extension, customers can extend an existing SAP IAM catalog (e.g., add authorization allowing the user to start a custom OData service) in an SAP Fiori app.

Business users can authenticate via the SAP cloud identity services. The Identity Authentication Service can be set up in proxy mode to use a corporate identity provider in hybrid landscapes.

More information can be found in this SAP community blog: Integrating Identity Authentication service in SAP Cloud Platform – Proxy & Conditional Authentication scenarios.

4.4.4 Connectivity

Connectivity capabilities of the ABAP Platform are key to enable integration between SAP cloud products, SAP BTP services, customer extensions and external services. In hybrid landscapes, the SAP Cloud Connector can be used to integrate with protected internal customer landscapes by providing detailed control over a secure tunnel connection.

The integration with external APIs is simplified through the generation of typed proxies in the system from uploaded metadata via the service consumption model (OData, SOAP, RFC) and event consumption model.

Several classic integration technologies can no longer be used (e.g. IDoc), only released frameworks are available for cloud development. A list of the current scope is given below.

INTEGRATION FRAMEWORKS RELEASED FOR CLOUD DEVELOPMENT
OData services
Business Events
HTTP services
RFC via cloud enabled WebSocket RFC
SOAP consumption (SOAP service provider planned)
SQL service for external ODBC clients
Information access (InA) for analytical clients

Table 4.4 - Integration frameworks for cloud development

To ensure content separation between credentials, customers do not have direct access to classical transactions for destinations (SM59) and logical ports (SOAMANAGER) in SAP S/4HANA Cloud.

Instead, SAP delivers a new Cloud Communication Management, which contains the following:

- The ABAP Platform provides SAP Fiori apps for both inbound and outbound communication (Maintain communication arrangements, maintain communication user ...).
- Custom development: using the development objects communication scenario, inbound & outbound services, customers can create integration content for their own applications and for SAP applications (with released services).
- SAP application development delivers communication scenarios for supported integrations.
- Key user extensibility: key users can create their own communication scenarios for OData services generated from released CDS views and own applications in an SAP Fiori app.

4.5 SAP S/4HANA business APIs, extension points and events

In addition to the ABAP Platform content, in SAP S/4HANA deployments publicly released business APIs and extension points are available in the ABAP Cloud development model (→ Table 4.5).

REMOTE CONSUMPTION	LOCAL CONSUMPTION	LOCAL EXTENSION POINTS
• OData services • SOAP services • Events	• CDS Views • RAP BO interfaces • Classes • Events	• Business Add-Ins (BAdIs) • RAP BO extensions

Table 4.5 - Available business APIs and extension points in SAP S/4HANA

Information on the released SAP S/4HANA APIs can be found in the product documentation and on the SAP Business Accelerator Hub. If an API is missing, please request it in the Customer Influence Chanel for SAP S/4HANA (SAP S/4HANA Cloud Private Edition: https://influence.sap.com/sap/ino/#/campaign/3516, SAP S/4HANA Cloud Public Edition https://influence.sap.com/sap/ino/#/campaign/2759)

4.6 Business Configuration (BC)

The ABAP Platform provides two programming models for business configuration (BC) apps:

- SAP GUI-based BC apps (SM30 – Table View Maintenance, SE54 - Create Table Maintenance View).
- SAP Fiori/RAP-based BC apps.

Some aspects of business configuration apps are different from other apps, and the programming models reflect these differences. BC maintenance apps are used less often, and they are usually very simple compared to the apps for master or transactional data. On the other hand, applications often have a greater number of BC maintenance apps compared to the apps for master and transactional data. This calls for highly standardized apps. Therefore, the programming models support standardized:

- Maintenance UIs.
- Authorization management.
- Change log.
- Maintenance of language-dependent texts.
- Transport recording and BC content management.

SAP GUI/Web GUI is not available in SAP BTP ABAP environment and in SAP S/4HANA Cloud Public Edition. Therefore, customers and partners can only implement SAP Fiori/RAP-based BC apps. To support development, SAP provides a wizard to generate the required RAP objects and a generic SAP Fiori UI (*Custom Business Configurations* SAP Fiori app). With these tools, customers can create simple maintenance apps with low development effort. Details are described in the following blog post: Create a Business Configuration Maintenance App

Customers and partners using SAP S/4HANA on-premise or SAP S/4HANA Cloud Private Edition can decide per scenario whether they go for SAP GUI or SAP Fiori-based BC apps.

As of version SAP S/4HANA release 2022, customers can use the generator for business configuration objects. The *Custom Business Configurations* SAP Fiori app is available as of release 2023. For older versions, customers must create SAP Fiori apps using the SAP Business Application Studio and a tile for the SAP Fiori Launchpad per BC app.

	SAP S/4HANA Cloud Private Edition and on-premise	SAP S/4HANA Cloud Public Edition	SAP BTP ABAP environment
SAP GUI-based custom BC apps (SM30/SE54) (including IMG integration)	✓	×	×
SAP Fiori/RAP-based custom BC apps (including generator in ADT)	✓ (2022)	✓	✓
Custom business configurations SAP Fiori app (generic UI, no dedicated SAP Fiori app required)	✓ (2023)	✓	✓
Integration SAP Fiori/RAP-based custom BC apps into the SAP implementation guide (IMG)	✓ (2023)	✓	IMG is not available

Table 4.6 - Availability matrix for Business Configuration apps

Customers and partners using SAP S/4HANA on-premise or SAP S/4HANA Cloud Private Edition should define a roadmap towards SAP Fiori/RAP for BC apps along the following considerations:

- SAP Fiori/RAP-based BC apps should be used:
  - For new greenfield projects and development teams that have no experience in SM54/SM30.
  - For BC apps with advanced UI requirements and rich custom logic.
- Maintenance views based on SM54/SM30 can be still used:
  - If the maintenance view is simple, has no advanced UI requirements and custom logic.
  - If customers have already a lot of maintenance views based on SE54/SM30, and the development team has experience in SE54/SM30.
- As of today, SAP Fiori/RAP-based BC apps do not support the following features. If these features are required, then SE54/SM30 must be used:
  - Solution Manager integration for customizing synchronization.
  - Cross client comparison and adjustment (SCU0, SCMP, SPRO/SIMG).
  - Content delivery via BC sets for custom configuration tables.
  - BC API

Customers can transform their maintenance views to SAP Fiori/RAP using the generator for business configuration objects. Configuration tables must fulfil the prerequisites of the generator.[17]

Maintenance views based on SM54/SM30 are created in classic ABAP. To get access to the data in ABAP Cloud development, customers must create a CDS view on top of the configuration table and release it for ABAP for Cloud Development.

[17] https://help.sap.com/docs/btp/sap-business-technology-platform/business-configuration-maintenance-object (prerequisite paragraph), and https://help.sap.com/docs/btp/sap-business-technology-platform/generating-business-configuration-maintenance-object-with-generate-abap-repository-objects-wizard (prerequisites paragraph)

5 EXTENDING A NEW SAP S/4HANA CLOUD PRIVATE EDITION OR SAP S/4HANA ON-PREMISE SYSTEM

Until today, in SAP S/4HANA Cloud Private Edition and SAP S/4HANA on-premise deployments, classic ABAP custom code is generally used for extensions. This endangers smooth upgrades and makes further cloud transformation steps more difficult. Hence, as explained in chapter 1 and chapter 2, reusing the SAP S/4HANA Cloud extensibility model also in private cloud or on-premise deployments is beneficial and recommended.

In this chapter we provide guidance on how to achieve this goal. For simplicity, we will only refer to the SAP S/4HANA Cloud Private Edition, keeping in mind that the statements also hold true for on-premise. In the private cloud we need compromises with respect to the pure ABAP Cloud extensibility model since the broader functional scope of SAP S/4HANA Cloud Private Edition is not covered by released SAP APIs and is still offered with SAP GUI based user experience which makes ABAP Cloud based SAP Fiori extensions with RAP less suitable.

Therefore, the private cloud must allow extension use cases that are not supported in the public cloud, e.g. using non-released APIs, extending Dynpro-based transactions.

To manage the coexistence of these different extensibility models in private cloud deployments we propose to work with the following technical guidelines which are in line with the recommendation given in the Clean core extensibility whitepaper with the clean core level concept for the classification of customer extensions.

ABAP Cloud development: default choice for all new ABAP-based extensions. The leading programming model is RAP and only released APIs from SAP can be used in custom development objects. The recommendation is to use a software component with ABAP language version ABAP for Cloud Development and create all new objects within this software component. In this case the usage of released APIs is enforced by the syntax check and at runtime, for dynamic usages of APIs. In case a released API from SAP is missing custom wrappers for non-released SAP objects can be built and released for ABAP Cloud development (released custom APIs). Once SAP provides a released local API, the custom wrapper can be removed. Another important mechanism to control the development of extensions is the use of proper authorizations. With the new authorization object `S_ABPLNGVS` administrators can control if a developer can create objects belonging to the ABAP Cloud development model only, or also objects based on classic ABAP. In this way it is possible to create a dedicated developer role to enforce ABAP Cloud development. The role template SAP_BC_ABAP_DEVELOPER_5 may be used as a starting point. According to the Clean core extensibility whitepaper ABAP Cloud development belongs to Level A.

Classic ABAP Development: classic extensibility based on classic ABAP technologies including frameworks that are not supported in the ABAP Cloud development model. Classic ABAP development is necessary for corrections or development in existing custom code that cannot be transformed to ABAP Cloud or for new extensions in application areas where no released APIs are available and the UI is built with classic UI technology, like SAP GUI. The recommendation for classic ABAP development is using the software component HOME. Following the adapted clean core extensibility rules SAP provides guidance and governance for classic ABAP development. This includes that SAP is offering a recommended list of APIs (classic APIs) to be used in classic ABAP development. The concept of classic APIs will be explained in more details in the chapter 5.1.1. The proper usage of classic APIs is governed by the ABAP test cockpit (ATC). Setup of ATC is provided in chapter 5.1.2.

The following picture provides an overview and interplay of the two development models:

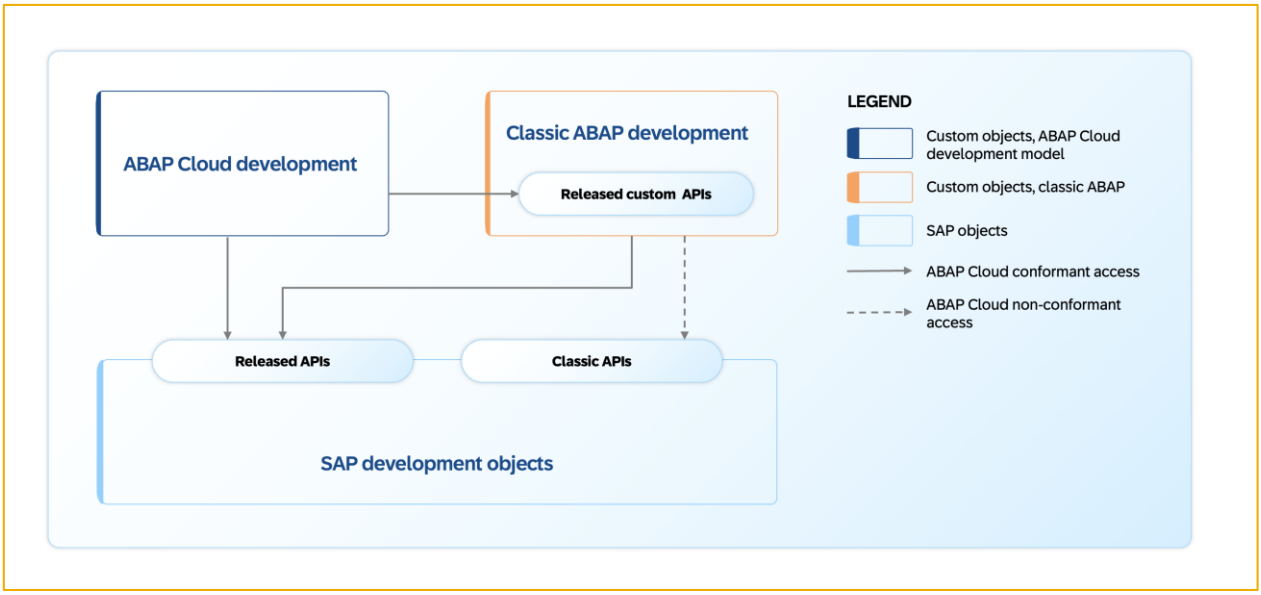


Figure 5.1 – Interplay between ABAP Cloud development and classic ABAP development

Custom objects developed using the ABAP Cloud development model follow strictly ABAP Cloud rules. They can only access the following objects:

- Custom development objects inside the same software component.
- Released SAP development objects provided by the public SAP interfaces (released APIs).
- Custom development objects released for ABAP Cloud development (released custom APIs) of a software component following the classic ABAP development model.

Custom objects developed in software components following the classic ABAP development model should access only the following objects:

- Custom development objects inside their own software component.
- Released SAP development objects provided by the public SAP interfaces (released APIs) and SAP development objects provided by the classic SAP interfaces (classic APIs).
- Custom development objects in software components based on classic ABAP and ABAP Cloud.

5.1 Setting up the governance for the classic ABAP development model

5.1.1 Classic APIs

SAP nominates objects (classic APIs) which shall be consumed in classic ABAP developments and provides the release independent list of classic APIs through the object classification file on GitHub: GitHub - SAP/abap-atc-cr-cv-s4hc.

SAP Note 3578329 lists the commonly used classic technologies, reuse services and application specific frameworks, which are considered as clean core compliant. SAP recommends classic APIs for these components which shall be used in classic ABAP developments.

Classic APIs are offered in the JSON file *objectClassifications_SAP.json*, which is designed to be used by ABAP test cockpit to govern the usage of classic APIs in custom developments with classic ABAP. Classic APIs are provided for the following main functional SAP object types:

- Function modules (FUNC)
- Classes (CLAS)
- Interfaces (INTF)
- CDS views (STOB)

- BO interfaces (BDEF)

Object classifications for SAP DDIC objects like data elements, structures and table types are not provided, and DDIC objects are therefore also not checked by ATC checks in case they are used as data types. SAP DDIC objects can be seen as central reusable data types for classic ABAP development without any restriction. Only the SQL access to SAP tables is addressed by ATC (→ chapter 5.1.2 for details).

Classic APIs can be inspected by the Cloudification Repository Viewer on GitHub: https://sap.github.io/abap-atc-cr-cv-s4hc/?version=objectClassifications_SAP.json

The classic APIs offer access to application specific services like the well-known BAPIs. In addition, classic APIs offer access to the above-mentioned classic technologies and reuse services, e.g., central reuse classes for SAP GUI programming like the ALV Grid Interfaces.

There is a second type of API classification in the JSON file – the so called *not recommended APIs*. These APIs are explicitly inspected by SAP experts and assessed as not suitable to be used in custom code.

According to the Clean core extensibility whitepaper the usage of classic APIs belongs to Level B objects whereas the usage of not recommended APIs belongs to Level D.

Here is a summary of the available API classifications:

API classification	Clean core level	Use of SAP objects and technologies	Technical attribute in JSON file	ATC result	Example
Classic APIs	**B**	These are the recommended APIs that should be used in addition to the released APIs in classic ABAP extensions. There are a few classic APIs that also provide a successor, e.g. a released API.	Classic API	Info message	CL_GUI_ALV_GRID

Not recommended objects	**D**	These are objects that customers and partners should not use in their extensions. These objects are explicitly classified by SAP experts as not recommended. Usage of not recommended objects should be replaced via released APIs or classic APIs There are a few APIs classified including a classic API as successor.	No API	Error	RFC_READ_TABLE

SAP development objects that are not publicly released and not listed in the JSON file are per default categorized as internal objects. The usage of these objects refers according to the Clean core extensibility whitepaper to Level C objects.

There is an additional classification available in the JSON file for classic APIs called *transactional-consistent*. This classification defines that the API can be used in the context of a RAP-based applications. Further information is provided in the SAP Help Documentation about RAP Transactional Model.

5.1.2 ABAP Test Cockpit setup

The ABAP test cockpit (ATC) is the tool of choice to govern the static quality assurance for ABAP Cloud development and classic ABAP development compliant to the clean core level concept. The recommendation is to setup a central ATC system for your whole landscape. A central ATC system can be used as a public cloud solution on SAP BTP - which is SAP's recommended option - or it can be installed in your private cloud or on-premise landscape.

It is recommended to setup one global check variant in the ATC configuration that is used during development and when releasing transport tasks. Furthermore, the ATC transport settings shall be configured so that priority 1 and 2 findings block the release of transport tasks and requests.

The following ATC checks should be part of your global ATC check variant:

Allowed enhancement technologies	Clean core	SYCM_ALLOWED_ENH_TECHNOLOGY	Reports enhancements (object type ENHO) based on not allowed enhancement technology. Only the BAdI enhancement technology is recommended to use
Usage of APIs	Clean core	SYCM_USAGE_OF_APIS	Reports the usage of classic ABAP development objects

			based on the utilization of SAP released APIs and classic APIs. Reports the usage of SQL statements on SAP database tables and views Reports the usage of form routines of ABAP programs from SAP
Search customer modifications	Clean core	CI_SEARCH_CUST_MODIFICATIONS	Finds modified objects and emits a finding for each modified object (not for each modification). Checks if the object types are supported in ABAP Cloud
Critical statements	Clean core	CI_CRITICAL_STATEMENTS	Checks whether certain statements that are critical for security or that threaten program stability are used, e.g. direct kernel calls
Code Vulnerability Analyzer (CVA)	Additionally recommended	SLIN_SEC	Checks for security vulnerabilities

Table 5.1 - ATC clean core checks and additionally recommended checks to add to the ATC global check variant

Remark: If the checks *Usage of APIs* and *Allowed Enhancement Technologies* are not available in the system then SAP Note 3565942 should be implemented.

The check for critical statements considers:

- Calling a kernel function
- Using a system-call
- Using an editor call
- EXEC SQL calls
- Using a database hint
- Generation of reports and Dynpros
- Read/insert report
- Read/insert Dynpro
- Import/export nametab

If you setup ATC the first time for your landscape, you can use the ATC variant ABAP_CLOUD_DEVELOPMENT_DEFAULT as template and enrich it with the checks described above. SAP

will deliver a predefined ATC variant ABAP_CLEAN_CORE_DEVELOPMENT, which can be used for the initial ATC setup.

The table below provides you an overview of the ATC findings that result from the clean core checks according to the clean core level concept:

ATC check priority	Clean core level	ATC finding	Recommended action
Priority 1 (Error)	**D**	Object is modified	Remove modification and use a BADIs instead if needed
		Enhancement technology is not allowed Implementation of implicit or explicit enhancement (source code plugin)	Delete enhancement implementation and use a BADI instead if needed
		Direct write access to SAP database tables	Use released or classic APIs instead
		Call of form routines in SAP program	Use released or classic APIs instead
		Usage of SAP object classified as *No-API*	Use released or classic APIs instead. Use successor information if available
		Usage of critical ABAP statements	Use released APIs or classic APIs instead
Priority 2 (Warning)	**C**	Direct read access to SAP database table/view	Use released CDS view or classic API instead
		Usage of SAP object that is not classified as classic API (SAP internal object) and is not released	Use released or classic APIs instead
Priority 3 (Info)	B	Usage of deprecated APIs	Use successor instead
		Usage of classic APIs	-

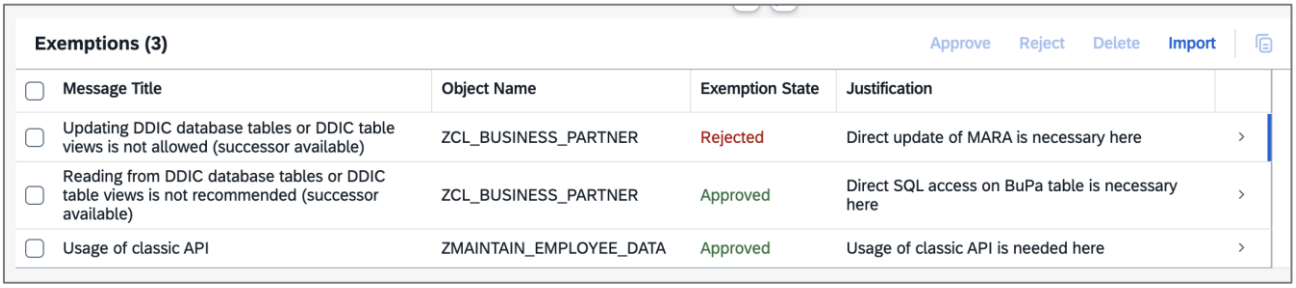
5.1.3 Clean core governance by using ATC exemptions

Even if the object classification already provides a wide range of classic APIs that can be used in classic ABAP development there might be reasons to still use SAP internal objects (not classified as classic API) especially in legacy custom code.

In these cases, we recommend using ATC exemptions to control their usage. The focus should be on findings with priority 1 (Error) and priority 2 (Warning). You should not create exemptions for findings with priority 3 (Info), such as those related to the use of classic APIs. There are various options for creating exemptions for ATC findings. Exemptions can be applied at the level of entire packages, whole ABAP objects/sub-objects, or more granularly on individual findings. For clean core findings, you should use the fine-granular option at the finding level to ensure the best possible governance. Only in the case of old legacy code that you do not want to change for clean core compliance, an ATC baseline would be a good option to hide all clean core–related findings for a larger set of objects.

If you use an SAP internal API frequently in your custom code, or if you often access a single database table directly, you will receive an ATC finding for each usage. This can make the process of creating exemptions for every individual finding quite tedious. The recommended practice is to create a wrapper around the SAP object and use this wrapper in your custom code, instead of directly accessing the SAP object. With this approach, there is only a single usage of the SAP object within the wrapper, and you only need to create one exemption for that finding. This is considered a best practice for working with internal SAP objects.

The ATC exemption browser can be used to monitor exemptions and is therefore the recommended tool for controlling the adoption of the clean core extensibility model.



Exemptions (3) Approve Reject Delete Import

Message Title	Object Name	Exemption State	Justification
Updating DDIC database tables or DDIC table views is not allowed (successor available)	ZCL_BUSINESS_PARTNER	Rejected	Direct update of MARA is necessary here
Reading from DDIC database tables or DDIC table views is not recommended (successor available)	ZCL_BUSINESS_PARTNER	Approved	Direct SQL access on BuPa table is necessary here
Usage of classic API	ZMAINTAIN_EMPLOYEE_DATA	Approved	Usage of classic API is needed here

Figure 5.2 –Overview of ATC exemptions for clean core governance

5.1.4 Assessing upgrade stability for usage of internal SAP objects in custom code

Custom code using internal SAP objects is not upgrade-stable per definition. Internal SAP objects can change at any point in time during an upgrade, a feature pack import or even during the import of an SAP note. Incompatible changes are for example:

- Deletion of objects (e.g., function modules)
- Renaming or deletion of function module parameters
- Incompatible changes to parameter types
- Renaming or deletion of methods in SAP classes
- Removal of fields from CDS views

To better estimate the risk of custom code not functioning properly after the next release upgrade, SAP plans to offer a changelog for SAP objects. This changelog contains the names of all objects that have undergone incompatible changes in the new release.

The changelog covers the following object types:

- Function modules (FUNC)
- Classes (CLAS)
- Interfaces (INTF)
- CDS views (STOB)
- Behavior definitions (BDEF)

Technically, the changelog information is transferred via the Simplification Database infrastructure from the SAP Support Portal to your system. You download the Simplification Database file containing the changelog data from the SAP Software Download Center of the SAP Support Portal and import it via transaction SYCM into your system.

There are plans for a new ATC check that analyzes the custom code against the changelog to detect the use of objects that have been changed incompatibly. You can use this ATC check to identify usages of incompatibly changed SAP internal objects during development, or execute this ATC check before the system upgrade, like the process for other simplification items ATC checks.

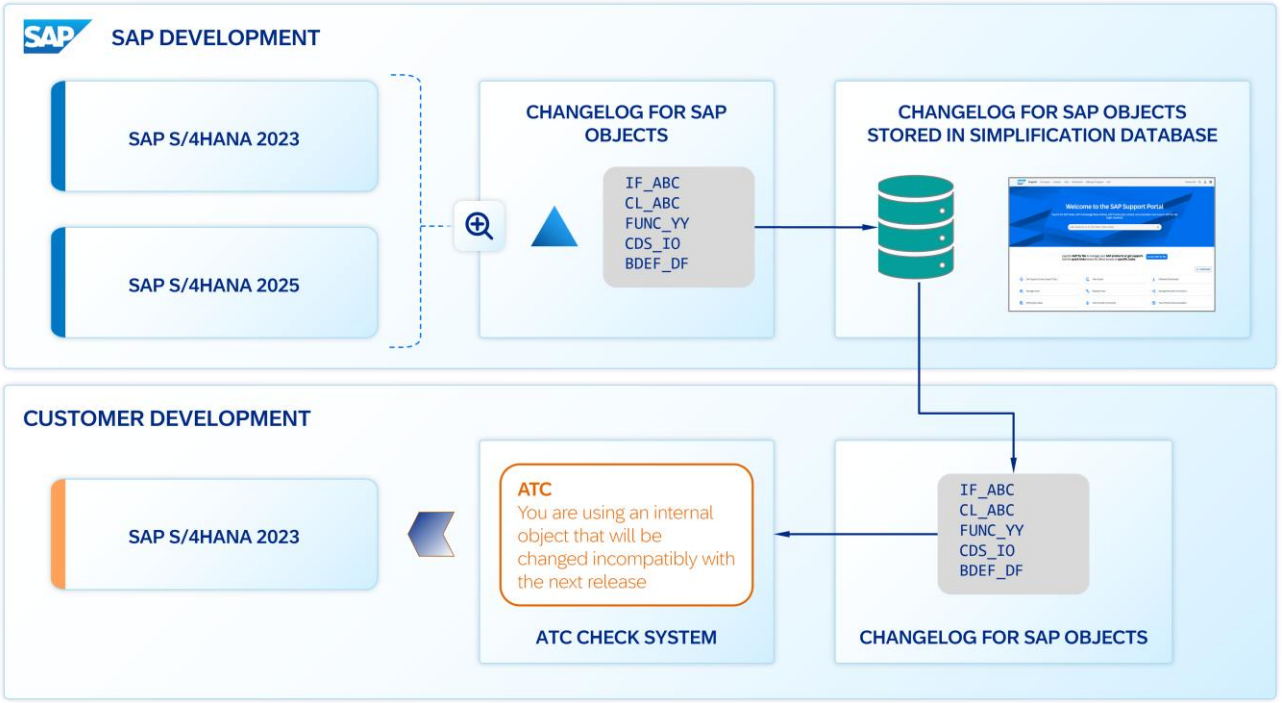


Figure 5.3 – Utilizing changelog for SAP objects to assess upgrade stability of custom code

5.1.5 Setup summary

An overview of setup, tools, and rules in the two development models is given in the table below.

	TOOLS / RULES FOR ON-STACK CUSTOM ABAP CODE	STRUCTURING
ABAP Cloud development **Intended for the cloud ready development of new applications and extensions**	The ABAP Cloud development model is used • Development objects are created in language version *ABAP for Cloud Development.* • ABAP Cloud rules are enforced by syntax and runtime checks. • ABAP development tools for Eclipse is the mandatory IDE. • Use developer role template SAP_BC_ABAP_DEVELOPER_5.	Software component with language version *ABAP for Cloud Development*.

Classic ABAP development **Usage of legacy technologies**	• Development objects are created in language version *Standard ABAP*. The development rules are controlled via the ABAP test cockpit, e.g. the usage of classic APIs	Packages in HOME or existing software component for classic ABAP

Table 5.2 – Setup summary

5.1.6 Transformation towards ABAP Cloud

The ABAP language version for individual development objects in a software component for classic ABAP can be changed to *ABAP for Cloud Development* if required (→ Figure 5.4).

This option can be used to transform code step-by-step to prepare it for a move to a software component based on ABAP Cloud[18]. The ABAP test cockpit check variant `ABAP_CLOUD_READINESS` can be used to check beforehand whether the rules of the ABAP Cloud development model are respected during this transformation[19].

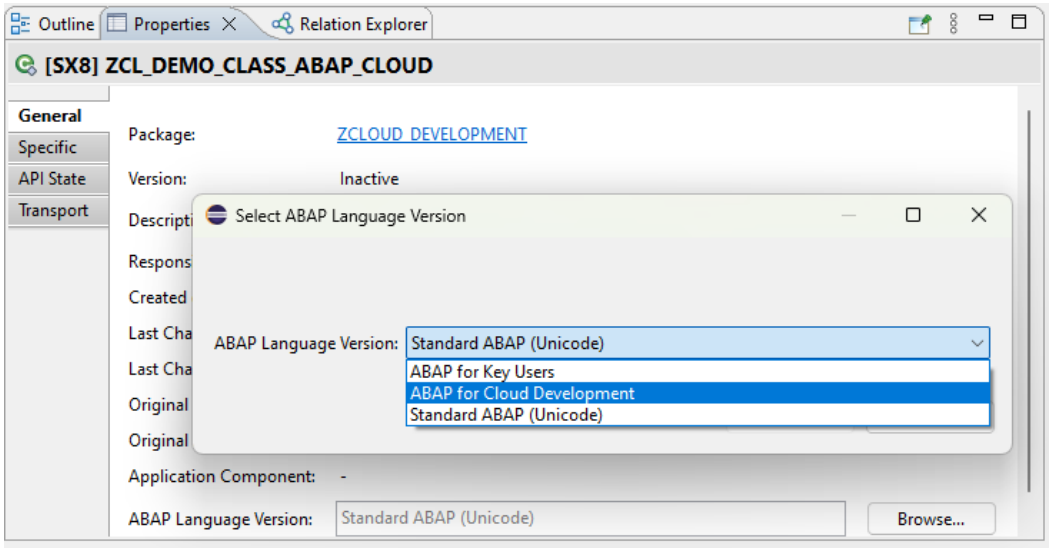


Figure 5.4 - Setting the ABAP Language Version for individual objects in ADT.

5.1.7 Custom code use cases and their related clean core level

To make the clean core extensibility model more tangible, here are some recommendations for typical custom code use cases and their mapping to the clean core levels:

[18] Note that the individual change of the ABAP language version is possible for ABAP code objects (e.g., classes, interfaces) as of SAP S/4HANA release 2019, while the change of the language version for DDIC, CDS and other objects is possible as of SAP S/4HANA release 2022

[19] For more information, see also: How to check your custom ABAP code for SAP BTP ABAP environment

CATEGORY	USE CASE	RELATED CLEAN CORE LEVEL	RECOMMENDATION
EXTENSIONS	Custom field on DB table or CDS view via released extension include	A	
	Implementation of a released SAP BAdI	A	
	Custom field on DB table or CDS view via extension include that has not been released	B	
	Custom field on DB table via classic append	C	Monitor when extension include will be available and adapt accordingly
	Implementation of an SAP BAdI that has not been released	B	Monitor when BAdI will be released and adapt accordingly
	Implementation of a BAdI that has been flagged as SAP-internal	D	Should be allowed in exceptional cases only
	Extension of an SAP Fiori app (RAP based)	A	
	Extension of an SAP Fiori app (SEGW, BOPF, UI5)	B	
	Extension of an SAP application with legacy UI technology, e.g., SAP GUI transaction	B	Monitor if SAP Fiori application will be available
CUSTOM APPLICATIONS	Custom SAP Fiori app (RAP based) for the core SAP S/4HANA Cloud scope	A	
	Custom SAP Fiori app (SEGW, BOPF, UI5)	B	Prefer to use RAP. Exception could be know-how or reuse
	Custom application with legacy UI technology (e.g., ABAP report with ALV, Web Dynpro application)	B	Should be prevented
	SE54 based BC UI	B	RAP-based BC-App
WRAPPER	Wrapper classes around SAP objects that have not been released (e.g., BAPI)	B if the wrapped object is a classic API, otherwise C	Monitor when the released RAP interface is available and replace the wrapper
	Wrapper CDS view for SAP table or CDS view that have not been released	B if the wrapped object is a classic API, otherwise C	Monitor when a released CDS view is available and replace the wrapper
MODIFICATIONS	Applying SAP note with manual corrections, e.g., DDIC object from SAP BASIS	D	Reset modification once the correction is part of the core
	Implementation of user/customer exits (e.g., SAPMV45A)	D	Monitor if SAP released BAdI can be used instead
	Modification (Business driven)	D	Should be allowed only in exceptional cases

Table 5.3 - Custom code use cases and their mapping

A major difference between SAP S/4HANA Cloud Public Edition on one hand and SAP S/4HANA Cloud Private Edition on the other is the Identity and Access Management (IAM) and Communication Management (COM). In SAP S/4HANA Cloud Private Edition, there is no strict content separation between SAP and

customers regarding users, roles, and communications. The administrative SAP Fiori apps for IAM and COM are not available. Instead, the classic transactions must be used (PFCG, SM59, SOAMANAGER, ...).

5.2 Custom wrappers: When and how to use classic ABAP code to mitigate missing APIs

As explained above, the target for any new extension is the ABAP Cloud development model. If a suitable publicly released API is missing, custom wrapper objects around the non-publicly released SAP APIs will be created. The wrapper objects are developed based on classic ABAP and need to be released for cloud development as shown in Figure 5..

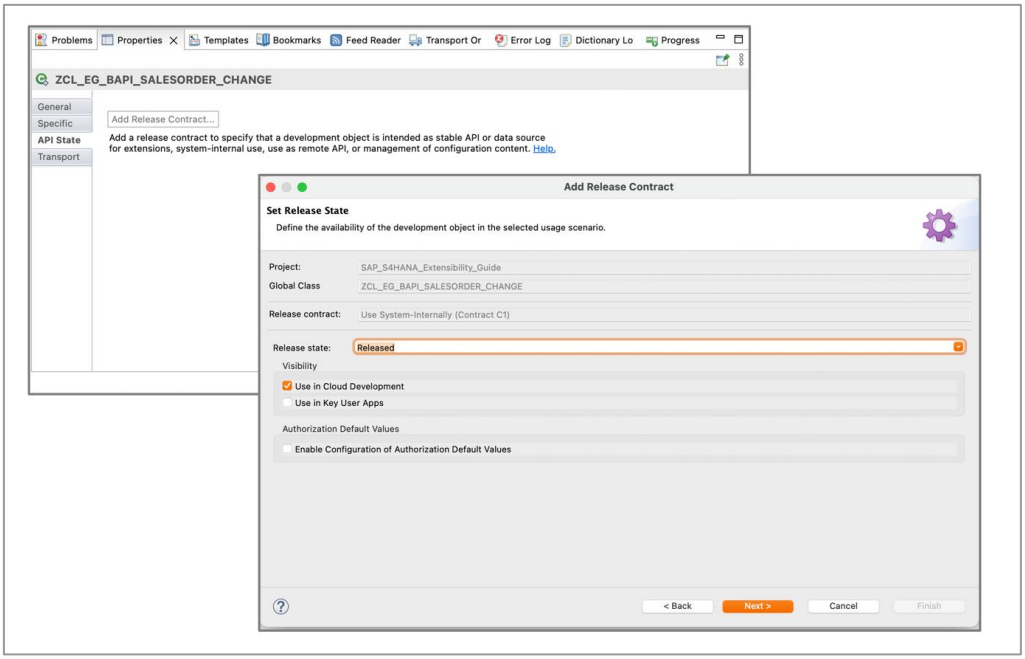


Figure 5.6 - ABAP class released for cloud development.

The rest of the code and other objects created during custom wrapper implementation will follow the ABAP Cloud development model. Once SAP releases the necessary APIs, the non-released objects used in the wrappers can be replaced by publicly released local APIs and the code can be moved from classic ABAP development to ABAP Cloud development.

In the following we provide more guidance as to how to develop such wrappers:

- **CDS views**
  If a suitable SAP CDS view exists that is not yet released, a custom CDS view on top of the SAP CDS view can be created and released for cloud development. CDS views that are referenced in the SAP CDS views (e.g., association targets or value help views) may need a custom wrapper as well.
  If a suitable SAP CDS view does not exist at all, a custom CDS view on top of the database table needs to be created.
- **ABAP classes/interfaces**
  If a suitable SAP function module or class exists that is not released, a custom wrapper class can be created and released for cloud development.
  When creating such a wrapper class, no copies of non-released data types (data elements, structures, table types) used in the signature of the object will be created. Instead, those data types will be wrapped as well, using TYPES declarations in the public section of the wrapper class. In this way, only the wrapper class itself needs to be released for cloud development.
- **Authorizations**
  If a classical SAP API is used, it is likely that non-released authorization objects will be checked. These authorization objects cannot be used with ABAP Cloud. Nevertheless, PFCG roles can be built using both authorization objects that have been released as well as ones that have not been released. To make the non-released authorization objects required for wrappers transparent, it is recommended to provide

SU22 data for the wrapper when releasing it for cloud development and to create SU22 variants for each designated usage of the wrapper. These can be used for role maintenance.

5.3 Classic extensions

Extensions that do not follow the rules of ABAP Cloud development and cannot be developed with ABAP Cloud development model need to be developed with classic ABAP. An extension using ABAP Cloud development model is always to be preferred over an extension using classic ABAP development.

While SAP has nominated objects (classic APIs) which shall be consumed in classic ABAP developments through the object classification file on GitHub, there is no central nomination of objects (classic extension point) which shall be extended in classic ABAP development. SAP may provide such an object classification file in the future.

In the following sections we give guidance on how to reduce the negative impact of such extensions for selected extension scenarios. SAP Note 3578329 provides more details regarding the rating of the classic extensions in the clean core level concept.

5.3.1 Using classic structure extensions

Structure extensions are extensions of the data model of database tables, structures, CDS views, and services (OData, SOAP, BAPI, IDOC, ...).

A general rule for related objects is: no modifications to the corresponding SAP objects.

DDIC

- Append fields to non-released structures and database tables using existing customer includes (CI includes) or extension includes (EEW includes).
- Use domain value appends, search help appends, search help exits.

CDS

- Use CDS extends.
- Use CDS metadata extension.

OData Services

- Use redefinition of OData services.
  More information can be found in: How to redefine RDS based OData services? | SAP Blogs.

5.3.2 Using classical business logic extension techniques

Classic business logic extensions should follow the below recommendations below:

- BAdIs: even if a BAdI is not released for usage in cloud development it is a good choice for customers to extend the application.
- User exits, like VOFM, SAPMV45A: user exits are coding parts pre-defined by SAP (form routines, includes) in an SAP namespace where the customer can implement custom code to extend the SAP business process. The predefined coding parts from SAP are stable and typically do not lead to issues after an upgrade. Technically the custom code here is treated as a modification.
- Customer exits (SMOD/CMOD): customer exits are the predecessor technology to BAdIs and will be replaced. Anyhow you can still use them in exceptional cases where no BAdIs exist yet in the application. Most likely they will be replaced completely with upcoming releases.
- Explicit enhancement spots: these are extension points that were mainly created by SAP to enable industry-specific adaptations to the core ERP applications. We recommend using them only in exceptional cases to include custom-defined BAdIs into the SAP core. You should not directly include extension code in the enhancement spot.
- Implicit enhancement spots: this extension technique is very similar to modifications and should therefore not be used to extend the core applications.

5.3.3 Modifications

Modifications of SAP core objects should be avoided completely. If they cannot be prevented, we strongly recommend using the modification assistant. This makes the adjustment of the modifications after an upgrade manageable in contrast to free-style modifications. Moreover, the ATC check *Search customer modifications* allow one to establish a governance for this purpose.

6 MANAGING AND TRANSFORMING EXISTING CODE IN SAP S/4HANA CLOUD PRIVATE EDITION AND SAP S/4HANA ON-PREMISE

6.1 How to handle existing classic ABAP custom code in parallel with the new cloud ready extensions

In chapter 5 we described how customers should apply the cloud extensibility model in a new implementation project. Now we want to look at SAP S/4HANA projects where a classic conversion is the basis for the transition. In this scenario the customer must handle existing classic ABAP custom code in the converted SAP S/4HANA system.

The goal is to minimize and control the content of classic ABAP custom code. Therefore, we recommend leaving behind as much old legacy code as possible when converting to SAP S/4HANA – cleaning up after the conversion involves much more work.

We do not want to repeat all the recommendations for custom code in a conversion project as this is already described in the whitepaper:

Mapping your journey to SAP S/4HANA Cloud Private Edition : A practical guide for senior IT leadership

In a nutshell the procedure is:

- Analyze the classic ABAP custom code with the Custom Code Migration App to estimate the impact.
- Detect unused classic ABAP custom code based on usage analysis and remove it during the conversion.
- Adapt the classic ABAP custom code by using ABAP test cockpit and automate the adaptation with quick fixes.

We now want to focus on the recommendations for customers as to how to apply the ABAP Cloud development model after a system conversion. The amount of legacy code in the converted system is high and rewriting the complete legacy code with the ABAP Cloud development model is expensive and would be like a new implementation project. Therefore, we recommend the following options for the existing legacy custom code:

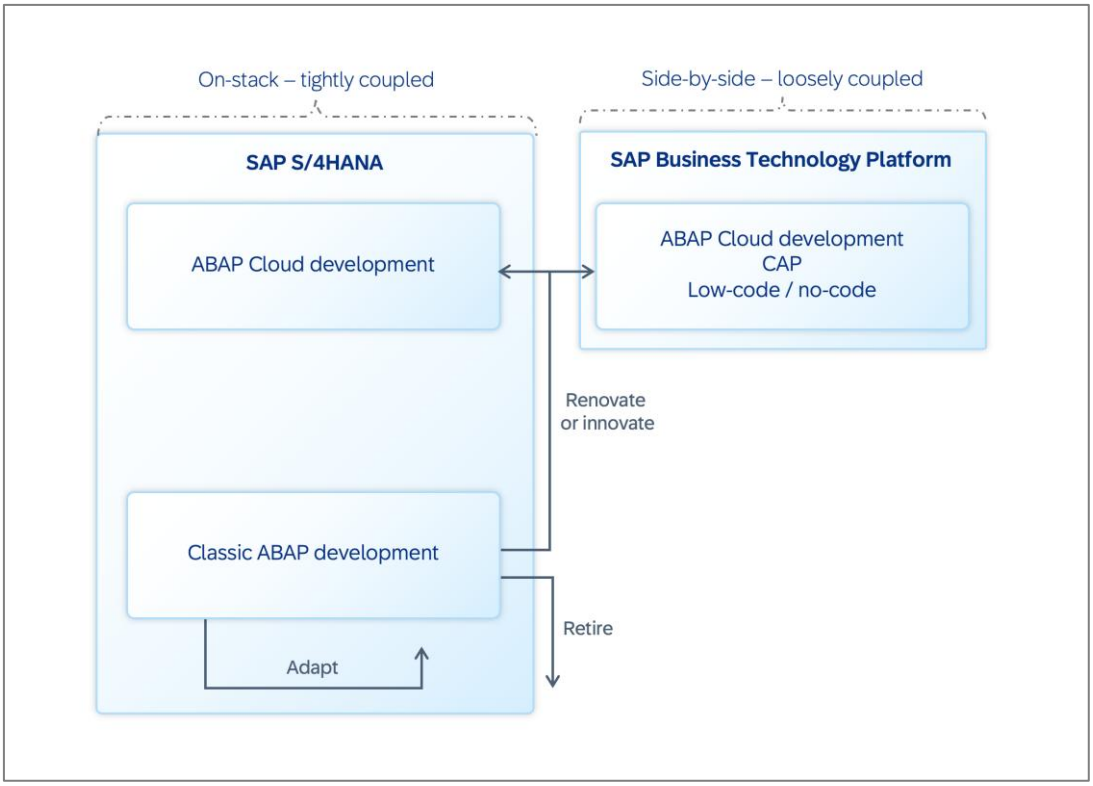


Figure 6.1 - Options for custom code migration

Retire unused code: the analysis of classic ABAP custom code in SAP S/4HANA projects shows that roughly 60% (sometimes even more) of the code is not used productively in a typical ERP system. If you did not use the option of removing classic ABAP custom code during system conversion, we recommend that you use the *custom code usage analysis* in the productive SAP S/4HANA system to detect unused code and retire the code accordingly.

After an SAP S/4HANA transition large parts of the classic ABAP custom code are no longer relevant because the use case of the customer extension is now part of the core application. We recommend removing the customer extension and using the core logic instead.

The same applies for clones of SAP code. Remove the clones and use the cloud extensibility model to meet the business requirements directly in the core application.

Typical tasks for custom code retirement:

- Collect usage data in production with transactions SUSG/SCMON.
- Analyze usage data in SAP's Custom Code Migration app.
- Remove/backup unused classic ABAP custom code with the Custom Code Migration app.
- Use the clone finder.

Renovate & Innovate side-by-side: another portion of the classic ABAP custom code no longer fits the modern approach of cloud ERP. Instead, the SAP Business Technology Platform (SAP BTP) should be used with its broad set of business services and innovative technologies.

Replace the classic ABAP custom code and move the extension use case to SAP BTP. Especially for loosely coupled extensions, this is the right way to keep the core clean[20].

Typical tasks are:

[20] The *clean core* concept is a mindset and philosophy supported with governance and guidelines that lay the foundation for a flexible and future ready ERP. The concept includes different dimensions as extensibility, integration, data, processes, and operations. With the focus on extensibility, the clean core means to keep extensions strictly separate from the SAP application. Extensions access SAP business objects only through well defined, released and upgrade-stable interfaces.

- Analyze dependencies between extensions and the core with the Custom Code Migration App (more information: Custom Code Migration app in SAP community).
- Transfer custom code from SAP S/4HANA to SAP BTP.
- Adapt custom code in SAP BTP via ATC and quick fixes.

Renovate & Innovate on-stack: after a system conversion there is still a certain amount of custom code which does not match the categories above. These are mainly extensions or even legacy modifications of SAP code, which are tightly coupled with the core application logic of SAP S/4HANA. For such code the recommendation is to analyze the business case and decide if a new implementation based on the ABAP Cloud development model is the way to go. In that case follow the guidance of chapter 5.

Typical tasks are:

- Analyze custom code complexity with Custom Code Migration App to detect TCO drivers.
- Make your custom code compliant with the ABAP Cloud development model following the guideline from chapter 4.
- Replace usage of not-released SAP objects with released APIs, e.g., CDS views instead of DB tables.
- To extend or change an SAP object use only released extension points (released BAdIs, released RAP BOs, custom fields, or logic from key user extensibility).

Adapt: if none of the above options for the custom code transformation is feasible you must at least adapt the custom code based on the simplification database. We recommend using ABAP test cockpit and quick fixes in ADT to adapt your custom code. Using quick fixes will significantly reduce your workload for adapting custom code by means of semi-automatic adaptation, and you can expect up to a 60% automation rate for the most frequent findings of the typical SAP S/4HANA simplification use cases. Consequently, we recommend also using SQL Monitor to detect performance optimization candidates which can be optimized with the new ABAP code pushdown capabilities, for example AMDP.

Typical tasks:

- Adapt usage of SAP objects based on SAP S/4HANA readiness ATC checks, the Simplification Database, and the Quick fixes in ADT.
- Usage of SQL Monitor to detect performance optimization candidates.
- Adapt your SQL usage to make it SAP HANA ready, e.g., fix ORDER BY issues.
- Use CDS, AMDP and ABAP SQL to adapt performance-critical SQL queries.

In addition to these must-have adaptation tasks, we recommend to also start the adoption of the clean core levels in your legacy code. For more information about the clean core levels please check Clean core extensibility whitepaper.

- **Try to eliminate all Level D extensions** or developments and adapt them to a higher Level:
    - No modifications. Use BAdIs for custom extensions instead.
    - No implicit or explicit enhancements. Use BAdIs for custom extensions instead.
    - No direct write access to SAP database tables. Use released or classic APIs instead.
    - Do not use APIs that are not recommended. Use released or classic APIs instead.
    - Eliminate code that uses critical ABAP statements. Use released or classic APIs instead.
    - Do not call form routines within SAP programs. Use released or classic APIs instead.
- **Try to eliminate Level C development** as much as possible.
    - No direct read access to SAP table. Use released or classic CDS views instead.
    - Replace usage of SAP internal APIs by released or classic APIs.
    - If SAP internal APIs are still needed explore the changelog for SAP objects for incompatible changes.

Summary: the following overview gives some guidance for the transition of classic ABAP custom code towards the clean core development model:

	USE CASE	GUIDANCE/RECOMMENDATION
EXTENSIONS	Existing custom field (Append) on DB table or CDS view	Migrate to released extension includes (if available)
	Existing implementation of an SAP BAdI	Check if a migration towards a released BAdI is possible and adapt the implementation to ABAP Cloud accordingly
	Existing extension of an SAP Fiori app (SEGW, BOPF, UI5)	Migrate the SAP Fiori app extension (or parts of it) to the newly delivered RAP application if available
	Existing implementation of user exit (e.g., SAPMV45A)	Analyze the program with ATC clean core checks and adopt to released and classic APIs where possible
	Existing modifications	Check if released or classic BAdI is available to replace the modification
	Existing enhancement implementation (implicit, explicit)	Check if released or classic BAdI is available to replace the modification
	Existing Custom SAP Fiori app (SEGW, BOPF, SAP UI5)	Analyze the application with ATC clean core checks and adopt to released and classic APIs where possible
CUSTOM APPLICATIONS	Classic ABAP reports (with or w/o ALV)	Analyze the report with ATC clean core checks and adopt to released and classic APIs where possible
	Dynpro, BSP or Web Dynpro Applications	Analyze the application with ATC clean core checks and adopt to released and classic APIs where possible
	Database access to SAP tables	Replace by using released or classic APIs
	Usage of not recommended or SAP internal APIs	Replace by using released or classic APIs

Table 6.1 - Overview: transition of classic ABAP custom code towards the clean core development model

6.2 How to handle existing custom code in a RAP compliant fashion

It is no surprise that a lot of existing custom ABAP code is based on legacy UI technologies like (Web) Dynpro or Business Server Pages (BSP). Some of the apps based on these technologies can be transformed with minimal effort, e.g., using the generator for repository objects (a.k.a. RAP generator) or by using the wizard for creating SAP Fiori Business Configuration apps. Nevertheless, transformation of most of the apps to SAP Fiori can be very complex. Consequently, many of these applications need to be rewritten to make them future proof. Therefore, you can continue using existing apps as long as no major changes to the applications are required, otherwise it should be checked whether a conversion to the modern technology stack makes sense. See the following blog post that helps you understand how to build modern applications or Web APIs using the RAP programming model.

Modernization with the ABAP RESTful Application Programming Model (RAP)

MORE INFORMATION

KEY USER EXTENSIBILITY

- Custom Extensions in SAP S/4HANA Implementations - A Practical Guide for Senior IT Leadership
- The Key User Extensibility Tools of SAP S/4HANA
- SAP S/4HANA Extensibility: A Learning Journey
- SAP Help Portal - Personalizing and Adapting Apps
- Low-Code/No-Code Development Tools
- SAP Help Portal - Extending an SAP Fiori Application
- Extending SAP-delivered SAP Fiori elements apps using adaptation projects to create SAP S/4HANA app variants

SAP FIORI AND SAP UI5

- SAP Fiori tools
- Developer Adaptation
- ABAP CDS annotations
- Developing Apps with SAP Fiori Elements
- SAP UI5 documentation

ABAP RESTFUL APPLICATION PROGRAMMING MODEL

- RAP Developer Guide on SAP Help Portal
- Modernization with the ABAP RESTful application programming model (RAP) | SAP Blogs
- Modern ABAP Development with the ABAP RESTful Application Programming Model
- Working with ABAP Cross trace tools
- Unit testing with ABAP unit
- Documenting ABAP development objects | SAP Blogs)

CUSTOM CODE ANALYSIS AND TRANSFORMATION

- Central ABAP test cockpit in SAP BTP on SAP Help Portal
- How to check your custom ABAP code for SAP BTP ABAP environment | SAP Blogs
- Custom Extensions in SAP S/4HANA Implementations - A Practical Guide for Senior IT Leadership
- How to redefine RDS based OData services? | SAP Blogs
- Get started with the ABAP custom code migration process | SAP Blogs

ABAP CLOUD

- ABAP Keyword Documentation

CUSTOMER INFLUENCE CHANNELS

- ABAP Platform (https://influence.sap.com/sap/ino/#campaign/2911)
- SAP S/4HANA Cloud Private Edition (https://influence.sap.com/sap/ino/#/campaign/3516)
- SAP S/4HANA Cloud Public Edition (https://influence.sap.com/sap/ino/#/campaign/2759)

BUSINESS CONFIGURATION

- Create a Business Configuration Maintenance App

NONE ABAP BASED DEVELOPMENT ON SAP BUINESS TECHNOLOGY PLATFORM

- The SAP Business Technology Platform
- Start developing on SAP Business Technology Platform
- SAP Business Technology platform developer center

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

A	
ABAP for cloud development	Refined, simplified, and standardized ABAP language (based on → Classic ABAP) for cloud-ready programming. Only modern ABAP statements and concepts, with a focus on cloud-enabled, object-oriented design and modern programming models are allowed
ABAP development tools (ADT)	Integrated development environment (IDE) for ABAP development implemented as a plugin for the Eclipse platform (www.eclipse.org)
ABAP Platform reuse services	A set of technical and business-related services to be easily integrated and used in ABAP development, addressing recurring development requirements
ABAP Cloud (development model)	A modern development model for cloud-ready, and upgrade-stable ABAP applications and extensions that follows the clear cloud rules to avoid the need to make adaptations after a version change of the SAP software
ABAP test cockpit (ATC)	Toolset directly integrated in the ABAP development tools for doing static and dynamic quality checking of ABAP code and associated objects
B	
Business user	Persona that personalizes SAP SaaS applications (e.g., adapt UI layouts, report views and dashboards, ...), by using → key user (extensibility) tools
C	
Citizen developer	Persona that composes and extends applications and automations on SAP Business Technology Platform using the → Key user (extensibility) tools SAP Build apps, SAP Business application studio, SAP Build Process automation and others)
Classic ABAP	Highly optimized programming language for business applications, both small-scale and large-scale. Classic ABAP allows developers to use the full range of ABAP statements and concepts provided by SAP
Classic (custom) (ABAP) code	ABAP code and artefacts developed for the traditional ABAP programming models such as Dynpro, BSP and Web Dynpro
Classic custom ABAP extensions	→ Classic extensibility (model)
Classic extensibility (model)	An extensibility model that offers a great degree of freedom for classical ABAP code, but that requires serious adaptation efforts after an upgrade
Clean core	The *clean core* concept is a mindset and philosophy supported with governance and guidelines that lay the foundation for a flexible and future ready ERP. The concept includes different dimensions as extensibility,

	integration, data, processes, and operations. With the focus on extensibility, the *clean core* means to keep extensions strictly separate from the SAP application. Extensions access SAP business objects only through well defined, released and upgrade-stable interfaces.
Cloud extensibility model	An extensibility model for SAP S/4HANA that ensures that the customer and partner extensions continue to run without any change or adaptation effort independent of changes made by SAP during cloud updates
Cloud-optimized ABAP language	→ ABAP Cloud
E	
Embedded steampunk	SAP internal project name for → SAP S/4HANA Cloud ABAP environment
H	
Hybrid extensions	An extension scenario where different extensibility options (→ key user extensibility, → on-stack (developer) extensibility, → side-by-side extensibility) are jointly used
I	
In-app extensibility	→ key user extensibility
K	
Key user	Persona that extends SaaS applications by using → key user (extensibility) tools to implement simple business requirements (custom fields, layout changes, validation rules, simple custom logic, ...)
Key user (extensibility) tools	A set of web-based, easy-to-use low-code/no-code tools used for → key user extensibility
Key user extensibility	An extensibility approach that allows key users to make small adaptions to standard applications or create lightweight custom apps without advanced development skills. The key user implements the extensions and apps directly on the software stack using dedicated key user apps that provide users with detailed guidance and convenient standard solutions for common problems
L	
Local public interfaces	Local public interfaces provide non-remote access to SAP APIs. These stable interfaces are the key to the lifecycle independence of SAP code and customer and partner extensions
Logical unit of work	A sequence of programming steps and data updates distributed across multiple work processes, whose database changes are updated within a single database commit

Loosely-coupled extensions	Extensions with a low proximity to data, transactional behavior, or apps and therefore typically run on a separate platform like the → SAP Business Technology Platform
Low-code/no-code tools	Toolset to develop apps with little or no code, mostly visual and drag-and-drop simplicity, based on a large library of pre-built content. → key user (extensibility) tools
LUW	→ Logical unit of work
O	
On-stack (developer) extensibility	An extensibility approach that allows ABAP developers to create sophisticated extensions and apps using a development environment that supports professional development teams. In the ABAP Platform, developer extensibility allows ABAP developers to implement the extensions and apps directly on the software stack using the → ABAP development tools
Original (RAP) BO	The original RAP BO is the base BO on which the RAP BO extension bears
P	
Private cloud	A cloud infrastructure that allows customers to run their solutions with maximum flexibility. However, adaptations are typically necessary after a version change of the SAP solution
Public cloud	A standardized cloud infrastructure that allows customers to run their solutions with minimal cost. A version change of SAP software has no impact on the custom extensions
Public interface	An interface such as publicly released SAP APIs and extension points that are publicly documented and available with stable semantics after a version change of SAP software
R	
Released APIs	An API that is publicly documented and released for use in extensions
Released interfaces	→ Released APIs
Released objects tree	A view in → ABAP development tools project explorer showing all SAP released → Public interfaces and local extension points
Remote API	An interface of a software component allowing communication between two or more stacks, providing a service to other software components
S	
SAP BTP	→ SAP Business Technology Platform

SAP Business Technology Platform, ABAP Environment	SAP Platform-as-a-Service (PaaS) offering for ABAP development that enables developers to make use of their traditional on-premise ABAP knowledge to develop and run ABAP applications in the SAP Business Technology Platform, either as an extension to SAP software or as standalone applications (https://community.sap.com/topics/btp-abap-environment))
SAP Business Technology Platform (SAP BTP)	SAP Business Technology Platform brings together application development, data and analytics, integration, and AI capabilities into one unified environment optimized for SAP applications (https://community.sap.com/topics/business-technology-platform)
SAP extension points	Extension points (such as Business Add Ins) offered by SAP to customers and partners to extend the SAP logic with custom and partner logic through well-defined interfaces
SAP S/4HANA Cloud ABAP environment	SAP S/4HANA Cloud ABAP environment enables modern ABAP development for SAP customers and partners of → tightly coupled extensions *running directly on the SAP S/4HANA technology stack*
SAP S/4HANA	SAP's flagship enterprise resource planning product (ERP) is available in the cloud (→ SAP S/4HANA Cloud Public Edition or → SAP S/4HANA Cloud Private Edition) or on-premise (→ SAP S/4HANA on premise)
SAP S/4HANA on premise	SAP S/4HANA on premise offers enterprise management capabilities for the Intelligent Enterprise. SAP S/4HANA is deployed and managed in the customer's premises
SAP S/4HANA Cloud Private Edition	SAP S/4HANA Cloud Private Edition offers enterprise management capabilities for the Intelligent Enterprise. SAP S/4HANA Cloud Private Edition is deployed and managed by SAP in private cloud data centers offered by hyper scalers and SAP
SAP S/4HANA Cloud Public Edition	SAP S/4HANA Cloud Public edition offers enterprise management capabilities for the Intelligent Enterprise. SAP S/4HANA Cloud Public Edition is deployed and managed by SAP in public cloud data centers offered by hyper scalers and SAP
SAP API Business Hub	An SAP operated platform to explore, discover and consume APIs, pre-packaged integrations, business services and sample apps from SAP (SAP Business Accelerator Hub)
Side-by-side extensibility	An extensibility approach that allows developers to implementing sophisticated development projects, such as creating custom UIs, custom applications (in Java, Node.JS, ABAP), or analytical or workflow extensions. The development projects are implemented on → SAP Business Technology Platform (SAP BTP) and integrated via released remote APIs
Side-by-side extension	→ side-by-side extensibility
Steampunk	SAP internal project name for → SAP Business Technology Platform, ABAP environment

T

Tightly coupled extensions	Extensions with a proximity to data, transactional behavior, or apps and therefore typically run on the same stack