



PUBLIC

Document Version: Latest – 2023-05-03

Best Practices to Generate Knowledge Base Runtime Versions

Content

- 1 Quick Reference – Knowledge-Base Runtime Version Generation (KBGEN) 4**
- 2 Basic Concepts of How Knowledge Bases and Runtime Versions Work. 5**
 - 2.1 Knowledge-Base Objects. 5
 - 2.2 Knowledge-Base Profiles. 6
 - 2.3 Generating New Runtime Versions. 6
 - 2.4 Regenerating Existing Runtime Versions. 7
 - 2.5 Configuration Using Runtime Versions. 8
- 3 Designing Knowledge Bases and Runtime Versions. 10**
 - 3.1 Changes for Compatibility. 10
 - 3.2 The Delta List(s). 10
 - 3.3 Changes for User Experience. 11
 - 3.4 Changes for Efficiency and Performance. 11
- 4 Naming Convention Recommendations. 13**
 - 4.1 Simplified Example. 13
 - 4.2 Knowledge Base Object Naming. 13
 - 4.3 Runtime Version Naming. 14
- 5 Knowledge Base Profiles. 16**
 - 5.1 Simplified Single Level Example. 16
 - 5.2 Real World Considerations. 17
 - 5.3 Extreme 1: Every KMAT In A Single KB. 18
 - 5.4 Extreme 2: Every KMAT In Its Own KB. 18
 - 5.5 Happy Medium: Similar KMATs In The Same KB. 19
 - 5.6 Simplified Multi-Level Example. 20
 - 5.7 Product Family - Single-Level Example with Additional Focus on Constraint Nets. 21
 - 5.8 Summary. 26
- 6 Runtime Version Generation 28**
 - 6.1 Number of RTVs Needed for a KB. 28
 - 6.2 When to Generate New RTVs. 29
 - 6.3 When to Regenerate Existing RTVs. 29
 - 6.4 RTVs Aligned to ECM Change Numbers. 29
 - 6.5 Real World Considerations. 30
 - 6.6 Classic and Advanced Knowledge Base Runtime Version Generation (KBGen). 31
 - 6.7 How to Switch the KB Generation Mode?. 32

6.8	KB and RTV Maintenance Utilities.	32
7	System Landscape Considerations.	34
7.1	Configuration Engine Versions.	34
7.2	Filter Unnecessary KB RTVs	34
7.3	SAP CPQ.	34
7.4	SAP ERP.	35
8	Advanced Use Cases.	36
8.1	Multiple Configuration Profiles.	36
8.2	Runtime Version Deletion.	36
8.3	No BOM Explosion.	37
9	Conclusion.	38
10	Appendix A – Report COM_CFG_DB_STAT SCREENSHOTS.	39

1 Quick Reference – Knowledge-Base Runtime Version Generation (KBGEN)

Basic Concepts	Knowledge-Base Objects [page 5]	Knowledge-Base Profiles [page 6]	Generating New Runtime Versions [page 6]	Configuration Using Runtime Versions [page 8]
Design Planning	Designing Knowledge Bases and Runtime Versions [page 10]		Naming Convention Recommendations [page 13]	
Setup and Maintenance	KB and RTV Maintenance Utilities [page 32]	Knowledge Base Profiles [page 16]	When to Generate New RTVs [page 29]	RTVs Aligned to ECM Change Numbers [page 29]
Advanced Concepts	Multiple Configuration Profiles [page 36]	Number of RTVs Needed for a KB [page 28]	Filter Unnecessary KB RTVs [page 34]	Classic and Advanced Knowledge Base Runtime Version Generation (KBGen) [page 31]

2 Basic Concepts of How Knowledge Bases and Runtime Versions Work

A basic understanding of Knowledge Bases (KBs) and Runtime Versions (RTVs) is required before you can understand the best practices and decisions for your implementation. The following pages contain excerpts from the SAP Online Help on this topic. See the online help for details on how to create and maintain these data.

2.1 Knowledge-Base Objects

To configure a product, the Variant Configuration (VC) service of SAP Variant Configuration and Pricing cannot directly access master data that was created and maintained in SAP S/4HANA or SAP ERP. For this reason, the master data must be grouped together in knowledge-base objects (KB objects) and made available in the form of runtime versions (RTVs) that can be replicated in SAP Variant Configuration and Pricing.

The KB object groups together all master data that the Variant Configuration service needs to configure a product. This includes the following VC objects:

i Note

Note that task lists are not included in KB objects.

- Configurable materials
- Classes
- Characteristics and related values
- Object dependencies
- One configuration profile for each configurable material
- BOMs and BOM components of configurable materials
- Variant tables
- Interface designs

A KB object is characterized by:

- Basic data: name, description, and status
- Language-dependent descriptions
- Optional dependency group parameter to simplify the search for KB objects
- Optional authorization group maintenance
- One or more KB profiles.

2.2 Knowledge-Base Profiles

KB profiles are part of the KB object and allow grouping multiple products. The profile determines if a product can be used as root product during configuration with the Variant Configuration service of SAP Variant Configuration and Pricing.

Adding a new KB profile for a product is equivalent to assigning a product to a KB object. You can define one or more profiles for a KB object. By defining several profiles, you can group several materials together in one knowledge base. This way, runtime versions can be generated faster, particularly if the affected materials have common objects, such as variant tables.

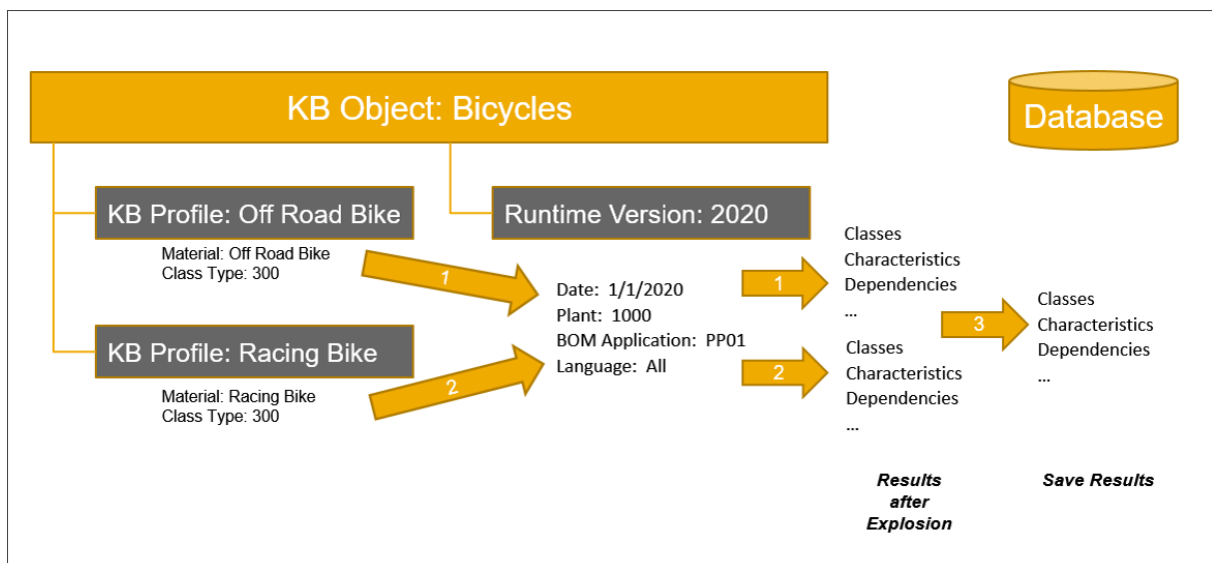
2.3 Generating New Runtime Versions

The KB object comprises the master data that describes a configurable product during configuration. The KB runtime version is created for a specific *valid-from date* and context. Thus, the RTV represents a snapshot of the master data needed for configuration. The Variant Configuration service uses KB runtime versions that are replicated for this purpose.

You specify a runtime version with the following information:

- A *valid-from date* which determines the date when the snapshot of the KB object is made
- Exactly one plant
- Exactly one BOM application
- One or all languages

The graphic below illustrates the runtime generation process. The configurable material for each KB profile is exploded using the parameters in the runtime version. The result of each explosion contains all relevant classes, characteristics, dependencies, BOMs, variant tables, and more for that configurable material. Finally, the distinct data from each explosion is saved for the runtime version.



i Note

Note: All objects of the KB object, such as characteristics, have the same validity in the runtime version. For future use of VC objects in the product configuration process with a valid-from date, i.e., when using Engineering Change Management (ECM), a separate runtime version with an appropriate valid-from date is required. See the second graphic in the following section.

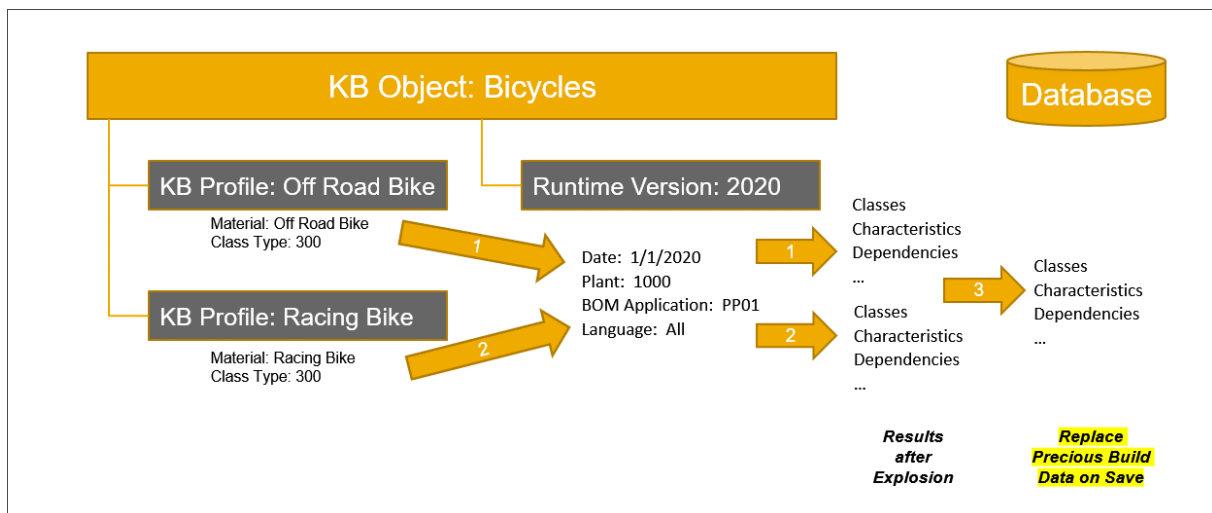
2.4 Regenerating Existing Runtime Versions

A runtime version (RTV) collects the master data as it was created. You regenerate an RTV if you want to:

- Activate changes you have made to the master data
- Change the attributes of the runtime version, such as the *valid-from date*

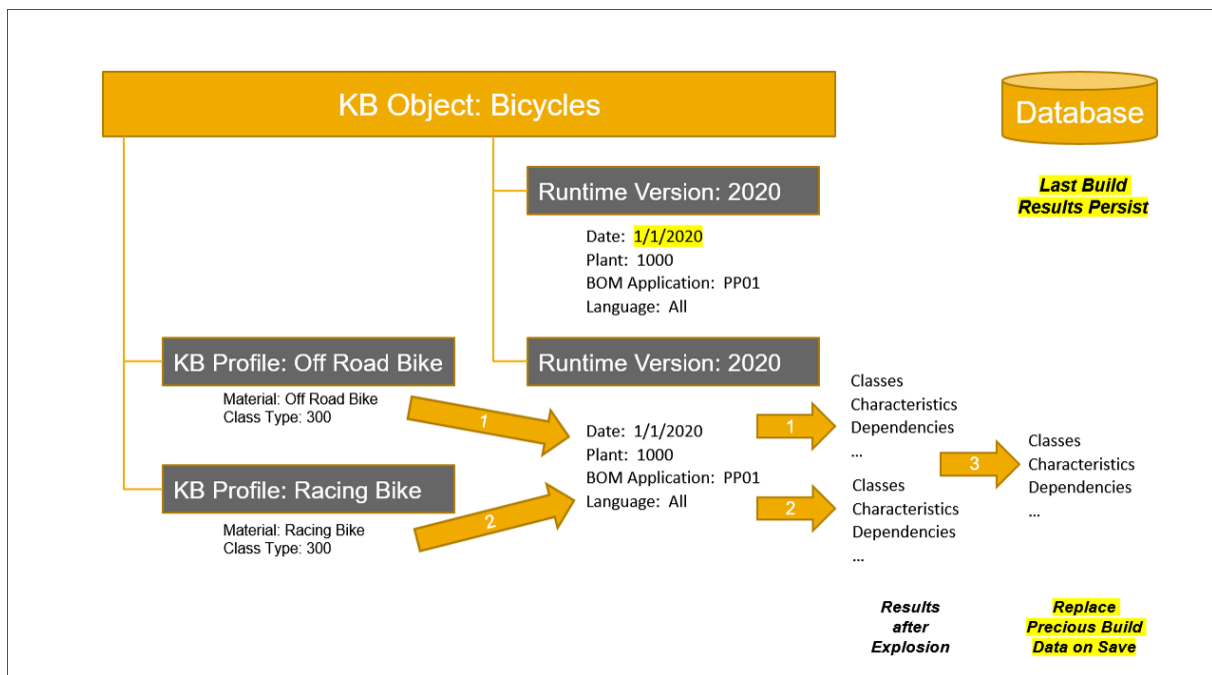
Regenerating overwrites the database records of a runtime version and creates a new build. When a runtime version is regenerated, the *valid-from date* is not automatically updated, it remains the same as in the previous build. You can manually change the *valid-from date* when regenerating a runtime version, if you want the valid-from date to be today. Changing the *valid-from date* of an RTV may change several of its objects as the master data is read with respect to the new *valid-from date*.

The graphic below illustrates the runtime regeneration process. It works essentially the same way as the generation process except that the data in the previous build is overwritten with the new result.



The build number of an RTV is automatically incremented whenever an RTV is successfully regenerated. For example, the build number after initial generation of an RTV is **1**. The build number of the next regeneration of that RTV is **2**, and so on. Previous builds for a given RTV are not stored in the database. Only the latest build persists and is loaded by Variant Configuration service for the RTV. Also, note that it is possible to check the regeneration of an RTV without saving it in the database. This can be useful for evaluating the effect that VC modeling changes would have on the current RTV build.

If you want to provide model versions for a *valid-from date* in the future, you must create a new, separate runtime version with the appropriate validity data. The graphic below illustrates how the original runtime version remains while a new and separate runtime version is created.

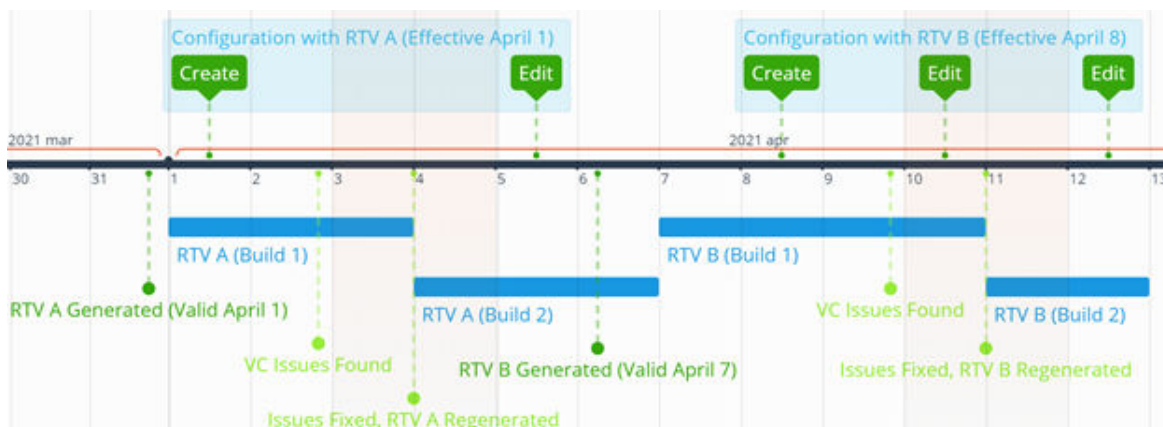


2.5 Configuration Using Runtime Versions

This example illustrates how different use cases for configuration and reconfiguration can interact with modeling fixes and product changes for a given configurable product. The timeline graphic is explained as follows:

1. **Build 1 of RTV A** is generated on March 31 with **valid-from date April 1**.
2. **Configuration 1** is created on April 1 with **effective date April 1**.
→ **Build 1 of RTV A** is selected because it is valid from April 1.
3. On April 2, issues with the VC model are reported and fixed.
4. On April 4, **RTV A** is regenerated to **build 2** with **valid-from date April 1**.
→ **Build 2 of RTV A** is used immediately, any model changes valid at a future date, e.g. when using ECM, are not considered.
5. On April 5, **configuration 1** is edited with **effective date April 1**.
→ **Build 2 of RTV A** is selected.
6. **Build 1 of RTV B** is generated on April 6 with **valid-from date of April 7**. The configurable product has new features that become available and have been modeled with an ECM change number for configurations placed on or after April 7
7. **Configuration 2** is created on April 8 with **effective date April 8**
→ **Build 1 of RTV B** is selected because it is valid from April 7. RTV A still exists but is only selected for configurations with an effective date between April 1 and April 6.
8. On April 9, issues with the new features in the VC model are reported and fixed. The previous ECM change number with *valid-from date* of April 7 is used to make the model changes
9. On April 10, **configuration 2** is edited (effective date remains April 8)
→ **Build 1 of RTV B** selected. Changes done on April 9 are not present (no new build was generated).

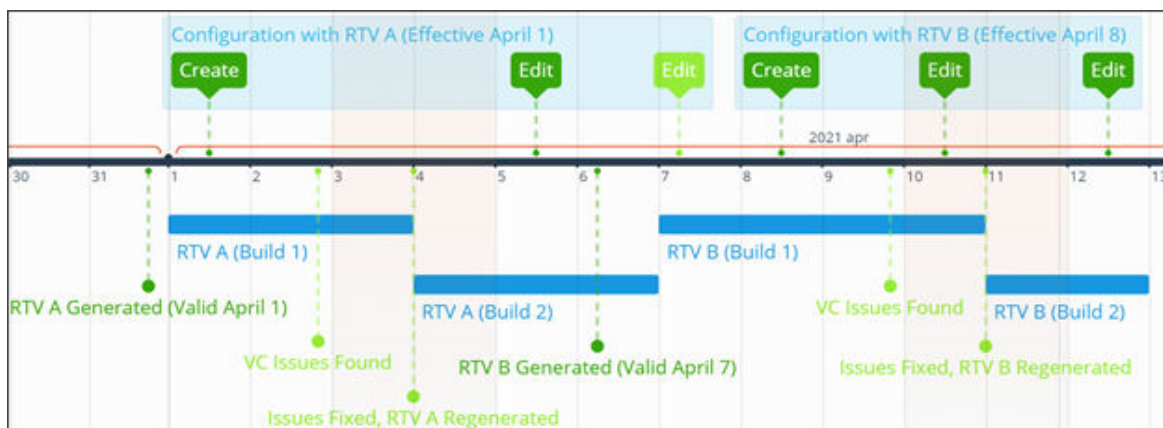
10. On April 11, **RTV B** is regenerated to **build 2** with **valid-from date April 7**.
11. On April 12, **configuration 2** is again edited (effective date remains April 8).
→ **Build 2 of RTV B** is selected (valid from April 7).



Note that runtime versions are selected using the effective date of the configuration rather than the system date. This can require implementing business logic to set the appropriate configuration effective date for reconfigurations that cross over a new RTV *valid-from date*. Continuing the previous example, if the first configuration is again edited on April 7, which RTV should be used?

→ If the effective date of the configuration remains April 1, then build 2 of RTV A will be selected.

12. However, the new features in RTV B are not available in RTV A even though they are offered for new configurations on or after April 7.
→ If the effective date of the configuration should change to April 7, then build 2 of RTV B will be selected.
13. The effective date must change if any features offered in RTV A are discontinued and no longer offered as of April 7, i.e., they must not be selected in the reconfiguration.
14. The selection of RTV B may result in inconsistencies in the configuration if obsolete features were previously specified and saved in the configuration using RTV A.



3 Designing Knowledge Bases and Runtime Versions

You may need to make changes to existing Variant Configuration (VC) models so that they are compatible with the knowledge-base runtime version generation. Even if your VC models are compatible to generate knowledge-base runtime versions (RTVs), you may choose to make changes to improve the user experience as well as RTV generation, and runtime efficiency. This can also lead to performance improvements. The table below summarizes the different preparations and the following sections discuss considerations for them.

KB Compatibility	User Experience	Efficiency / Performance
Generation errors must be resolved to save an RTV	Characteristic granularity must be appropriate for all use cases	Advanced KB generation optimally stores variant tables
Generation warnings must be evaluated and addressed to ensure desired RTV behavior	Conflict situations must be avoided or easily understood and resolved	Bloated RTVs consume more storage and take longer to load
Delta-list discrepancies must be avoided and/or addressed	Characteristic/value descriptions must be understood by all users	Filter BAdI limits RTV data to the minimum needed

3.1 Changes for Compatibility

Do you need to change your VC models to generate runtime versions? You can answer that question by creating KB objects and then attempting to generate runtime versions. The runtime version generation process will display information and potentially warnings and errors. Warnings indicate situations that you may need to address for compatibility; errors indicate situations that you must address.

If you have many VC models, it can be very time consuming to create all the KB objects and RTVs that you need to determine the compatibility of your models. You likely do not yet know how to optimally define your KB objects and KB profiles which makes this more difficult. This could lead to rework when you do decide how to do this. If you are new to KB generation, you will likely benefit from working with experts, such as consultants and partners who have performed this conversion process many times.

3.2 The Delta List(s)

The *Delta List* is a synopsis of the key differences between LO-VC models and engines using RTVs. Some items on the *Delta List* are informational and thus educate the reader about differences in the use of LO-VC

and RTVs. Other items highlight modeling changes that may be required to make it possible to generate an RTV of an LO-VC model. The remaining items denote LO-VC features or modeling constructs that are simply not supported in KBGen and must be redesigned for compatibility. There are three key challenges to fully understand the *Delta List*:

1. The *Delta List* is continually updated. Items are added if new differences are found and items are removed when SAP makes program changes to eliminate a difference to improve compatibility. SAP removes differences when it is determined that customers are significantly challenged by a given difference and eliminating that difference requires a reasonable amount of development. The removal of differences is typically delivered with an ERP Enhancement Pack or on Support Pack level. This means that you may need to upgrade and/or install Support Packs to remove a specific difference if you are currently running an earlier level.
2. There is not a single, comprehensive *Delta List*, but several sources. The SAP online help provides one source of the *Delta List*, but this help occurs in multiple places and occasionally becomes out of date when differences are removed. The most up-to-date source regarding detailed technical information is provided via SAP notes. SAP Notes like [1819856](#) are a good starting point, but other notes better address specific incompatibilities. Those SAP notes include [651112](#), [664274](#), [837111](#), [1529481](#), and [1537346](#). Be careful as some notes are superseded by other notes as differences change or are eliminated.
3. ERP message class VSCE is used in RTV Consistency Checker. Not all differences are detected by the RTV Consistency Checker, but all those that prevent the generation of an RTV are.

3.3 Changes for User Experience

If your VC models were developed primarily by engineering or manufacturing for back office order entry, these models may not be suitable for future use by less technical users like salespeople (in CPQ) or end customers (in Commerce Cloud). The full scope of this topic is beyond the scope of this technical best practices document, but it is an important consideration for successful user adoption of your configuration solution throughout your SAP system landscape.

Perhaps the most common user experience problem is the resolution of configuration inconsistencies and conflicts. Declarative logic enabled through restrictable characteristics and constraints are the best way to avoid such conflicts. On the other hand, procedural logic with heavy use of procedures and preconditions may result in conflict states that a user cannot understand how to resolve. For example, if the user is expected to specify characteristics in a specific sequence but does not or backtracks to change a previously specified characteristic without respecifying each subsequent characteristic.

3.4 Changes for Efficiency and Performance

Some companies have RTVs that still include every characteristic value, variant table row, and object dependency that has ever been used in the lifecycle of a KMAT, even though most of those characteristic values are no longer offered, and the related object dependencies are no longer relevant.

If you are not using ECM change numbers to continually update your VC models, you may have this situation. Deleting objects from a VC model without ECM is generally not allowed, for instance if a class, characteristic, or

value was used in a saved configuration, nor prudent, for instance when deleting an object dependency that is needed to correctly edit or display saved configurations.

Bloated RTVs consume more storage and memory than necessary, and they take longer to generate, migrate, and load. While you can technically operate this way, a good practice in VC design and maintenance follows the axiom that less is more. See [KB and RTV Maintenance Utilities \[page 32\]](#) to determine precisely what an RTV version contains.

Filter out unnecessary documents

You can filter unnecessary documents such as image files and PDFs linked to materials, characteristics, and characteristic values out of your RTVs by implementing a BADI. During KB RTV generation, these data are included in the RTV even though they are in most cases not relevant for the IPC and they are not supported for Variant Configuration service. If these documents are not required in your implementation, it is strongly recommended to implement the BADI described in SAP Note [1030427](#) to suppress the replication of the documentation data to the *COMM_CFG** tables. This will considerably lower the amount of data to be copied and speed up both the initialize report *CFG_ERP_INITIALISE_DB* and the request report for an individual KB RTV in report *CFG_ERP_REQUEST_DB*. In the case of Variant Configuration service, it additionally reduces the data to be replicated to the cloud.

4 Naming Convention Recommendations

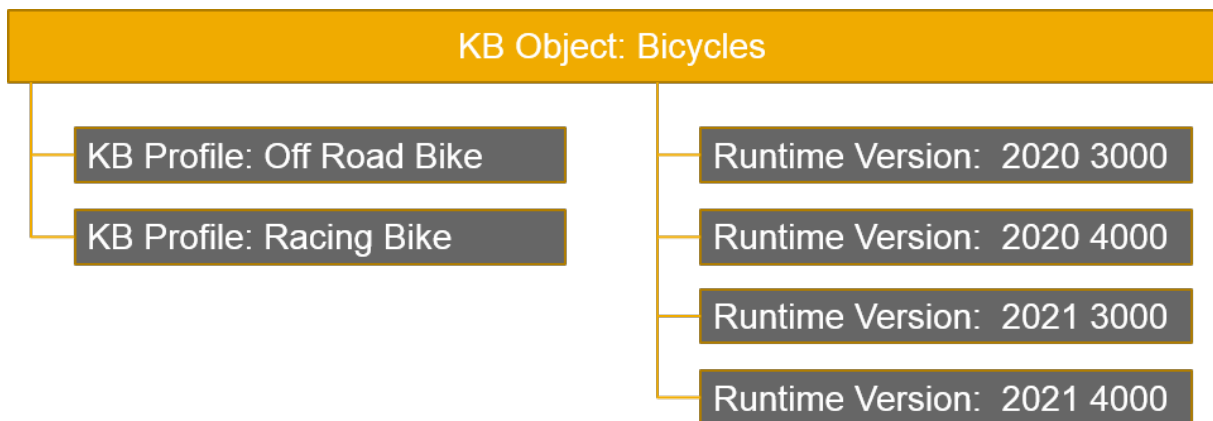
When Knowledge base (KB) objects contain multiple KB profiles with multiple runtime versions (RTVs), an intuitive naming convention is quite useful to indicate what data each one contains. Your naming convention could follow these generally suggested patterns:

- KB Object: a general name, e.g., the product family or product hierarchy that it represents
- KB Profiles: the same name as the configurable material in the profile
- RTVs: a concatenation of relevant key fields in its header

The following simplified example and explanations illustrate various considerations for establishing your naming convention.

4.1 Simplified Example

A company makes two configurable bicycles, models 3000 and 4000, in two different plants. There are subtle differences in the configuration structures of the bikes in each of the plants. The company changes the features and options of the model of each year, for instance 2020 and 2021. The graphic below illustrates how these names follow the suggested naming convention.



4.2 Knowledge Base Object Naming

Like other VC naming conventions, you should avoid *hard coding* the name of a knowledge base (KB) object that may reasonably evolve in the future. For example, if your KB object initially contains only one KB profile which contains only one KMAT, it may be tempting to name your KB object the same as the corresponding KMAT. Over time however, it may be advantageous to add additional KB profiles to that KB object, making the

existing KB object name confusing since it implies the existence of only one KB profile. Remember that you can always update the KB object's descriptions if you wish to convey up-to-date information about an object.

4.3 Runtime Version Naming

The name should generally align with the header data of the RTV in the sequence shown below:

1. Valid-from date
2. Plant
3. BOM Application (optional, see below)
4. Language Key (optional, see below)
5. Configuration Profile (optional, see Advanced Use Case section)

Valid-from date:

From the publication of SAP note [2932857](#) on, the PMEVC sorts multiple runtime versions for a given KB object by *valid-from date* and *version name*. There are four commonly used validity conventions:

1. A series designation such as the year of the model (e.g., 2020, 2021)
2. A version number, like the one used for software (e.g., 1.0, 2.0)
3. An 8-digit date, with format *YYYYMMDD* (e.g., 20200101, 2021010)
4. An ECM change number

The first two conventions are tied to product releases. The modeler decides when a new release, and corresponding runtime version, is required.

While dates are commonly used in runtime version names, they do have one distinct disadvantage. If you decide to change the *valid-from date* of the runtime in a later build, the runtime version name would no longer correspond to the actual runtime version *valid-from date*. In such cases, you need to create a new runtime version to keep consistent naming even if you do not need a separate version for validity purposes, therefore, the two runtime versions may have the same contents on those *valid-from dates*.

Including an ECM change number in your runtime version name makes sense if you relate your VC modeling effectivity to change numbers. In this scenario, you need one runtime version for each change number. This assumes that each change number used in your VC model has a unique *valid-from date*. An advantage of this approach is that you can date shift your ECM change number and corresponding runtime version without causing naming confusion. Note however that ECM change numbers and runtime versions are linked by name only, that is, changing the *valid-from date* of one, does not automatically change the *valid-from date* of the other, you must do this manually.

Plant:

If your configurable materials are sold by multiple plants, and there are configuration differences between those plants, then you will need separate runtime versions by plant. If there are no configuration differences between plants, you can use the runtime version for one plant in any other plant. However, it is still recommended to include the plant in the runtime version name in case plant-specific differences do occur in the future.

BOM Application:

Typically, the same BOM application will be used for all runtime versions of a KB object, however it is possible that a different BOM application is required in the future. In such cases, the BOM application can be appended to the end of the runtime version name and omitted if it is not necessary.

Language:

Runtime versions are typically created for all languages, but it is possible to create them as language specific. In such cases, the language key can be appended to the end of the runtime version name and omitted if the runtime version is not language specific.

5 Knowledge Base Profiles

A knowledge base must contain one or more profiles. These profiles determine which configurable materials can be configured at the root level using its runtime versions. Design time and runtime performance can be optimized by strategically deciding which configurable materials have profiles in the same knowledge base. The following example provides a simplified illustration of how knowledge base profiles can be arranged. We will start with single-level configuration scenarios.

5.1 Simplified Single Level Example

There are three configurable materials (KMATs) in your system named *A*, *B*, and *C*. For the sake of clarity, we will only consider characteristics but usually the KBs contain a multitude of other configuration objects, such as large variant tables, which can have stronger effects on performance and memory. In our example, the KMATs each have five characteristics.

KMAT A	KMAT B	KMAT C
Char1	Char2	Char6
Char2	Char3	Char7
Char3	Char4	Char8
Char4	Char5	Char9
Char5	Char6	Char10

You need to decide the best way to define the structure of the knowledge base for these three materials. The following scenarios are possible:

- A separate KB object for each KMAT: three KB objects with each having a single KB profile for *KMAT A*, *KMAT B*, and *KMAT C*
- Two KMATs share one KB object: one KB object *AB* with a KB profile for *KMAT A* and a KB profile for *KMAT B*, one KB object *C* and KB profile for *KMAT C* or any other combination
- All three KMATs share a single KB object: The KB object *ABC* has three profiles, one for each KMAT

The characteristics that are included in each KB runtime version are shown below where KB stands for KB object.

KB A	KB B	KB C	KB AB	KB AC	KB BC	KB ABC
Char1	Char2	Char6	Char1	Char1	Char2	Char1

KB A	KB B	KB C	KB AB	KB AC	KB BC	KB ABC
Char2	Char3	Char7	Char2	Char2	Char3	Char2
Char3	Char4	Char8	Char3	Char3	Char4	Char3
Char4	Char5	Char9	Char4	Char4	Char5	Char4
Char5	Char6	Char10	Char5	Char5	Char6	Char5
			Char6	Char6	Char7	Char6
				Char7	Char8	Char7
				Char8	Char9	Char8
				Char9	Char10	Char9
				Char10		Char10

Leaving aside the fact that due to the simplicity of the example, a single KB object would not be problematic from a performance perspective, what are the optimal KB definitions for these three KMATs?

The best overall option is KB object *AB* and KB object *C*. Since *KMAT A* and *KMAT B* share most of their characteristics, a shared KB object *AB* is very efficient and consumes the least memory and cache size. *KMAT C* shares only one characteristic with *KMAT B* and no characteristics with *KMAT A*, so including *KMAT C* with either of the other KMATs would nearly double the size of the resulting KB. If you change Char6, then you need to regenerate runtime versions for *KB AB* and *KB C* since both KBs contain that characteristic. If you change any other Char, then you only need to regenerate runtime versions for *KB AB* or *KB C* depending on which KB contains the characteristic you changed.

A single KB object *ABC* would be the simplest to maintain. If you change any characteristic, then you regenerate the runtime version of that KB. You do not need to decide which runtime versions to generate because you have only one KB. Runtime versions for this KB will take nearly twice as long to load as KBs that have half the characteristics.

Separate KB objects, KB object *A*, KB object *B*, and KB object *C*, would provide the best runtime performance. Runtime versions will load as fast as possible since they contain the absolute minimum required data because there is no data from other KMATs. However, if you change any characteristic, you must determine which runtime versions to regenerate, that is which KB objects contain the Char that was changed

5.2 Real World Considerations

The preceding example was intentionally simplified so that it could be easily understood. In that case, a single *KB ABC* would be fine because the KMATs in the example are so small. Even doubling any performance metric would amount to a few microseconds which would be imperceptible to a user.

Furthermore, characteristics are only one of the VC objects to consider in the size of a KB or runtime version. Variant tables with many large numbers of rows and columns often consume the most processing time, to generate or load, and the most storage space.

The following sections describe the extremes of KB profile definition as well as the *happy medium* that falls somewhere between those extremes and represents the optimal KB definition for a given implementation. For these explanations, assume that we have 100 large and complex KMATs, so the optimal KB definition will have a large impact on overall maintenance and performance.

5.3 Extreme 1: Every KMAT In A Single KB

As explained in the simplified example above, the simplest maintenance option is to have a single KB with profiles for every KMAT. However, typically this is a bad design for large VC implementations for the following reasons:

- Runtime versions will be very large and will take a long time to generate
 - You must regenerate the full runtime version after any VC change
 - Regeneration may time out
 - Moving your runtime version to applications such as Variant Configuration and Pricing may take a long time
 - *Trial and error* debugging will be painfully slow: make one VC change, regenerate and move runtime version, test
- Loading the runtime version to configure any KMAT will take a long time
 - Applications like SAP Variant Configuration and Pricing must load the full KB to configure any KMAT in the KB
 - This load time will be noticeably slow because the entire KB must be loaded even though only a small fraction of the KB data may be needed
 - You may run out of memory to load such a large runtime version

5.4 Extreme 2: Every KMAT In Its Own KB

Some customers are not aware of the possibility to group several configurable materials, KB profiles, in one KB. As a result, they create a separate KB for every configurable material. Here are some advantages and disadvantages of this scenario:

- Individual KB runtime versions will be as compact as possible, but you will have to maintain many
 - Regeneration of an individual runtime version is done as fast as possible
 - *Trial and error* debugging of a single KMAT is done as fast as possible: make one VC change, regenerate the runtime version for that KMAT, move runtime version, and test
 - To deploy a VC change that affects multiple KBs, you must determine all KBs that are affected and regenerate their runtime versions

- Because it is often difficult to determine which KBs are affected by a VC change, many companies regenerate all runtime versions, to make sure that none are inadvertently missed
- Regenerating runtime versions for all KBs after VC changes results in the longest cumulative maintenance duration to generate and move runtime versions, and in storage requirement because duplication of data is likely done across multiple KBs. To illustrate this point, we use the simplified example from above:
 - The sum of Chars in *KB A*, *KB B*, and *KB C* is 15. This is *Extreme 2*
 - The sum of Chars in *KB ABC* is 10. This is *Extreme 1*
 - The sum of Chars in *KB AB* and *KB C* is 11. This is the *Happy Medium*, see [Happy Medium: Similar KMATs In The Same KB \[page 19\]](#)
- Loading a runtime version to configure any KMAT will be as fast as possible
 - The runtime version will contain only the data needed to configure that KMAT
 - You will also consume the least amount of memory per runtime version

5.5 Happy Medium: Similar KMATs In The Same KB

When you understand the extremes above, then you are ready to define the *happy medium*, a balanced solution, for your KB definition. Keep in mind that no single approach is optimal for all implementations and that your perfect solution depends on your needs. The following section outlines the main considerations for making these decisions. The rest of this section outlines some common ways that KB definitions are aligned between the two extremes described above.

If you have well-formed design patterns that leverage object orientation and good naming conventions, then consider aligning KBs to product families, groups of configurable materials that are likely to share many VC objects. For example, your company has various KMATs for bicycles and various KMATs for computers, the bicycle KMATs and computer KMATs would be unlikely to share many VC objects. Combining all these KMATs into a single KB would generally not make sense. However, combining the bicycle KMATs into one KB and combining the computer KMATs into another KB could result in an efficient design.

Within a product family, it may be advantageous to further divide KMATs according to how they are modeled. For example, if off-road bicycle KMATs share a set of variant classes, constraint nets, and variant tables, while racing bicycles share a different set of variant classes, constraint nets, and variant tables, then you should consider one KB for off-road bicycles and another KB for racing bicycle KMATs.

In the end, the VC modelers know best where there is a high degree of commonality among KMATs. This commonality is what makes it efficient to share KB definitions.

Exception: Some companies implement VC in a way that no VC objects are shared across KMATs. All VC objects are KMAT specific. This approach does not leverage the robust object orientation capabilities of variant configuration and is therefore generally not considered good practice, although it is technically feasible. If this approach is used by your company, then the *Extreme 2* is your *happy medium*.

5.6 Simplified Multi-Level Example

You add three configurable materials (KMAT), named *X*, *Y*, and *Z*, to your system. They each have five characteristics as shown below.

KMAT X	KMAT Y	KMAT Z
Char11	Char12	Char16
Char12	Char13	Char17
Char13	Char14	Char18
Char14	Char15	Char19
Char15	Char16	Char20

KMAT Y and *KMAT Z* can be configured as subassemblies under *KMAT X*, i.e. the root item. *KMAT Y* can also be configured independently as a single level root item, *KMAT Z* cannot be configured independently. You need to decide the best way to define knowledge bases for these three KMATs. The following scenarios are possible:

- **A separate KB object for each KMAT X and KMAT Y:** KB object *X* with single KB profile for *KMAT X*, KB object *Y* with single KB profile for *KMAT Y*. *KMAT Z* should not have a KB profile in any KB object because it cannot be the root item
- **Two KMATs share a KB object XY:** KB object *XY* has a KB profile for *KMAT X* and a KB profile for *KMAT Y*. *KMAT Z* does not have any KB profile assigned

What are the optimal KB definitions for the three KMATs of this simple example?

KB object *Y* is technically redundant to KB object *X* and KB object *XY*, because KB object *X* and KB object *XY* contain the same data except for KB profiles.

KB object *X* technically contains all data needed to configure *KMAT Y* as a single level root item, but it will not be used because it does not have a KB profile for *KMAT Y*. The fact that *KMAT Y* can be a subassembly under *KMAT X* is not considered for root item configuration of *KMAT Y*. Creating a separate KB object *Y* for *KMAT Y* therefore serves no purpose.

The best option is the KB object *XY*. It contains all data needed to configure *KMAT X* as a root item and *KMAT Y* and *KMAT Z* as subassemblies because it contains a KB profile for *KMAT X*. It also contains all data needed to configure *KMAT Y* as a single level root item because it contains a KB profile for it.

This example illustrates why you should identify KMAT subassemblies that can also be root item configurations and then avoid creating separate KBs for those KMATs.

5.7 Product Family - Single-Level Example with Additional Focus on Constraint Nets

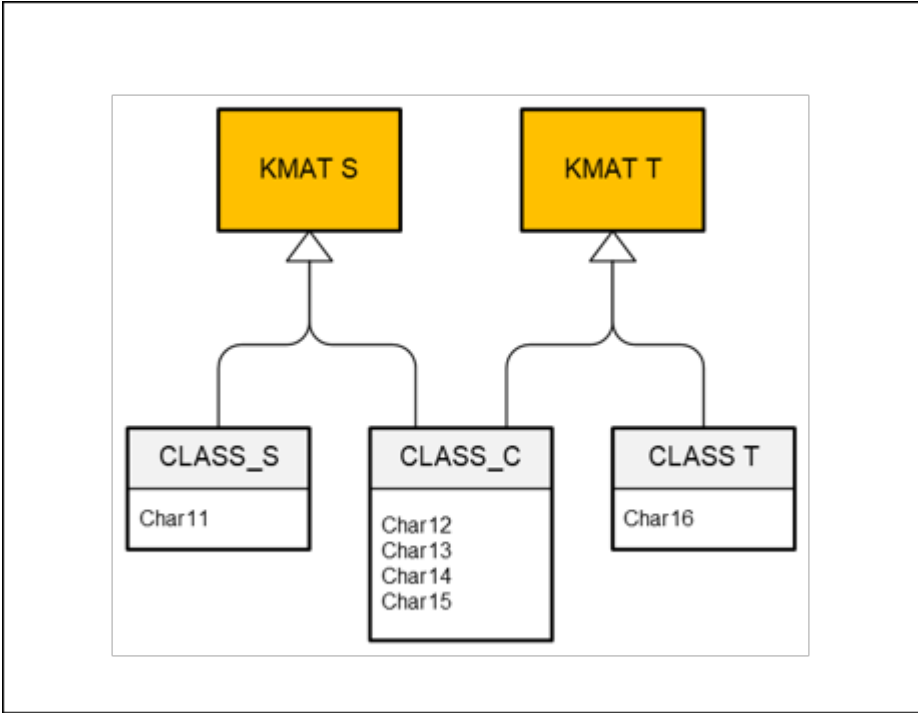
Now let us take a look at distinct single level products and when to combine them in one knowledgebase.

For different but similar products, we need to consider not only the reuse of common classes, characteristics, dependencies, and variant tables, but also the use of constraint nets because storing and loading the constraint nets differs slightly in the Knowledgebase Runtime-Version from the LO-VC.

Let us assume that you add two configurable materials (KMAT), named *S* and *T* to your system. Both have five characteristics, as shown below, and both inherit the characteristics from a common class *CLASS_C*, as well as from both *CLASS_S* and *CLASS_T*.

CLASS_C has assigned characteristics: *Char12*, *Char13*, *Char14*, and *Char15*.

CLASS_S has the assigned characteristic has the assigned characteristic *Char11*. *CLASS_T* has the characteristic *Char16*.



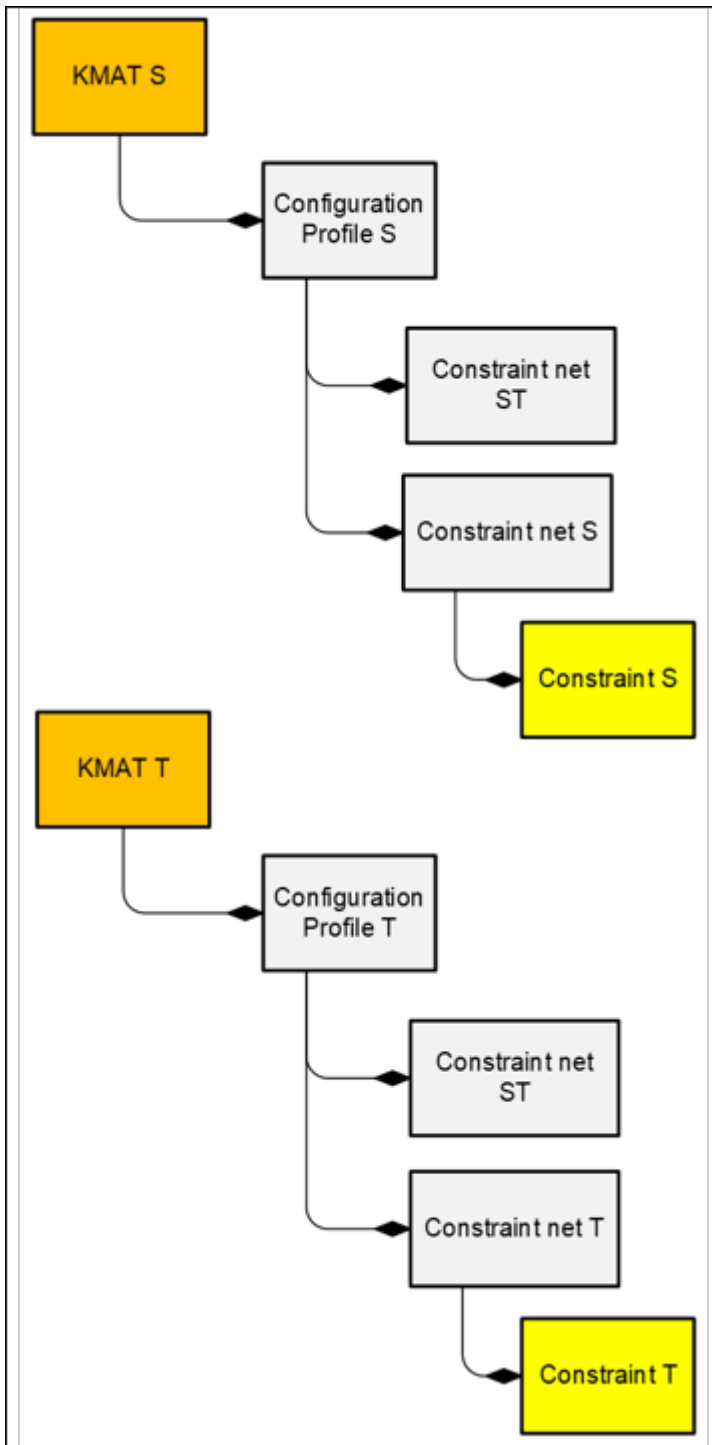
which result in:

KMAT S	KMAT T
Char11	Char12
Char12	Char13
Char13	Char14

KMAT S	KMAT T
Char14	Char15
Char15	Char16

Additionally, let us suppose that both products use common constraints, and each product uses specific constraints where product specific logic is implemented even for the common characteristics *CHAR12* to *CHAR15*. These specific constraints are assigned to extra constraint nets. The constraint nets are assigned to the respective configuration profiles for *KMAT S* and *KMAT T*.

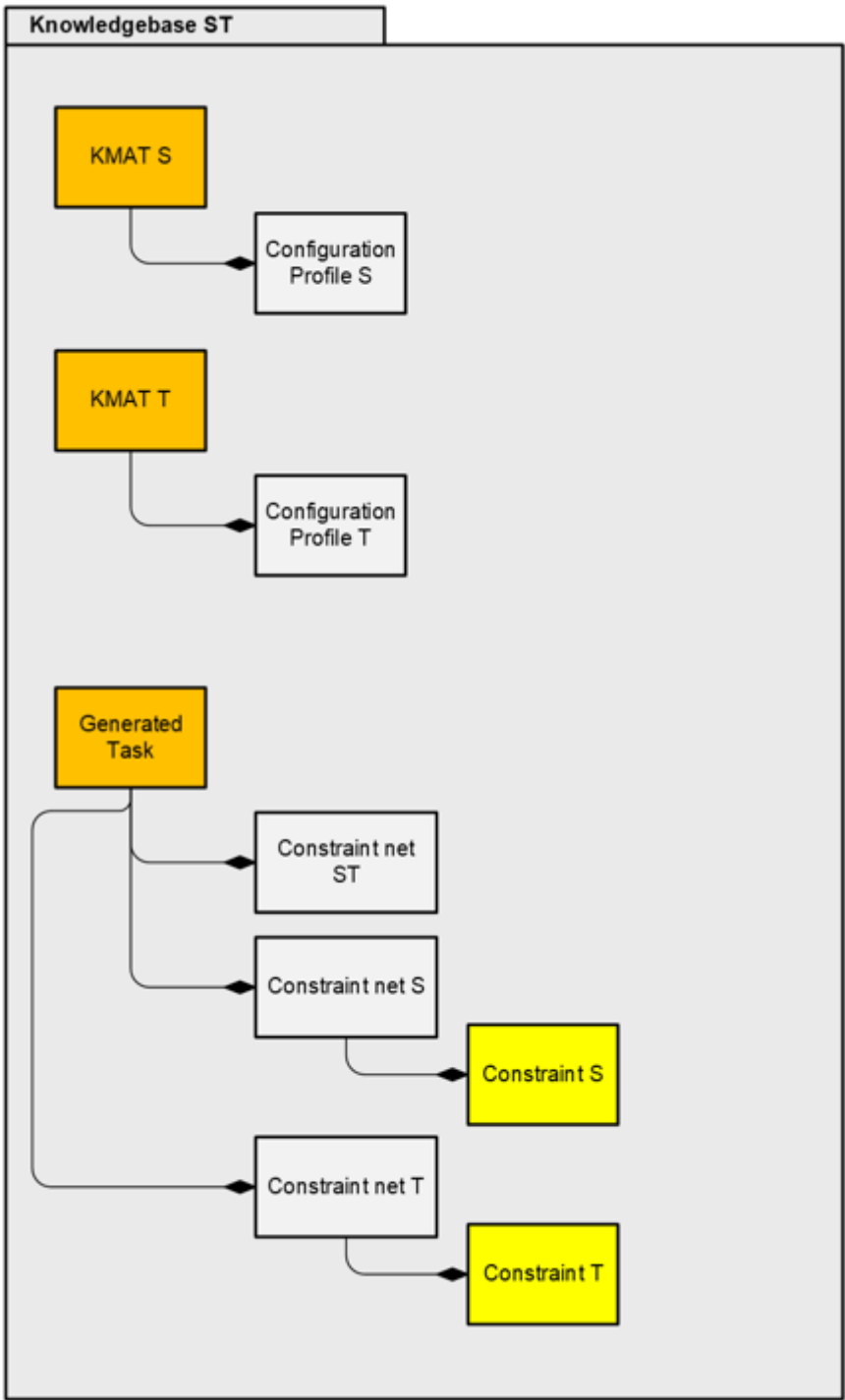
The following picture shows that this leads to the commonly used constraint net: Constraint ST. Specific constraint net for product S: Constraint net S. And specific constraint net for product T: Constraint net T.



Combining both products in one knowledgebase is a good approach from the data-redundancy point of view. However, there is one particularity to be noted which is also documented in the list of deltas between the VC and RTV usage in the Configuration Engine. For example, in SAP Variant Configuration and Pricing service:

Description	Configuration Engine	LO-VC
Loading constraint nets	All constraint nets are loaded at the start of the configuration process.	Constraint nets that are assigned to a configuration profile are loaded only if the material was selected during the configuration.

This is true because in the Knowledgebase Runtime-Version the constraint nets are collected in a so-called task without the previous link to the Configuration Profile as shown in the picture below. Thus, all constraints nets and all constraints are loaded into the configuration from the generated task for any product which is configured.



Example and solution:

If both products are maintained to be configured with the same knowledgebase *ST*, then all maintained constraint nets (constraint nets which are assigned to any contained configuration profile) are loaded into the configuration during configuration start, whether it is for product *S*, or for product *T*. Thus, constraint net *S*, *T*, and *ST* are all three loaded when a configuration is created!

If the constraint *S* makes inferences for product *S* only, you must make sure to use a material specific object declaration in constraint *S*:

```
?0 IS_OBJECT(MATERIAL)(300 )(NR='S') WHERE  
C2 = Char12;  
C3 = Char13
```

The same applies for the declaration in constraint *T*:

```
?0 IS_OBJECT(MATERIAL)(300 )(NR='T') WHERE  
C2 = Char12;  
C3 = Char13
```

whereas the constraints in constraint net *ST* should use a declaration on class level, as they apply to both products *S* and *T*:

```
?0 IS_a(300 )CLASS_C WHERE  
C2 = Char12;  
C3 = Char13
```

i Note

Products can be grouped by *product family* if you can assign all used constraint nets to all relevant configuration profiles. Common constraints should be collected in a common net, and specific constraints should be collected in product specific nets using the material specific declaration.

5.8 Summary

The following summarizes the main considerations for defining knowledge bases. Your weighting of these various factors will influence your optimal KB definitions.

1. Redundancy avoidance in KBs
 - Identify high commonality among KMAT data, such as:
 - Variant tables and constraint nets that contain large amounts of data
 - Class hierarchies with potentially many preconditions for characteristics and characteristic values
 - Identify KMAT subassemblies that can also be root item for configurations
 - Avoid creating separate KBs for such KMATs
2. Ongoing maintenance of runtime versions
 - Ability to train support staff to maintain properly
 - Which runtime versions to generate after a change?
 - Overall effort
 - How many runtime versions to generate after a VC change?
 - Overall duration
 - How long does it take to regenerate and move runtime versions?
3. Configuration load time
 - How much data is there in the runtime version?

- How much memory is consumed?
4. Constraint net loading, as all constraint nets, is loaded at the start of a configuration, as well as the constraint nets from other configuration profiles!
 5. Feasibility of rapid trial and error testing.

6 Runtime Version Generation

This section provides guidance on when and how to regenerate RTVs. For this document, it is assumed that RTVs are generated in your production ERP system which is the most common scenario.

6.1 Number of RTVs Needed for a KB

As discussed in the Naming Convention section, separate RTVs are required for each combination of the following parameters:

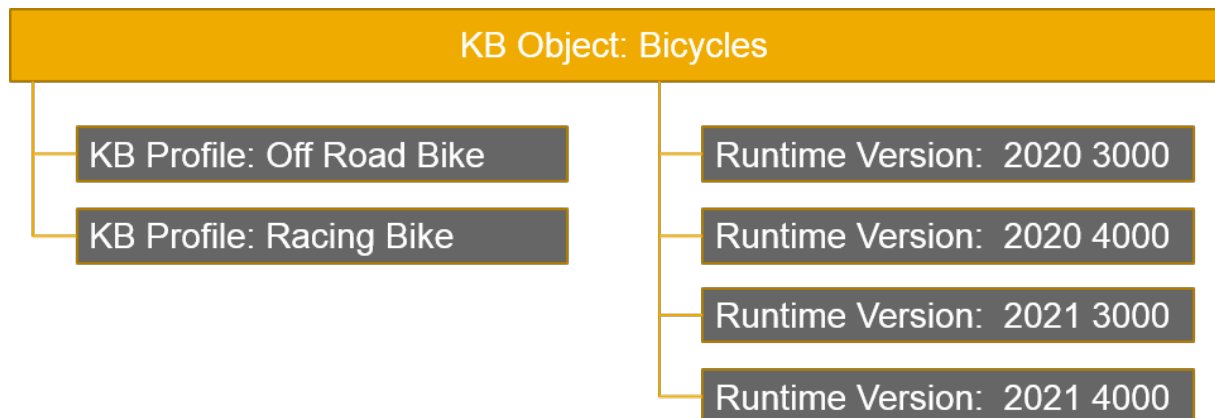
- *Valid-from date*
- *Plant*
- *BOM Application*
- *Language Key*
- *Configuration Profile*

In other words, the number of required RTVs is the multiplication of distinct (and only applicable) numbers of valid-from dates, plants, BOM applications, language keys, and configuration profiles:

"# of RTVs"="valid-from dates"×"plants"×"BOM appl."×"language keys"×"configuration profiles"

In the practice, the *valid-from date* is typically the only parameter that distinguishes from one RTV to another.

Continuing with the naming convention example, the bicycles had two separate *valid-from dates* for two separate plants, which have different configuration logic each, leading to four RTVs. BOM application, language keys, and configuration profiles are already distinct in this example and thus not applicable.



6.2 When to Generate New RTVs

Continuing the example of the previous page, two additional RTVs will be needed in 2022:

- [2022 3000](#) for plant 3000
- [2022 4000](#) for plant 4000

Additional RTVs are needed because it is necessary to maintain separate offerings for each model year, especially at yearend when it may be necessary to configure according to previous or next year offerings.

With SAP Note [2676709](#), a new RTV that contains the exact same settings and contents as its predecessor will not be saved. The predecessor is determined via the valid-from date.

In the above example, if a new RTV is generated using plant 4000 and:

- The new *valid-from date* is in 2022, the predecessor version will be RTV 2021 4000
- The new *valid-from date* lies between the one of RTV 2020 4000 and RTV 2021 4000, the predecessor version will be RTV 2020 4000
- The new *valid-from date* lies in 2019, no predecessor exists

6.3 When to Regenerate Existing RTVs

RTVs are typically regenerated (i.e., rebuilt) whenever changes need to be immediate and retroactive to existing configurations for that RTV. There are typically two scenarios for such updates:

- When there are corrections to the VC model data or logic, because it was previously incorrect
- When the midstream product changes have certain features that are immediately available or discontinued

Continuing with the previous bicycle example, a certain handlebar configuration becomes immediately discontinued due to a quality problem. The option is excluded in the VC models and the RTVs for that model year are regenerated and deployed. Existing quotations where that handlebar was allowed will now be inconsistent during reconfiguration or order submission which is ofcourse the desired outcome. The user would be alerted that a different handlebar configuration must now be selected.

6.4 RTVs Aligned to ECM Change Numbers

As mentioned in the Naming Convention section, RTV *valid-from dates* can be aligned with the *valid-from dates* of the ECM Change Number (ECN) when ECNs are used for maintaining the corresponding VC models. Note that RTVs themselves cannot be maintained with ECNs, therefore RTV *valid-from dates* must be aligned manually and realigned manually, and then regenerated, after a date shift of the ECN.

Using this approach, you will need one or more RTVs (see [Number of RTVs Needed for a KB \[page 28\]](#)) for each Change Number that was used to maintain the VC models. This approach works well for scenarios where

you do periodic releases to your VC models, to introduce and discontinue certain features, and perform the maintenance across all relevant VC models with a single change number. You may also decide that the effective date for introduction of these changes needs to move slightly in response to market or production readiness. The Change Number ensures that the VC model changes are coordinated between your ERP system and your RTVs.

Note that you can use various Change Numbers on manufacturing BOMs, assuming that manufacturing BOMs are not included in your RTV, and this will not affect the Change Number that is used to align with the RTV validity. Also note that you cannot use two Change Numbers with the same *valid-from date* since the Configuration Service will not be able to select both RTVs on that date.

As a footnote, KB objects cannot be maintained with ECM even though a read only and blank *Change Number* field appears on the initial screens of the classic transactions *CU31-33*.

In general, it is recommended to use engineering change management, but it is particularly important to do it when using Variant Configuration service with AVC forwarding enabled. That is to ensure that the version of the product model processed by AVC is the same as the knowledge-base version that is used by Variant Configuration service.

Variant configuration service does not process AVC configuration results that contain objects like characteristics or products that are unknown to the knowledge base, because this would lead to errors in SAP CPQ.

6.5 Real World Considerations

The following table summarizes approaches and extremes for regenerating existing or generating new RTVs for a given KB. Expected behavior of reconfiguration (i.e., changing a saved configuration) use cases play a key factor in the decision process along with the frequency of VC model changes.

Frequency	Regenerate an Existing RTV	Generate a New RTV
Daily	Reasonable if all fixes should become effective immediately and retroactively	Bad practice unless daily changes must not affect reconfiguration use case
Never	No fixes made to previous RTVs; always reconfigure with current effective date	Reconfiguration always uses a single (i.e., latest) version of VC data and logic
After ALE to Production	Common practice, especially reusing a previous change number (or no ECM)	Common practice, especially when ALE and RTV aligned to new change number

The following are comparisons of approaches and extremes for regenerating existing RTVs for a given KB:

1. **Regenerate your RTVs daily:** This is a reasonable practice if you want to ensure that VC model changes, especially changes made directly in your production system, are properly reflected in your RTVs, i.e., changes should be effective immediately and retroactively. Note that a new build will not be saved if there is no difference with the current build.
2. **Never regenerate your RTVs:** This implies that you create a new RTV whenever you make VC model changes. This approach does not provide accurate reconfiguration when changes must be immediate and

retroactive. For example, a configuration is created with RTV1. The next day, a VC model issue is corrected and a new RTV2 is generated. Any new configurations or reconfigurations using RTV1 will still be incorrect

3. **Regenerate RTVs after ALE:** This is a common practice. It assumes that VC models in your production system are not changed except by ALE from a VC Gold Client or similar. Relevant RTVs should be regenerated immediately following the ALE, without a *Change Number* or reusing a previous *Change Number*, to ensure that your ERP VC Models and corresponding RTVs remain synchronized.

The following are comparisons of approaches and extremes for generating new RTVs for a given KB:

1. **Generate a New RTV Daily:** This is a bad practice unless you have daily differences in your underlying VC models and need your configurations to work differently for each configuration date. More likely, this approach will result in many RTVs with the same contents when there was no change in the underlying VC models from one day to the next. Such redundancy will eventually impact storage and performance. With SAP Note [2676709](#), a new RTV that contains the exact same settings and contents as its predecessor will not be saved.
2. **Never Generate New RTVs:** This implies that you regenerate your single RTV whenever you make VC model changes. In effect, all VC changes become effective immediately and retroactively. It will not be possible to reconfigure existing configurations with the same configuration logic that was valid when a configuration was created. All reconfiguration scenarios will use the current VC model logic.
3. **Generate New RTVs after ALE:** This is a common practice and similar to Regenerate RTVs after ALE (see above). The difference is that a new RTV should be generated following ALE with a new *Change Number* for the related VC model(s). Beware of using this approach without a *Change Number*; see the [System Landscape Considerations \[page 34\]](#) section for SAP ERP.

6.6 Classic and Advanced Knowledge Base Runtime Version Generation (KBGen)

Advanced knowledge base runtime version generation (KBGen) was introduced especially for [SAP Variant Configuration and Pricing](#) to provide solutions to various classic KBGen limitations. The prefix *advanced* refers to the literal meaning, it was not exclusively developed to support the Advanced Variant Configuration (AVC).

SAP Note [3022814](#) explains the differences between classic and advanced KBGen as well as general features, limitations, and instructions for use. The key improvements of the advanced KBGen are:

- Advanced runtime versions are not replicated to CRM systems via middleware, as the middleware is based on the *SCE** tables
- Runtime version data is saved directly to the *COMM_CFG** tables only, making any copy process unnecessary
- Runtime version data includes an additional identifier for BOM position, implemented by a new field in the DB table *COMM_CFGCLPART*
- Characteristics of new compressed storage of variant tables, generically in new DB tables *COMM_CFGVARVAL* and *COMM_CFGVTGVAL*:
 - Variant table cells with multiple values are no longer divided into separate rows during RTV generation, reducing the amount of processing time and data storage required for generated RTV variant tables
 - Numeric interval values
 - Wildcard values, for AVC variant tables only.

- Advanced RTVs cannot be processed by the IPC if they contain variant tables.
- Only one selection screen to pick all classic configuration profiles
- The RTV header table, DB table [COMM_CFGKB](#) now includes additional fields for:
 - BOM Application
 - Plant
 - Authorization Group
 - KB Schema, classic or advanced KBGen
 - Engine Processing Mode (e.g. AVC).

To distinguish between classic and advanced KBGen, a KB schema was introduced as part of the RTV header table. The classic KBGen sets the KB schema to the value [2](#) whereas the advanced KBGen sets it to [4](#). Note that it is not possible to change the KB schema of an existing RTV, all future builds of an RTV will continue to use the KB schema of the original RTV build following regeneration.

The engine processing mode is set according to the configuration profile used. All configuration profiles of a runtime version must have the same processing mode, it is not allowed to mix classic and AVC configuration profiles in the same RTV. If an AVC configuration profile exists and is released, it will be used automatically with the new KBGen.

Advanced KB Generation is available in S/4HANA and ECC EhP8 SP16 or higher. See SAP Note [3022814](#) for specific details about how to enable and use advanced KBGen by product version.

6.7 How to Switch the KB Generation Mode?

In package VSCE, add an implementation for method [GET_DEFAULT_KBSHEMA](#) of enhancement spot [KBSHEMA_CURT](#). This method has one export parameter [EV_KBSHEMA](#) which needs to return either the value [4](#) for advanced KB generation or value [2](#) for the classic KB generation.

Please note that only newly-generated KB runtime versions will take the changed schema into account. If previously generated runtime versions exist, the new build will consider the KB schema of the previous build. It is not possible to re-generate existing runtime versions with a different KB schema.

6.8 KB and RTV Maintenance Utilities

VC Modeling Environment, transaction PMEVC, provides a way to see all KB objects where a configurable material has a KB profile. It shows all RTVs for a given KB object and provides a way to generate a new RTV or regenerate an existing RTV.

Report [COM_CFG_DB_STAT](#) provides a detailed overview of the contents of an RTV. See examples of this report in Appendix A. Follow these steps to use the report:

1. Execute report [COM_CFG_DB_STAT](#)
2. Enter filter criteria for KB IDs if desired, then execute (button or F8)

3. See the matching KB IDs with their high-level statistics, note the abbreviations in the legend at the bottom
4. Double click on a KB ID to see the detailed contents of the RTV, including header data at the top
5. Double click on any object to see additional details about that object

The material table lists all materials in the RTV, and those with a KB profile show an *X* in the *Pro* column. This means that relevant subordinate KMATs are included regardless of whether the KB has a profile for those KMATs.

Transaction *EXPO_TEST* is useful to diagnose the framework of explosion (FOX) paths which is used for product data replication (PDR). It can also help to understand why certain materials and other objects are included, or not, in an RTV.

7 System Landscape Considerations

For integration testing, the mechanics of knowledge base objects and runtime versions (RTVs) are, for the most part, easily learned and tested with simple Variant Configuration (VC) models and various RTVs that illustrate various change states of a VC model.

7.1 Configuration Engine Versions

SAP continually updates the Variant Configuration service to improve performance and remove so called *deltas* to the LO-VC (SAPGUI) configurator.

The Configuration Engine as part of SAP_AP, aka IPC, is not updated with all the improvements from the Variant Configuration service. The innovation focus is clearly on the service side. As mentioned earlier, the advanced knowledge base generation, for example, is not supported by the Configuration Engine (IPC).

7.2 Filter Unnecessary KB RTVs

Classic KB RTVs are stored in two sets of database tables in ERP/S4H, namely *SCE* tables and *COMM_CFG* tables. Contents of the *SCE* tables must be copied to the *COMM_CFG* tables for use with the Variant Configuration service on the SAP Business Technology Platform (used by CPQ) or with the Configuration Engine in ERP. CRM middleware, for example, only uses data from the *SCE* tables and copies the data to the *COMM_CFG* tables in CRM to be used by the Configuration Engine.

All runtime version data in the *COMM_CFG* tables are sent to the Business Technology Platform if that integration is implemented. You can filter the runtime version data copied to the *COMM_CFG* tables as explained in SAP Note [3006114](#). This is useful to reduce communication with and storage in the Business Technology Platform to only those runtime versions that are needed in the Variant Configuration service. The note explains how to implement a BADI that contains your required filter logic.

When creating a filter logic, avoid filtering out future builds of an existing RTV, because the Business Technology Platform might continue using an out-of-date, previous build. You must also avoid filtering in a later build of a previously filtered out RTV, because the later build will be ignored if the previous builds of that RTV were filtered out.

7.3 SAP CPQ

CPQ can be integrated with SAP Variant Configuration and Pricing in the SAP Business Technology Platform. When there are multiple runtime versions for a KMAT, each runtime version will create a corresponding CPQ

Product Version when those runtime versions are synchronized to CPQ via the Variant Configuration service. The effective date of a CPQ quote is used to select the appropriate runtime version for a configuration based on the *valid-from date* of the runtime versions.

There are nuances to this integration regarding the maintenance of CPQ product data and the product catalog that are addressed in the CPQ Setup Guide. Note that to prevent a KMAT material from being used as a main item in CPQ, you can assign its CPQ product to a CPQ category where no end user has permissions.

7.4 SAP ERP

Because runtime versions are typically generated in SAP ERP, there is a common assumption that a variant configuration created outside of ERP using a runtime version will always be valid when sent to ERP via integration, e.g., submitting a CRM quote to become an ERP sales order. While good practices make this true, bad practices can result in failures. The following example explains how this integration can fail:

1. Create a KMAT in ERP
2. Create runtime version for that KMAT in ERP and load the runtime version into CRM
3. Make changes to that KMAT in ERP, disallowing a specific characteristic value
4. Create a quote in CRM selecting that specific characteristic value
5. Submit the CRM quote to become an ERP sales order
6. The integration fails because CRM allows the specific characteristic while ERP does not

This example illustrates how VC changes made in ERP are not known to runtime versions that were previously generated. One way to avoid the failure in step 6 is to perform step 3 with an ECM Change Number with a valid-from date that is later than the valid-from date of the runtime version. You must also ensure that the configuration date used in the ERP sales order matches that of the CRM runtime version, there are user exits in ERP that could have a different configuration date. Note that the same situation also applies for the cloud integrations.

The good practices also dictate that you create a new runtime version for the KMAT with a valid-from date in the future and equal to the valid-from date of the ECM Change Number. See [RTVs Aligned to ECM Change Numbers \[page 29\]](#) for additional details.

8 Advanced Use Cases

The following topics are about advanced use cases such as multiple configuration profiles, runtime version deletion and no BOM explosion.

8.1 Multiple Configuration Profiles

If you maintain multiple released configuration profiles for a KMAT, you may be prompted to select a configuration profile when RTVs are regenerated for a KB that contains that KMAT. The system will match the RTV BOM application with the configuration profile BOM explosion. For example, you have a KMAT with three released configuration profiles as follows:

- Profile A, BOM Application = SD01
- Profile B, BOM Application = SD01
- Profile C, BOM Application = PP01

If you are regenerating an RTV with BOM application *SD01* that includes this KMAT, you will be prompted to select either Profile *A* or *B*. If you are regenerating an RTV with BOM application *PP01* that includes this KMAT, Profile *C* will be selected automatically.

Your RTV name should distinguish which configuration profile is used if a manual selection is made at RTV regeneration. You will have to select that configuration profile each time you regenerate the RTV, assuming there continues to be more than one matching configuration profile for the RTV. Continuing the example above with the previously recommended naming convention, RTVs for each configuration profile for the year 2021 would be named as follows, respectively:

- 2021 SD01 A
- 2021 SD01 B
- 2021 PP01

8.2 Runtime Version Deletion

In principle, an RTV can be deleted using transactions *PMEVC* or *CU35* when all the following are true:

- New configurations are no longer allowed to use it and
- All existing configurations that used it have been archived along with the sales documents which are referring to the configurations

8.3 No BOM Explosion

Plant and BOM application are required fields in an RTV. If you want to create a runtime version without including manufacturing BOM explosion, use a BOM application where no BOM will be found to create a runtime version without a BOM. Keep in mind that the BOM application of the RTV and the BOM explosion of the configuration profile must match.

9 Conclusion

Like variant configuration (VC) itself, knowledge bases (KBs) are a very powerful and nuanced functionality. Successful implementations and maintenance can be ensured when you do the following:

- Follow the generally regarded best practices in VC modeling
- Understand the mechanics of how KB objects and runtime versions work and then define your knowledge bases in the best way to meet your implementation and maintenance objectives
- Develop and follow a KB runtime version generation policy and procedure
- Understand the mechanics of runtime version integration within your system landscape
- Ensure that your runtime versions stay in synchronization with the VC models from which they are generated

10 Appendix A – Report COM_CFG_DB_STAT SCREENSHOTS

The following screenshots show the output of report *COM_CFG_DB_STAT* for the Simplified Multi-Level Example in the knowledge base profiles section. Knowledge base object *XY* with version *1* can also be found in the ECC sandbox system of the SAP Configuration Workgroup.

The top section of the report output shows the classes and materials in the KB RTV. The *Pro* column indicates that a material or class has a KB profile in this KB object. The *Superclass* column indicates to which classes a material or class is allocated. In this example, there is a one-to-one relationship between configurable material and class. The bottom section of the report output shows the distinct characteristics from the list of classes.

Knowledgebase : XY		Version : 1		Valid from : 03/27/2021		ID : 566		Creation date : 03/27/2021		Created by:	
Build : 4		SCE mode : classic		Last change date : 03/27/2021		Languages : <all>		Changed by:			

Class	ID	Description	Clt	Pro	BOM	Superclass
X	2	Class X (Char11 - Char15)	300			
Y	4	Class Y (Char12 - Char16)	300			
Z	6	Class Z (Char16 - Char20)	300			

Material	ID	Description	Clt	Pro	BOM	Superclass
X	1	Material X	300	X	X	X
Y	3	Material Y	300	X		Y
Z	5	Material Z	300			Z

Characteristic	ID	Description	Typ	Cat	Len	Dc	Uni	Cas
CHAR11	1	Characteristic 11	sym	sv	1			
CHAR12	2	Characteristic 12	sym	sv	1			
CHAR13	3	Characteristic 13	sym	sv	1			
CHAR14	4	Characteristic 14	sym	sv	1			
CHAR15	5	Characteristic 15	sym	sv	1			
CHAR16	6	Characteristic 16	sym	sv	1			
CHAR17	7	Characteristic 17	sym	sv	1			
CHAR18	8	Characteristic 18	sym	sv	1			
CHAR19	9	Characteristic 19	sym	sv	1			
CHAR20	10	Characteristic 20	sym	sv	1			

If we compare the following KBs and RTVs with the one above, we will see that *KB X* and *KB Y* have identical contents except that *KB X* cannot be used to configure material *Y* as a root item because it has no KB profile for material *Y*. You will also see that the contents of *KB Y* are entirely contained within *KB XY* therefore, a separate *KB Y* would be technically redundant.

Knowledgebase : X
 Version : 1
 Valid from : 03/27/2021
 ID : 567
 Build : 4
 SCE mode : classic

Creation date : 03/27/2021
 Last change date : 03/27/2021
 Languages : <all>

Created by:
 Changed by:

Class	ID	Description	Clf	Pro	BOM	Superclass
X	2	Class X (Char11 - Char15)	300			
Y	4	Class Y (Char12 - Char16)	300			
Z	6	Class Z (Char16 - Char20)	300			

Material	ID	Description	Clf	Pro	BOM	Superclass
X	1	Material X	300	X	X	X
Y	3	Material Y	300			Y
Z	5	Material Z	300			Z

Characteristic	ID	Description	Typ	Cat	Len	Dc	Uni	Cas
CHAR11	1	Characteristic 11	sym	sv	1			
CHAR12	2	Characteristic 12	sym	sv	1			
CHAR13	3	Characteristic 13	sym	sv	1			
CHAR14	4	Characteristic 14	sym	sv	1			
CHAR15	5	Characteristic 15	sym	sv	1			
CHAR16	6	Characteristic 16	sym	sv	1			
CHAR17	7	Characteristic 17	sym	sv	1			
CHAR18	8	Characteristic 18	sym	sv	1			
CHAR19	9	Characteristic 19	sym	sv	1			
CHAR20	10	Characteristic 20	sym	sv	1			

Knowledgebase : Y
 Version : 1
 Valid from : 03/27/2021
 ID : 568
 Build : 3
 SCE mode : classic

Creation date : 03/27/2021
 Last change date : 03/27/2021
 Languages : <all>

Created by:
 Changed by:

Class	ID	Description	Clf	Pro	BOM	Superclass
Y	2	Class Y (Char12 - Char16)	300			

Material	ID	Description	Clf	Pro	BOM	Superclass
Y	1	Material Y	300	X		Y



Characteristic	ID	Description	Typ	Cat	Len	Dc	Uni	Cas
CHAR12	1	Characteristic 12	sym	sv	1			
CHAR13	2	Characteristic 13	sym	sv	1			
CHAR14	3	Characteristic 14	sym	sv	1			
CHAR15	4	Characteristic 15	sym	sv	1			
CHAR16	5	Characteristic 16	sym	sv	1			

Important Disclaimers and Legal Information

Hyperlinks

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

About the icons:

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

Videos Hosted on External Platforms

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

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

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

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

© 2023 SAP SE or an SAP affiliate company. All rights reserved.

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

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

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

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

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