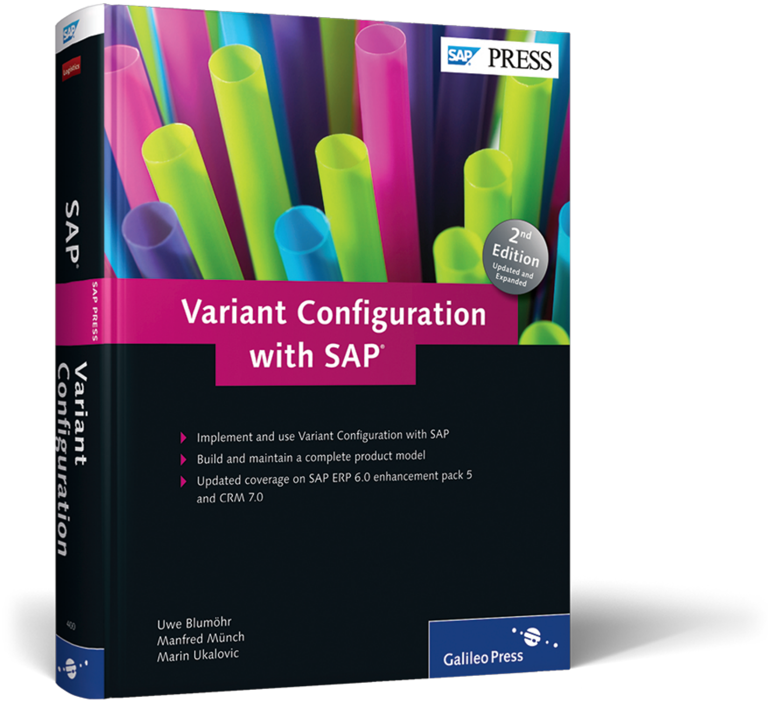
Uwe Blumöhr, Manfred Münch, and Marin Ukalovic

Variant Configuration with SAP®


SAP PRESS
2nd Edition Updated and Expanded
Variant Configuration with SAP®
Implement and use Variant Configuration with SAP
Build and maintain a complete product model
Updated coverage on SAP ERP 6.0 enhancement pack 5 and CRM 7.0
Uwe Blumöhr
Manfred Münch
Marin Ukalovic
Galileo Press
SAP
SAP PRESS
Variant Configuration

Galileo Press®

Bonn • Boston

Contents at a Glance

Contents

- A list of all BOM explosions including assigned object dependencies.
- An evaluation of the class nodes.
- A detailed list of the characteristic value assignment including assigned object dependencies for characteristics and values.
- An evaluation of the configuration profiles, including object dependencies.

2.6 Object Dependencies for the Value Assignment Interface or the Sales View

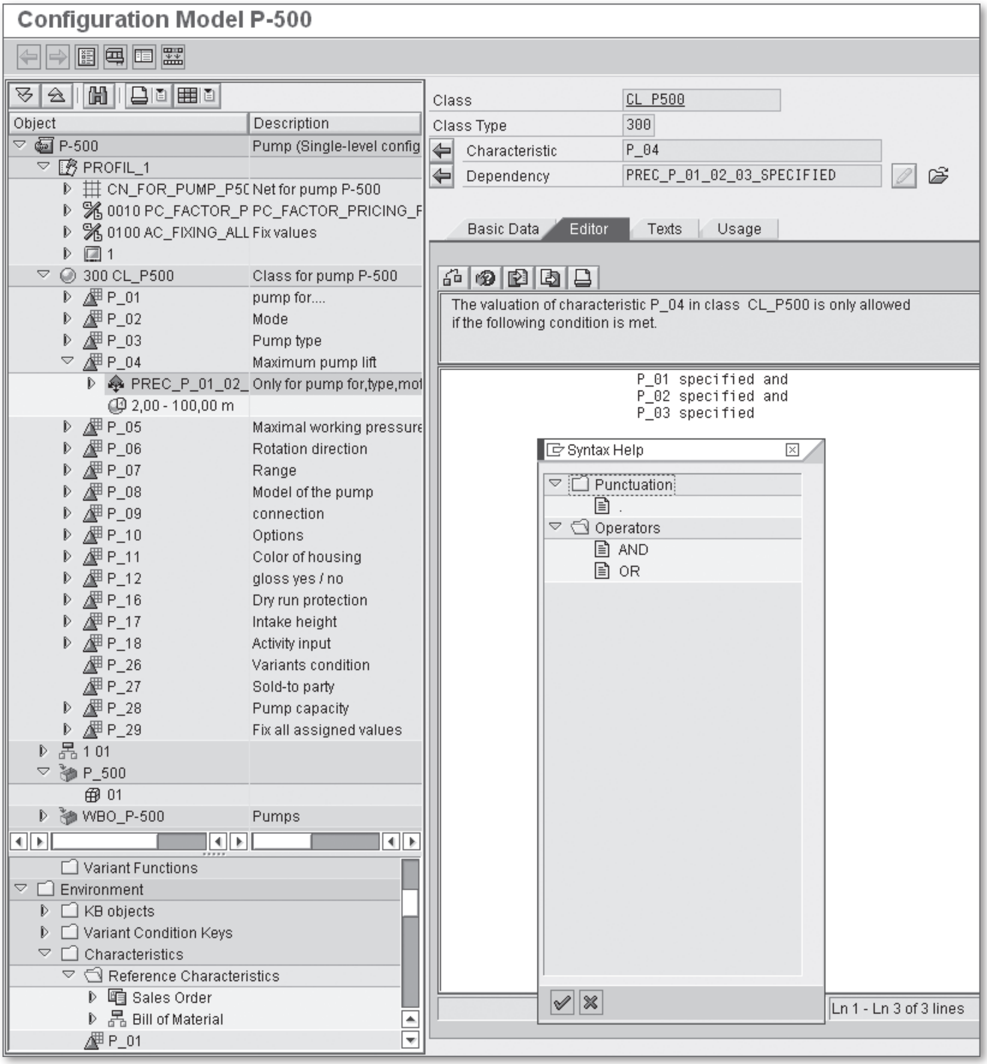
As already described, object dependencies are required for two usages: the high-level configuration (sales configuration, in the dialog, for the value assignment interface) and the low-level configuration (BOM and routing explosion, also without dialog). The following section discusses the first usage in more detail.

2.6.1 Product Modeling Environment PMEVC

Various transactions or methods are provided for the maintenance of object dependencies for the value assignment interface. The most important maintenance environment for object dependencies for the value assignment interface, both local and global, is the PMEVC product modeling environment (see Figure 2.29). This section therefore focuses on this environment. Section 2.7 introduces additional maintenance options.

The PMEVC product modeling environment has been available as Transaction PMEVC since SAP ERP Release 5.0. A similar function is also part of the IPC. PMEVC is short for *Product Modeling Environment Variant Configuration.* The concept behind this transaction is to create an environment in which you can maintain the entire variant model via the model structure from the high-level configuration perspective. Similar to the CUMODEL variant model browser, you can first obtain an overview of the existing model structure and then navigate to details.

You can also create and change numerous components of the configuration model from this product modeling environment. This enables you to create and change all types of object dependencies, both global and local. The same applies to configuration profiles, variant tables, and IPC data.



Configuration Model P-500

Object	Description
P-500	Pump (Single-level config
PROFIL_1	
CN_FOR_PUMP_P50	Net for pump P-500
0010 PC_FACTOR_P	PC_FACTOR_PRICING_F
0100 AC_FIXING_ALL	Fix values
1	
300 CL_P500	Class for pump P-500
P_01	pump for....
P_02	Mode
P_03	Pump type
P_04	Maximum pump lift
PREC_P_01_02_	Only for pump for,type,mot
2,00 - 100,00 m	
P_05	Maximal working pressure
P_06	Rotation direction
P_07	Range
P_08	Model of the pump
P_09	connection
P_10	Options
P_11	Color of housing
P_12	gloss yes / no
P_16	Dry run protection
P_17	Intake height
P_18	Activity input
P_26	Variants condition
P_27	Sold-to party
P_28	Pump capacity
P_29	Fix all assigned values
1 01	
P_500	
01	
WBO_P-500	Pumps

Variant Functions
Environment
KB objects
Variant Condition Keys
Characteristics
Reference Characteristics
Sales Order
Bill of Material
P_01

Class	CL_P500
Class Type	300
Characteristic	P_04
Dependency	PREC_P_01_02_03_SPECIFIED

Basic Data | Editor | Texts | Usage

The valuation of characteristic P_04 in class CL_P500 is only allowed if the following condition is met.

```
P_01 specified and
P_02 specified and
P_03 specified
```

Syntax Help
- Punctuation
  - .
- Operators
  - AND
  - OR

Ln 1 - Ln 3 of 3 lines

Figure 2.29 Product Modeling Environment PMEVC

The product modeling environment uses an additional editor for the maintenance of object dependencies. In contrast to the traditional object dependency editor, this editor allows you to use the following elements:

- Context-sensitive input help
- Drag-and-drop

- Object dependency wizard for preconditions, selection conditions, and table-based constraints

You can call the context-sensitive input help via the function key F4 or the second button in the editor (see Figure 2.29). All types of syntax elements as well as characteristics and characteristic values are provided. You can insert them in their respective positions in the object dependency syntax.

The drag-and-drop function enables you to copy individual characteristic values from the lists on the left (see Figure 2.29) to the editor in PMEVC. As the example in Listing 2.7 shows, the characteristic and the characteristic value are transferred in the form of an equation.

```
char1 = 'value1'
```

Listing 2.7 Example of a Syntax Generated via Drag-and-Drop

As already mentioned, the object dependency wizard enables you to create preconditions and selection conditions as well as table-based constraints without having to write the syntax yourself. The wizard queries all necessary information, and the system creates the syntax and all other required data.

The PMEVC product modeling environment not only enables you to completely maintain all object dependencies that are relevant for the value assignment interface; PMEVC also allows for nearly all maintenance steps of the modeling for the high-level configuration. This includes the following aspects:

- Maintenance (creation, modification, display, assignment) of object dependencies for the configuration profile, characteristics, and characteristic values
- Class-specific characteristic adaptation
- Simple classification (no multiple classification)
- Creation of a configuration profile (several profiles for one material master are not possible, but you can maintain all existing configuration profiles)
- Usage of change numbers
- Maintenance of the structure of variant tables
- Maintenance of the content of variant tables
- Modification of object dependencies for the BOM
- Creation of the knowledge base and runtime version for the IPC

- Material-specific activation of the IPC as the configurator in SAP ERP
- Maintenance of the user interface design
- Maintenance and assignment of variant conditions for pricing

Enhancement Package 5 (EHP5) of ERP Release 6.0 is likely to provide the additional option to not only change existing object dependencies in the BOM but also create new object dependencies. You can also use drag-and-drop.

The system functions are also supposed to be enhanced. If you start the simulation from PMEVC (button in Figure 2.29), the system first displays a screen for assigning values to reference characteristics. Depending on the configuration scenario—that is, whether it is a sales order or material variant scenario—you can assign values to all relevant reference characteristics before the actual value assignment screen opens.

Besides these two aspects (creation of new object dependencies for BOM items and simulation with reference characteristics), EHP5 will probably provide the following additional PMEVC functions:

- Creation of new characteristic values
- Creation of new descriptions and long texts in additional languages for characteristics and characteristic values
- Easier maintenance of characteristics groups within the scope of user interface design, including drag-and-drop option
- Detailed view for BOMs and BOM items

Most of the master data of the Variant Configuration model cannot be maintained via PMEVC. Consequently, it is required that the following master data be created via the common transactions in advance:

- Characteristics
- Classes
- Material masters
- BOMs including all objects at the item level
- Routings including all objects at the operation level
- Variant functions

- Change numbers
- Objects that couldn't be created in PMEVC before Release ERP 6.0

After this introduction of PMEVC, the following section describes an example of its usage.

2.6.2 Example

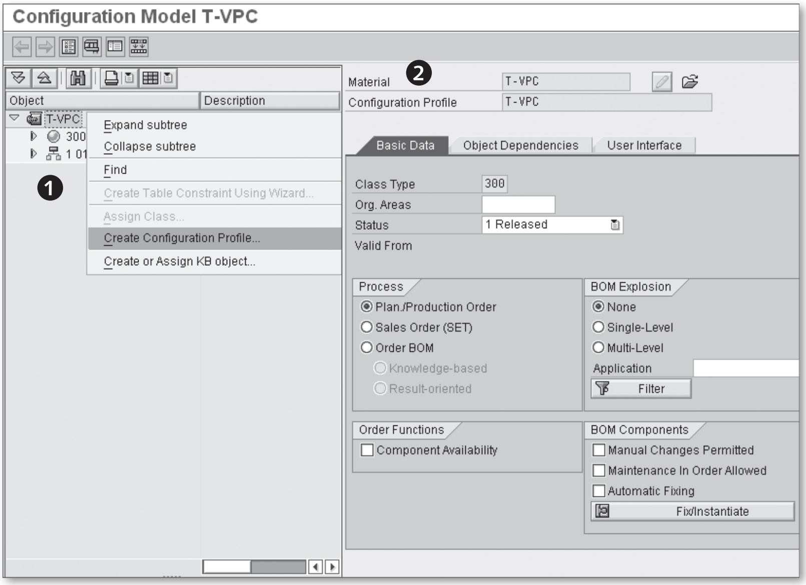
After these rather theoretical explanations, it may be helpful to provide you with a practical example of how you can use PMEVC to create object dependencies for the value assignment interface. For this purpose, you use the object dependency wizard in PMEVC.

To map dependencies between the individual characteristics of the value assignment interface, you should use tables or, if they don't become too long, variant tables. The advantage of tables is that you can read the dependency type from them more easily than when evaluating the syntax of the object dependencies directly. Furthermore, the usage of variant tables has a major advantage if the model is "alive": If the dependencies in the model change, you have to change only the content of the variant table, without having to modify the syntax of the object dependencies.

The easiest way to evaluate the table is to write object dependencies that query the table and only allow for value assignments that comply with the table. This is to be implemented in such a way that no disallowed value assignments are possible. The list of the allowed values for each characteristic is supposed to be dynamically restricted in such a way that only allowed value assignments are possible. For the selection of such "elegant" object dependencies, you must use a type for which the user doesn't have to specify a point in time when the object dependencies are to be processed. You should therefore use constraints.

In Figure 2.30, PMEVC was called with material T-VPC. The system has found the BOM for the material. These are initially the two only entries in structure (❶). You now require at least the variant class and the configuration profile. As already mentioned, the variant class, including its characteristics, needs to be created outside PMEVC. In Figure 2.30 the existing variant class is already assigned.

This variant class (or a complete group of variant classes) was previously included in the PMEVC environment (see the bottom left of Figure 2.31, here under ENVIRONMENT • CLASSES AND CONTEXT menu). You then simply assign the variant class via drag-and-drop.



Configuration Model T-VPC

Material T-VPC

Configuration Profile T-VPC

Basic Data | Object Dependencies | User Interface

Class Type 300

Org. Areas

Status 1 Released

Valid From

Process

● Plan./Production Order
○ Sales Order (SET)
○ Order BOM
○ Knowledge-based
○ Result-oriented

BOM Explosion

● None
○ Single-Level
○ Multi-Level

Application

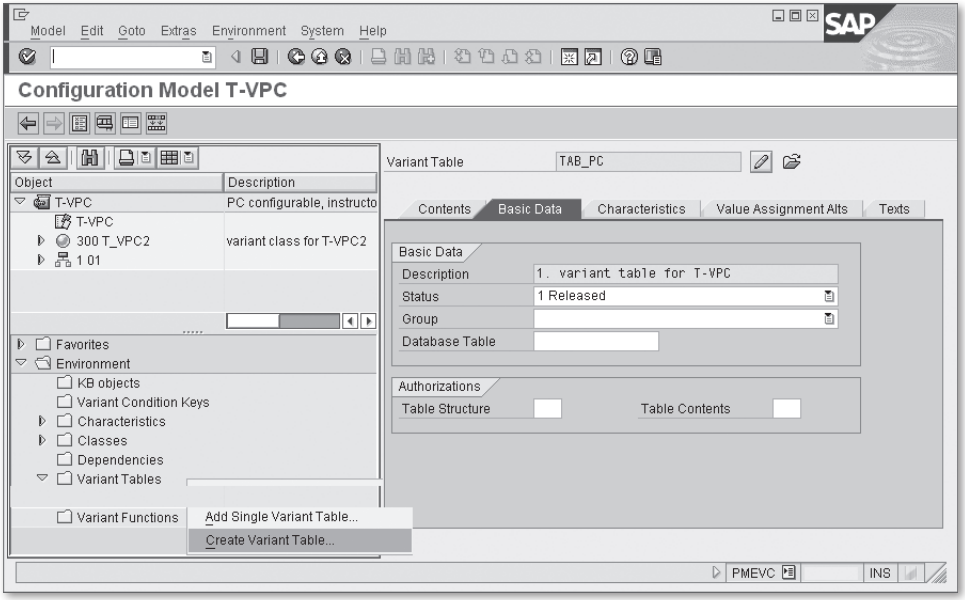
Order Functions

□ Component Availability

BOM Components

□ Manual Changes Permitted
□ Maintenance In Order Allowed
□ Automatic Fixing

Figure 2.30 Getting Started in PMEVC: Class Assignment and Creation of a Configuration Profile



Configuration Model T-VPC

Variant Table TAB_PC

Contents | Basic Data | Characteristics | Value Assignment Alts | Texts

Basic Data

Description 1. variant table for T-VPC

Status 1 Released

Group

Database Table

Authorizations

Table Structure □ Table Contents □

Object	Description
T-VPC	PC configurable, instructo
T-VPC	
300 T_VPC2	variant class for T-VPC2
1 01	

Figure 2.31 Variant Tables—Creation and Content Maintenance with PMEVC

In contrast to the variant class, you can create the configuration profile directly from PMEVC. As you can see in Figure 2.30 (❷), this function is available in the context menu at the material master level. Similar to the common transaction for the creation of configuration profiles, you can create a configuration profile with the default values. Unlike the common transaction, there are also default values for the name and variant class type.

In addition to the two previously mentioned steps, the model structure consists of four objects: material master, configuration profile, variant class, and BOM.

A variant table (see Figure 2.31) is supposed to map the allowed combinations for the value assignment of numerous characteristics. In this example, we assume that this includes the three characteristics: “Special wish,” “Casing,” and “CPU.” For this purpose, perform the following steps:

1. **Create a variant table**
   First you create the variant table via the context menu in the environment, because it isn't provided here yet.
2. **Name of the table**
   The window Create Variant Table opens, in which you enter a name for the table. Engineering Change Management is optional and not supposed to be used in this example.
3. **Description of the table**
   The system then displays the detail screen (see Figure 2.31) with five tabs. In the Basic Data tab, enter a description (language-dependent), and then release the variant table.
4. **Assigning characteristics**
   In the Characteristics tab, specify the three mentioned characteristics in any sequence.
5. **Entering the table content**
   Finally, enter the allowed combinations of the value assignment with regard to the three characteristics as rows in the Contents tab.

After the variant tables have been created and their content has been maintained, you require object dependencies. Object dependencies read the table and dynamically restrict the value lists of the corresponding characteristics in such a way that only value assignments from the table are possible. For this purpose, start the table constraint wizard via the context menu for the configuration profile, as shown in Figure 2.32.

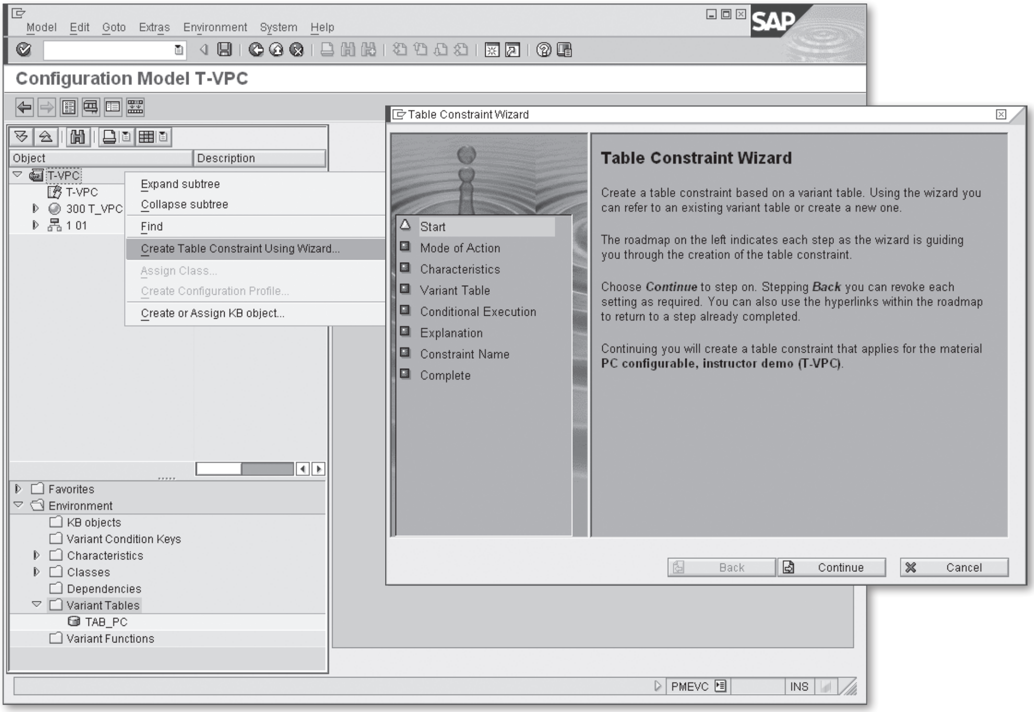


Table Constraint Wizard

Create a table constraint based on a variant table. Using the wizard you can refer to an existing variant table or create a new one.

The roadmap on the left indicates each step as the wizard is guiding you through the creation of the table constraint.

Choose *Continue* to step on. Stepping *Back* you can revoke each setting as required. You can also use the hyperlinks within the roadmap to return to a step already completed.

Continuing you will create a table constraint that applies for the material **PC configurable, instructor demo (T-VPC)**.

Figure 2.32 Creating Object Dependencies Using the Table Constraint Wizard

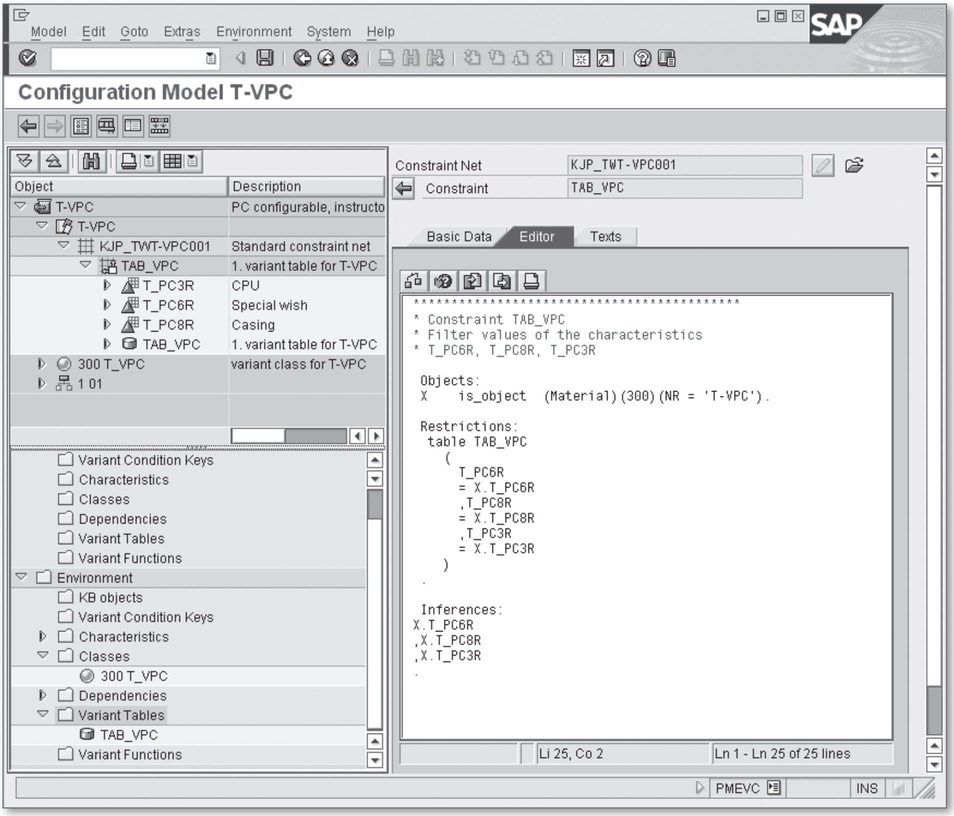
The wizard guides you through the individual steps of the creation of constraints and queries you for all required information. It is possible that the wizard may dynamically adapt the steps to the already specified information. The wizard processes the following steps:

1. **Start**
   The first step of the wizard provides information on the procedure for the creation of a constraint with reference to a variant table.

2. **Mode of Action**
   The second step queries you about the mode of action. Here, you're supposed to restrict the value lists for characteristics. This is the first selection option in this step. The difference from other selection options is described in the following.

   It is possible that the wizard won't provide the VALUE RESTRICTION option. In this case, the required prerequisite—that the characteristics whose value lists are supposed to be restricted are indicated as restrictable in the characteristic definition—is not met.
3. **Configuration Object**
   Next, the wizard queries you for the configuration object. In addition to characteristics from classes, you can also use and evaluate other reference objects in constraints. Because this example is supposed to restrict characteristics of a variant class, the corresponding variant class is also the configuration or reference object.
4. **Variant table**
   After the configuration object step, the variant table selection is implemented. Because this example contains only one variant table in the PMEVC environment, no comprehensive selection can be made.
5. **Explanation**
   In the EXPLANATION step, you can assign a language-dependent long text to the object dependencies, that is, to the table constraint. It also lists the characteristics of the variant table to exclude characteristics for object dependencies if required.
6. **Constraint Name**
   For constraints, the name must be assigned externally, that is, by the user. This request, including a short text, is made in the CONSTRAINT NAME step.
7. **Complete**
   All steps are completed. A description of these steps is provided once again in detail before the user can complete the table constraint.

The system then displays the completed table constraint (see Figure 2.33). After saving, you can directly test it by calling the configuration simulation from PMEVC via the TEST button.


Model Edit Goto Extras Environment System Help
SAP
Configuration Model T-VPC
Basic Data
Editor
Texts
PMEVC
INS

Constraint Net KJP_TWT-VPC001
Constraint TAB_VPC

Object	Description
T-VPC	PC configurable, instructo
T-VPC	
KJP_TWT-VPC001	Standard constraint net
TAB_VPC	1. variant table for T-VPC
T_PC3R	CPU
T_PC6R	Special wish
T_PC8R	Casing
TAB_VPC	1. variant table for T-VPC
300 T_VPC	variant class for T-VPC
1 01	

Variant Condition Keys
Characteristics
Classes
Dependencies
Variant Tables
Variant Functions
Environment
KB objects
Variant Condition Keys
Characteristics
Classes
300 T_VPC
Dependencies
Variant Tables
TAB_VPC
Variant Functions

```
********************************************
* Constraint TAB_VPC
* Filter values of the characteristics
* T_PC6R, T_PC8R, T_PC3R

 Objects:
 X    is_object  (Material)(300)(NR = 'T-VPC').

 Restrictions:
  table TAB_VPC
    (
      T_PC6R
      = X.T_PC6R
      ,T_PC8R
      = X.T_PC8R
      ,T_PC3R
      = X.T_PC3R
    )
  .

 Inferences:
X.T_PC6R
,X.T_PC8R
,X.T_PC3R
.
```

Li 25, Co 2 | Ln 1 - Ln 25 of 25 lines

Figure 2.33 Result of the Table Constraint Wizard

The value assignment interface displays the three characteristics: "Special wish," "Casing," and "CPU." Initially, the corresponding input help (F4 key) provides all values from the variant table for each characteristic. However, if you start assigning values to any of the three characteristics, the system provides the values only for the other two characteristics that lead to an allowed value assignment according to the variant table. The same applies to two characteristics to which values have been assigned.

A special feature of the traditional configurator is that if the allowed value range for a characteristic is restricted to a value, the system automatically sets this value. This is done only for required characteristics in the IPC. Note that the characteristics need to be restrictable according to the characteristic definition. As a result, the lists of the allowed values for characteristics to which values have been assigned are restricted to the value assignment, as already described.

2.6.3 Variant Tables in Detail

The first usage of variant tables was introduced in the example above. You can address variant tables in all types of object dependencies. The columns of variant tables are always characteristics. The rows represent value assignment combinations. You can use variant tables for different purposes:

- Value restrictions in constraints (in this context, values can already have been assigned—as described in the example)
- Inferences of values in constraints or procedures
- Conditions as preconditions, selection conditions, if conditions in procedures or constraints, and as a condition part in constraints
- Consistency checks via constraints

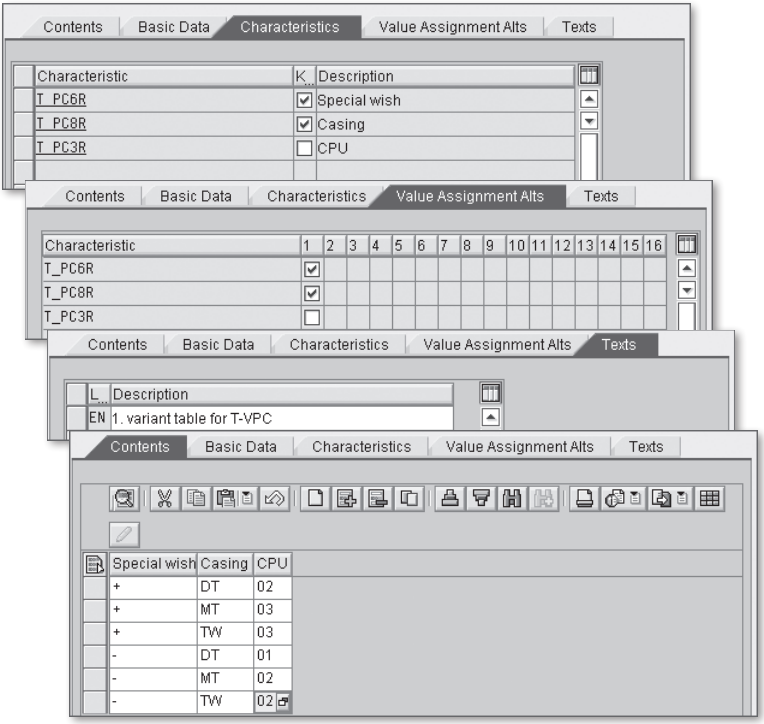
These purposes are discussed in more detail in the introduction to the corresponding types of object dependencies in Sections 2.6.4 through 2.6.7.

At this point, the structure and maintenance of variant tables will be introduced (see Figure 2.31 and Figure 2.34). You can maintain the variant table's structure and content via PMEVC. There are also specific transactions for the following tasks:

- Content maintenance (Transaction CU60)
- Creating, changing, and displaying the table structure (Transactions CU61 through 63)

The BASIC DATA tab of variant tables (see Figure 2.31) contains names, the description (language-dependent short text), the status, and the group. The status can be adapted via Customizing and also includes the content maintenance and usage in object dependencies as well as distribution locks for the content and structure.

The same aspects apply to groups as to characteristic and class groups; it is also a separate list in Customizing. Furthermore, you can couple the variant table with a database table in the basic data. This is described in detail later in this section. The basic data is complemented by the authorization groups for the content and structure maintenance.



Characteristic	K..	Description
T_PC6R	☑	Special wish
T_PC8R	☑	Casing
T_PC3R	☐	CPU

Characteristic	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
T_PC6R	☑															
T_PC8R	☑															
T_PC3R	☐															

L..	Description
EN	1. variant table for T-VPC

Special wish	Casing	CPU
+	DT	02
+	MT	03
+	TW	03
-	DT	01
-	MT	02
-	TW	02

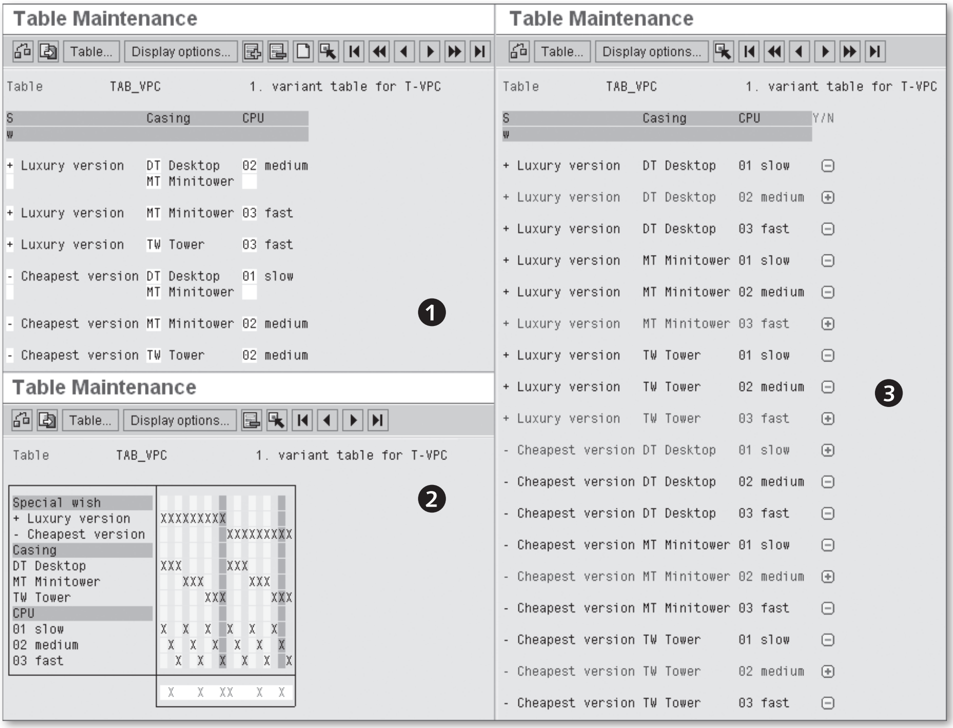
Figure 2.34 Structure and Maintenance of the Variant Table (PMEVC)

The Characteristics tab, which is assigned to a variant table, provides the columns of the variant table. For the maintenance of the table content and for the usage in object dependencies, the settings in the characteristic definition—such as single-level/multilevel, restrictable, required characteristic, default values, or object dependencies—are irrelevant. In the characteristic view, you can define a first key. This key is merely a prerequisite and is significant only if you want to infer values from the variant table. This can be done via constraints or procedures. For value restrictions, conditions, or mere consistency checks, the system ignores the key information.

For inference of values, constraints with an inference part can evaluate more than the Value Assignment Alternative (key in the Characteristics tab) that is specified in the tab. You can create these additional alternatives in the corresponding view.

In PMEVC, you can also maintain the elements of the Content directly in the variant table.

Besides maintaining content from PMEVC, you can implement the content via a common transaction, namely, Transaction CU60. In addition to the standard display (❶ in Figure 2.35), this transaction also allows for displays as a matrix (❷) and as a list (❸). These last two displays enable you to easily decide which combination is supposed to be used (decision table).



```
Table         TAB_VPC            1. variant table for T-VPC

S                   Casing       CPU
W

+ Luxury version    DT Desktop   02 medium
                    MT Minitower

+ Luxury version    MT Minitower 03 fast

+ Luxury version    TW Tower     03 fast

- Cheapest version  DT Desktop   01 slow
                    MT Minitower

- Cheapest version  MT Minitower 02 medium

- Cheapest version  TW Tower     02 medium
```

```
Table         TAB_VPC            1. variant table for T-VPC

Special wish
+ Luxury version     XXXXXXXXX
- Cheapest version            XXXXXXXXX
Casing
DT Desktop           XXX      XXX
MT Minitower            XXX      XXX
TW Tower                   XXX      XXX
CPU
01 slow              X  X  X  X  X  X
02 medium             X  X  X  X  X  X
03 fast                X  X  X  X  X  X

                      X   X  XX   X  X
```

```
Table         TAB_VPC            1. variant table for T-VPC

S                   Casing       CPU        Y/N
W

+ Luxury version    DT Desktop   01 slow     ⊖
+ Luxury version    DT Desktop   02 medium   ⊕
+ Luxury version    DT Desktop   03 fast     ⊖
+ Luxury version    MT Minitower 01 slow     ⊖
+ Luxury version    MT Minitower 02 medium   ⊖
+ Luxury version    MT Minitower 03 fast     ⊕
+ Luxury version    TW Tower     01 slow     ⊖
+ Luxury version    TW Tower     02 medium   ⊖
+ Luxury version    TW Tower     03 fast     ⊕
- Cheapest version  DT Desktop   01 slow     ⊕
- Cheapest version  DT Desktop   02 medium   ⊖
- Cheapest version  DT Desktop   03 fast     ⊖
- Cheapest version  MT Minitower 01 slow     ⊖
- Cheapest version  MT Minitower 02 medium   ⊕
- Cheapest version  MT Minitower 03 fast     ⊖
- Cheapest version  TW Tower     01 slow     ⊖
- Cheapest version  TW Tower     02 medium   ⊕
- Cheapest version  TW Tower     03 fast     ⊖
```

Figure 2.35 Variant Tables—Content Maintenance with the Common Transaction

[!]

IPC-Compatible Table Content

You can also select multiple values for each field in the standard display. This display isn't IPC-compatible, but you can transfer it to such a display using the respective menu entry (Add up, Untag).

Besides the already introduced options of maintaining variant table content using PMEVC or a common transaction, namely, Transaction CL60, you can also import content from Excel tables. For this purpose, you can use Transaction CU60E.

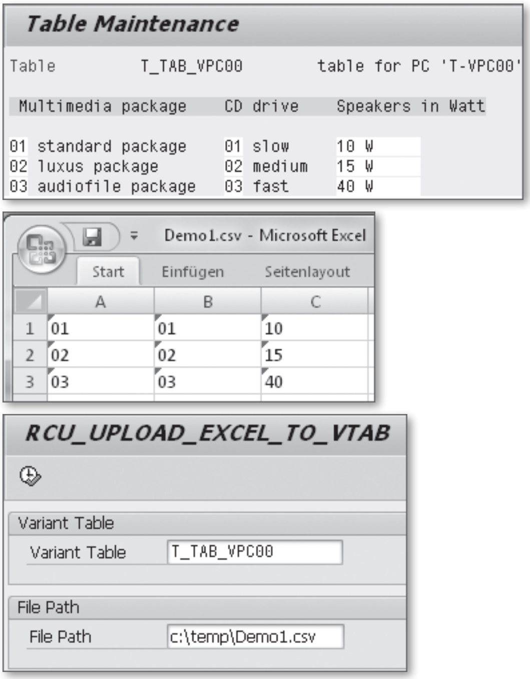


Table Maintenance

Table T_TAB_VPC00 table for PC 'T-VPC00'

Multimedia package	CD drive	Speakers in Watt
01 standard package	01 slow	10 W
02 luxus package	02 medium	15 W
03 audiofile package	03 fast	40 W

	A	B	C
1	01	01	10
2	02	02	15
3	03	03	40

RCU_UPLOAD_EXCEL_TO_VTAB

Variant Table

Variant Table: T_TAB_VPC00

File Path

File Path: c:\temp\Demo1.csv

Figure 2.36 Transaction CU60E—Data Import from Excel Tables into Variant Table

To import data using Transaction CU60E, the SAP ERP system requires an existing variant table. This means that you first have to create the required characteristics and then use these characteristics to create the variant table structure.

The example in Figure 2.36 assumes that the result of the data import is a variant table (as illustrated at the top of Figure 2.36). The specified preparations in the ERP system include the creation of the three characteristics that are used in this example (multimedia package, CD drive, and speakers) as well as the creation of the variant table structure (T_TAB_VPC00).

When creating and filling the Excel table, the following needs to be considered: The columns of the Excel table and their sequence must correspond to those of the variant table, and all values in the Excel table must be in the *text* format. This also applies to numerical characteristics in the variant table. The Excel table mustn't contain headers or descriptions—only language-independent characteristic values. The system checks the format of the values during the data import and cancels the transaction if necessary. The Excel file needs to be saved as a *csv* file (see Figure 2.36). This means that semicolons are used as separators for this Excel file format.

Transaction CU60E requires solely the name of the variant table as well as the name and address of the Excel file as specifications. You start the data import using the Execute button (F8). Only complete data imports are possible; an option for change uploads does not exist. The system deletes possibly existing variant table content and fills the variant table with the new entries.

[+]

Transaction CU60E

With regard to Transaction CU60E, please refer to SAP Note 516885. It also contains a documentation for this transaction.

As already mentioned, you can use variant tables in all types of object dependencies. Basically, the syntax always looks the same (see Listing 2.8).

```
table <table name>
(<column characteristic>=<interface characteristic>,...)
```

Listing 2.8 Syntax Concept for Calling Variant Tables

You can break down this visual similarity as follows:

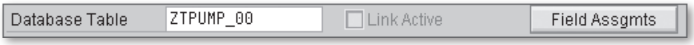
- The call starts with the keyword `table`.
- Then the name of the table is specified.
- Next, all columns of the variant table that are to be evaluated are listed in brackets. You don't always have to evaluate the entire table. Because the characteristics of the table don't have to correspond to the characteristics of the user interface, the system assigns the corresponding characteristic to the value assignment interface after each column characteristic.

Figure 2.33 shows an example of such a table call in constraints. Note that `X.` precedes the characteristics from the value assignment interface. This refers to material T-VPC, as you can see in the constraint directly above the table call. As mentioned in Chapter 1, you cannot address objects (the material master in this case) as flexibly as in constraints. In this case, you can only use the `$self.`, `$root.`, or `$parent.` levels. The table access from Figure 2.33 would then be as shown in Listing 2.9.

```
table TAB_VPC (  T_PC6R = $self.T_PC6R,
                 T_PC6R = $self.T_PC6R,
                 T_PC6R = $self.T_PC6R)
```

Listing 2.9 Table Access in Preconditions, Selection Conditions, and Procedures

Characteristics of the value assignment interface for which the values are supposed to be inferred must be linked with `$self.`; $parent. and $root. are also allowed.



Database Table ZTPUMP_00 □ Link Active

Figure 2.37 Coupling of Variant and Database Tables

Coupling with database tables also needs to be discussed (see Figure 2.37) in the context of the maintenance of the variant tables. The background for this function is the requirement to also address database tables in object dependencies. However, this is not directly possible in object dependencies. There are two options for addressing database tables in object dependencies.

- **Directly addressing a database table (variant function)**
  You create a variant function and address the database table in the function module that corresponds to the variant function. The variant function is used in object dependencies.
- **Indirectly addressing a database table (coupled variant table)**
  You create an appropriate variant table and couple it with the database table. If the coupling is active, you can directly include the variant table in any type of object dependencies. Then the database table is also addressed.

The second option uses an assignment of the database table in the maintenance of the basic data of the variant table structure. You can also distinguish between the following two scenarios.

1. Scenario 2—Starting Point Database Table

An existing database table is to be addressed using this approach. You must create the appropriate characteristics, that is, characteristics with the appropriate format, in this scenario. You have to create characteristics only for the fields (that is, the columns of the database table) that are to be addressed in the object dependencies. Then you create a variant table that contains exactly these characteristics as columns. You must include the name of the database table in the basic data of the variant table. This allows for a field assignment. All characteristics of the variant table are linked to the columns of the database table. Finally, you must activate the coupling. Afterward, you can evaluate the database table in all types of object dependencies by addressing the coupled variant table in the syntax.

2. Scenario 2—Starting Point Variant Table

An existing variant table that may already be used in object dependencies is to be converted into a database table. This scenario is used, for example, when variant tables reach a size that may lead to performance problems. However, there may also be other reasons.

In this case, you need to create an appropriate database table. With regard to the format, the fields of the table must be created in such a way that they can include at least the values of the corresponding characteristics. The key fields of the database table are not relevant for Variant Configuration. Only the key combination of the variant table is critical for the evaluation of object dependencies. Of course, you must select the database table key in such a way that uniqueness is ensured for later content.

Analogous to the first scenario, you must now couple the variant table with the still empty database table, activate it, and assign the corresponding fields. Finally, the content of the variant table is transferred to the database table using Transaction CU59. In this second scenario, you don't have to adapt all object dependencies that use this variant table. They have exactly the same function as before. Bear in mind that the content of the variant table is inactive when the coupling has been activated.

In the standard version, you maintain the content directly in the database table. You can lock the content maintenance of the variant table via the status. You can also delete the content of the variant table to prevent misunderstandings.

Additional maintenance in the variant table with a delta transfer of this data to the database table is also possible. However, this isn't very useful because you cannot delete rows when implementing a delta transfer, and problems may occur due to the key of the database table.

2.6.4 Constraints in Detail

As already mentioned, you can use all types of object dependencies for the configuration in the value assignment interface. Constraints are the most important and default type of object dependencies, because they can be used for nearly all tasks. You can therefore perform the following tasks using constraints:

- Setting values
- Checking values (in consistency checks)

- Addressing any objects that are also in the multilevel configuration (not only `$self`, `$parent`, or `$root`)
- Working with a high performance level
- Working in a declarative way (this means you don't have to consider a processing sequence or similar factor in the modeling process)

Constraints are collected in dependency nets (also referred to as constraint nets). Dependency nets are generally assigned to the configuration profile. It doesn't matter whether a large number of constraints are assigned to a dependency net. However, the number of dependency nets in a configuration model should be kept to a minimum, even though you cannot always ensure that only one dependency net is assigned to the configuration profile of the header material of the configuration profile.

[Ex]

Need for Multiple Constraint Nets

If, for example, individual configurable assemblies of the overall model are sold separately, all constraints that are related to this assembly must also be assigned to the configuration profile of this assembly via a dependency net.

Within the dependency net, the constraints are local object dependencies; the dependency net itself is global. Note that both have a status. Constraints are consequently active only if the following conditions are met:

- The constraint is released.
- The dependency net is released.
- The dependency net is assigned to a configuration profile that is considered during configuration.

A constraint consists of at least two and at the most four sections. Objects and restriction are the mandatory sections here. The following explains the four sections according to their sequence:

1. **Objects**
   In this section, you enter all used classes and objects and define variables.
2. **Condition**
   This section is only used to (optionally) specify a central condition under which the constraint is supposed to be evaluated.

3. **Restriction**
   In this section, you define equations, tables, and functions for inferences of values and/or value checks.
4. **Inferences**
   This section enables you to (optionally) enhance the inference and restrict characteristic values.

The objects section is mandatory and must contain the objects that are addressed in the constraint. Objects can be classes, material masters, and documents. The declaration has the following syntax:

- **Class**
  (<class type>)class name, for example, `(300)T_VPC`
- **Material master**
  (Material)(<class type>)(Nr = '<material number>'),
  for example, `(Material)(300)(Nr = 'T-VPC')`
- **Document**
  (Document)(<class type>)(Type = '<document type>', Version = '<document version>', Part = '<document part>', Nr = '<document number>'),
  for example, `(Document)(017)(Type='DRM',Ver-sion='01',Part='000',Nr='T-VPC')`

In the objects section, you can also locally define variables for the constraint; the constraint in Figure 2.33 uses such an object variable. Object variables for classes are declared by `is_a`, and for all other objects by `is_object`, as shown in Figure 2.33. The constraint in Figure 2.33 would be as shown is Listing 2.10 without variables.

```
objects:       (300)t_vpc
restrictions:  table tab_vpc (  t_pc6r = (300)t_vpc.t_pc6r,
                                t_pc8r = (300)t_vpc.t_pc8r,
                                t_pc3r = (300)t_vpc.t_pc3r).
```

Listing 2.10 Sample Constraint: Table Call without Using Variables

Note that in this context you can also declare the variant class instead of the material master in the objects sections, as done here. After the equals sign, the syntax is as follows: `<object>.<characteristic>`.

In the objects section, you can also define characteristic variables, which are class-specific. You therefore don't have to use object variables here. In the syntax, this is implemented with where and a list of the variables separated by a semicolon. The constraint in Figure 2.33 would be as follows with characteristic variables (Listing 2.11):

```
objects:      (300)t_vpc where  ?spe = t_pc6r ;
                                ?cas = t_pc8r ;
                                ?cpu = t_pc3r.
restrictions:  table tab_vpc (  t_pc6r = ?spe,
                                t_pc8r = ?cas,
                                t_pc3r = ?cpu).
```

Listing 2.11 Sample Constraint—Table Call Using Characteristic Variables

[+]

Specific Variable Names

As shown here, question marks are often used as variables so that they can be easily identified. You cannot use variables for characteristics from tables, that is, for columns.

If the objects section lists multiple objects, use a comma to separate them.

The restriction section is the second section that is mandatory in constraints. Here, you can carry out consistency checks. The constraint reports an inconsistency when the restriction section is not met. You can also use constraints to infer values. For this purpose, the equations must be solved for the value that is supposed to be inferred, or you must use the inference section. This is further detailed in the context of the inferences section below. The restriction section enables you to restrict values (see Figure 2.33). You can use variant tables and variant functions.

You can also work with conditions in the restriction section. The if syntax element is provided for this purpose. Additional syntax elements, such as then or else, are not available. As shown in Listing 2.12, first the statement and then the if condition is provided.

```
restrictions:
(300)t_vpc.m1 = 'a'    if (300)t_vpc.m2 = 'x1',
     false             if (300)t_vpc.m2 = (300)t_vpc.m3.
```

Listing 2.12 Restrictions with if Conditions

This restriction section contains two statements that are listed with a comma. The first statement consists of the assignment of the a value for the m1 characteristic. The second statement only leads to an inconsistency message if the if condition after false is met. The processing of the false syntax element basically generates inconsistency messages and can only be used in constraints.

In constraints, you can use a condition section. The condition section must always be inserted between the objects and the restriction section. It contains exactly one logical expression. The constraint is not processed until the condition of the condition section is met, which is very good regarding the performance. You can also work with variables and variant tables in the condition section, as Listings 2.13 and 2.14 show.

```
objects:   (300)t_vpc where   ?os = t_pc01;
                              ?hd = t_pc04,
           (300)t_vpr where   ?tr = t_pr02.
condition : specified ?hd.
restrictions : ?tr = ?os
```

Listing 2.13 Sample Constraint with Condition Section

```
objects:   (300)t_vpc where   ?spe = t_pc6r;
                              ?cas = t_pc8r;
                              ?cpu = t_pc3r.
condition:   table tab_vpc (  t_pc6r = ?spe,
                              t_pc8r = ?cas,
                              t_pc3r = ?cpu).
restrictions: false.
```

Listing 2.14 Sample Constraint with Table Call in the Condition Section

In Listing 2.13, the restriction section is evaluated under the condition that a value has been assigned to the t_pc04 characteristic. In this case, the system checks whether the same values have been assigned to the t_pc01 and t_pr02 characteristics. If no values have been assigned to t_pc01, the system copies the value assignment of t_pr02 to this characteristic.

In Listing 2.14, an inconsistency message is output if the (300)t_vpc variant class in the value assignment interface corresponds to a row of the tab_vpc variant table in the configuration. This is used if inconsistent value assignment combinations are collected in variant tables. You can also create such constraints with the table constraint wizard. In this case, you must select the Checking Inconsistent Combinations entry as the mode of action. In this context, have a look at Figure 2.32 in Section 2.6.2.

The inferences section, which is optional just like the condition section, is the fourth section of a constraint. This section is always the last section of the constraint and enhances the evaluation of the restriction section. This "enhanced evaluation" can refer to equations, variant tables, variant functions, and restrictable characteristics. The syntax of the inferences section is merely a list of characteristics.

Consequently, an equation, `V = L * W * H`, for example, in a restriction section without a subsequent inferences section is only evaluated for calculation in such a way that V is the product of L, W, and H. However, if the constraint has the following structure, the fourth value is inferred from any three values (see Listing 2.15).

```
objects:   (300)t_vpc where
                v = t_pc91 ;   l = t_pc92;
                b = t_pc93 ;   h = t_pc94.
restrictions:   v = l * b * h.
inferences:     v, l, b, h.
```

Listing 2.15 Sample Constraint with Equation and Inferences Section

Variant tables and functions for which more than one value assignment alternative is defined are additional examples (see Figure 2.34 in Section 2.6.3). If you don't use the inferences section here, the system can only evaluate the first key, that is, the first value assignment alternative. However, if the constraint has the following structure and if two additional value assignment alternatives exist for the first and third characteristics and for the second and third characteristics, the system can infer the remaining third characteristic from the table from any two characteristics to which values have been assigned (see Listing 2.16).

```
objects:   (300)t_vpc where   ?spe = t_pc6r ;
                              ?cas = t_pc8r ;
                              ?cpu = t_pc3r.
restrictions: table tab_vpc ( t_pc6r = ?spe,
                              t_pc8r = ?cas,
                              t_pc3r = ?cpu).
inferences: ?spe, ?cas, ?cpu.
```

Listing 2.16 Sample Constraint with Variant Table and Inferences Section

For usage with restrictable characteristics, have a look at the constraint in Figure 2.33 in Section 2.6.2.

Now that we've discussed constraints in detail, the following sections deal with the other types of object dependencies. As already mentioned, you can use all types of object dependencies to design the value assignment interface in sales and distribution.

2.6.5 Preconditions

You can use preconditions to disallow individual characteristic values or entire characteristics for the value assignment interface. If you don't use preconditions, values can be assigned to any characteristic of the value assignment interface in any sequence. You can select any value from the list of the allowed values—irrespective of the value assignment of other characteristics. In this context, you have to find answers to two questions:

1. What is supposed to be disallowed? That is, which characteristic or which characteristic value is supposed to be dynamically disallowed?
2. When is it supposed to be allowed? That is, when is the corresponding characteristic or characteristic value supposed to be allowed within the scope of the value assignment?

The storage location of the respective precondition answers the first question. The syntax answers the second question. For example, if the XYZ engine is only supposed to be offered for the sport version of a car configuration, a precondition must be assigned to the XYZ characteristic value of the characteristic for the engine selection (What is supposed to be disallowed?). The syntax contains the prerequisite that the sport version was selected for which the XYZ+ engine is allowed (When is it supposed to be allowed?).

What effect does it have when the `$self.version = 'sport'` precondition is assigned to the XYZ characteristic value?

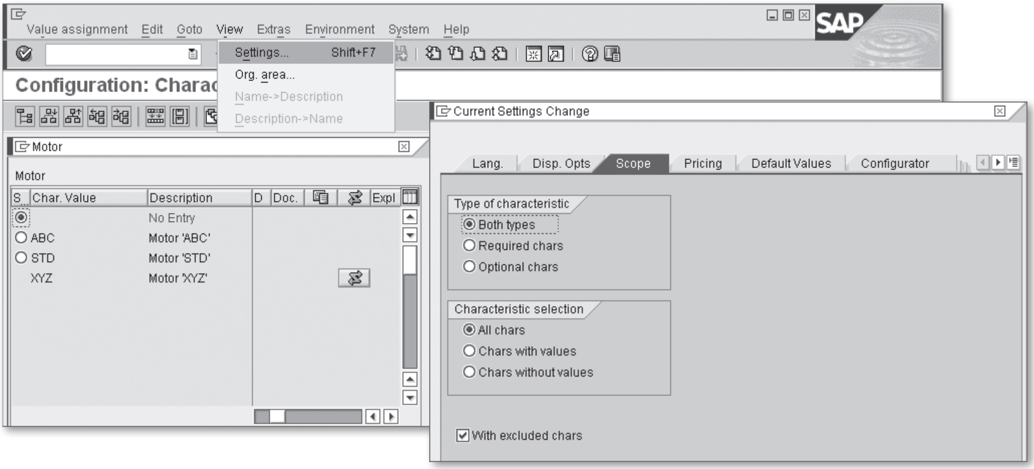
- **Assigning the "Sport" value to the "Version" characteristic**
  If the "Sport" value is assigned to the "Version" characteristic, the list of allowed values in the characteristic for the engine will provide all values as if no precondition exists.
- **Assigning a different value than "Sport" to the "Version" characteristic**
  If a different value than "Sport" is assigned to the "Version" characteristic, the "XYZ" value will be missing in the list of allowed values in the characteristic for the engine.
- **No value assigned to the "Version" characteristic**
  Note that the precondition is considered to be met if no value is assigned to the

"Version" characteristic. In this case, the list of allowed values of the engine characteristic would include all engine values. In the standard version all allowed values are initially available. During the value assignment, the system hides the values that are no longer allowed. If you don't want to use this standard logic, you must implement this by adding `$self.version = 'sport'` and `$self.version specified`. The system then first provides all values that can be generally selected, and the list of allowed values is gradually extended.

- **Sequence of the value assignment**
  The precondition mentioned in the previous item is elegant in one direction only: if values are assigned first to the version and then to the engine. If you start by assigning values to the engine, you can select any engine and any version. Only then is the precondition evaluated. It retroactively disallows the XYZ engine. This leads to an inconsistency message if the XYZ engine and a version other than sport are selected. You can avoid this by assigning a precondition to the sport version. Another option is to force a processing sequence. A value is not supposed to be assigned to the "Engine" characteristic until the version is known. In this case, you use a precondition for the characteristics. You assign a precondition of the `$self.version specified` (When is it supposed to be allowed?) to the engine characteristic (What is supposed to be disallowed?).
- **Multiple preconditions**
  You can also assign multiple preconditions to characteristic values or characteristics. In this case, the value assignment is only allowed if all preconditions are met. It can be considered an `And` link. You can only implement an `Or` link between preconditions when you include the conditions in a precondition. A precondition can be any complex condition using any brackets, negations, and concatenation with `and` and `or`.

Values or characteristics that are excluded via preconditions are not displayed in the value assignment interface by default. However, you can use the settings in the configuration profile (see Figure 2.38), for example, to define that disallowed characteristic values or characteristics are displayed but not used for the value assignment.

In Figure 2.38, the settings were called via the menu during the configuration. The WITH EXCLUDED CHARACTERISTICS checkbox was selected here. As a result, the disallowed value, XYZ, is displayed but cannot be selected. Similarly, disallowed characteristics would be displayed, but you couldn't assign values to them.



Motor

S	Char. Value	Description	D	Doc.	Expl
●		No Entry			
○	ABC	Motor 'ABC'			
○	STD	Motor 'STD'			
	XYZ	Motor 'XYZ'			

Type of characteristic
● Both types
○ Required chars
○ Optional chars

Characteristic selection
● All chars
○ Chars with values
○ Chars without values

☑ With excluded chars

Figure 2.38 Displaying Excluded Characteristic Values (Preconditions)

You can use variant tables in all types of object dependencies, that is, also in preconditions. The syntax of a precondition that uses the table from Figures 2.31 and 2.34 could be, as shown in Listing 2.17:

```
table TAB_VPC (   T_PC6R = $self.T_PC6R,
                  T_PC8R = $self.T_PC8R,
                  T_PC3R = $self.T_PC3R)
```

Listing 2.17 Sample Variant Table in a Precondition

This example has the same syntax as a procedure. Instead of `$self`, you can also use `$parent` or `$root` everywhere—provided that this is correct with regard to content. You could assign such a precondition with exactly this syntax to one of the three mentioned characteristics; however, this wouldn't be a clean solution.

Considering the increasing elegance of the solution, the following three options can be used to disallow values.

- Precondition for the characteristic (as described)
- Precondition for the characteristic value (as described later)
- A constraint with this variant table and restrictable characteristics (as the most elegant solution; see Figure 2.33)

If you selected the variant with the precondition for the characteristic, the precondition wouldn't affect the value assignment of the three characteristics. Not until values have been assigned to all three characteristics addressed in the variant table does the precondition become active and output an inconsistency if the table doesn't contain this value assignment combination.

However, preconditions for characteristic values are more elegant than this variant with a precondition for the characteristic but still not as elegant as the variant with a constraint. For example, if you assume that values are assigned to the characteristics in a fixed sequence and the T_PC3R characteristic is the last characteristic to which a value is assigned, you can provide preconditions for the characteristic values of this characteristic. The precondition for the value '03' would have the following syntax (see Listing 2.18):

```
table TAB_VPC (   T_PC6R = $self.T_PC6R,
                  T_PC8R = $self.T_PC8R,
                  T_PC3R = '03')
```

Listing 2.18 Sample Variant Table in a Precondition for Characteristic Value '03'

Analogous preconditions also apply to the other values of this characteristic. Compared to the first variant—that is, the precondition for the characteristic—this has the advantage that only the values that lead to a consistent value assignment are provided for the third characteristic. Preconditions for characteristics or characteristic values don't require restrictable characteristics.

2.6.6 Selection Conditions

Selection conditions can dynamically convert optional characteristics into required characteristics.

[+]

Optional Characteristics

This technology requires that the ENTRY REQUIRED checkbox in the characteristic is not set, that is, the characteristic is actually a so-called optional characteristic.

If you assign a selection condition to such a characteristic, the characteristic dynamically becomes a required characteristic if the condition is met. Let's illustrate this with an example.

[Ex]

Sport Version Requires a GPS

For the car configuration, the selection of a GPS needs to become a prerequisite for the sport version. In this context, the rules and the syntax must be analogous to the preconditions. The selection conditions must be assigned to the GPS characteristic (What is required?); the content of the syntax is the condition of the sport version (When is it required?). The syntax is analogous to the precondition specified previously: `$self.version = ’sport’`. The selection condition with this syntax is assigned to the GPS characteristic. This way, the selection of a GPS is required for the sport version.

Compared to the rules of preconditions, there are two differences, which must be considered:

- **Multiple selection conditions for a characteristic**
  If multiple selection conditions are assigned to a characteristic, it is sufficient for the characteristic to become a required characteristic when one selection condition is met.
- **No value assigned to a characteristic in a selection condition**
  If a selection condition addresses characteristics to which no value has been assigned, the selection condition is not met. In contrast to a precondition with the same syntax, the selection condition above outputs “false,” that is, not met if no version is selected.

2.6.7 Procedures

Procedures are object dependencies that set values. In contrast to other types of object dependencies, they can affect processing sequences. For this purpose, the configuration profile is assigned with procedures, and a sorting is transferred. There is also the option of assigning procedures to characteristics and characteristic values. However, this method is not supported for the configurator of the IPC and is consequently not discussed here. Instead, we'll discuss in detail the assignment to the configuration profile. Procedures allow for *fixed* or *dynamic* assignments of values. The following list describes in detail what this means:

- **Fixed assignment of values**
  A fixed assignment of values refers to an assignment of values that cannot be overwritten by users or constraints. Attempts to do so would result in inconsistencies. Procedures can also not overwrite *external* assignments of values, that is, values set by users or constraints. The same applies to users and constraints.

Index